HOME OF GAMES

UNITYSCRIPTING - BOOKS

CANNON OF LIGHTS GAME 2D

محتوى الكتاب يعمل على النسخة UNITY 5.1.2F1

المطور و المؤلف: عماد عارف التوي – YEMEN, ADEN

[ورشة عمل لعبة CANNON OF LIGHTS 2D]

بسم الله الرحمن الرحيم.

السلام عليكم ورحمة الله وبركاته.

{اللهم علمنا بما ينفعنا و انفعنا بما علمتنا و زدنا علماً انك انت العليم الحكيم}

كتاب ورشة عمل لعبة Cannon of Lights هذا الكتاب يناقش طريقة عمل لعبة 2D بسيطة في محرك Unity3D، هذا المحرك المشهور و المعروف بسهولته وقوته في صنع الالعاب و كل لعبة تختلف في تعقيدها و بنائها حسب نوعها، فمحرك اليونتي قادر على صنع اي لعبة و لكن هذا الشيء يعتمد على قدراتك في صنع الالعاب على هذا المحرك او غيرة، في هذا الكتاب اردت مناقشة طريقة عمل هذه اللعبة التي كلفتنا الكثير من الوقت و الجهد لكي اكملها بشكل صحيح بأذن الله، واجهتنا الكثير من المشاكل في عملها و الحمد لله استطعت التغلب عليها، و هي الهمتنا بشكل كبير و ساعدتنا على دراسة هذا المحرك، ليس من باب تصميم لعبة فقط بل التعرف على الطريقة التي تفكر بها الكائنات و مبدا ان كل شيء جماد و عليك بث الروح فيه، اليوم اشارككم اجمل اللحظات و اصعب اللحظات في هذا الكتاب خلف الكواليس (صنع اللعبة).

ان هذا الكتاب مجاني ولا يحق لأي احد بيعة بأي طريقة كانت، فكل ما اطلبه منكم هو الدعاء لي و لوالدي و لجميع المسلمين، و اتمنى ان تمروا على جميع فقرات هذا الكتاب و في حالة وجدت مشكلة في تطبيق شيء ما فأرجوا التواصل معي على بريد قناتي Home OF Games، ايضاً ادعوكم للدخول الى مدونتي emadarif.blogspot.com لكي تتابعوا كل جديد في اليونتي، الى هنا اتمنى لكم قراءة جميلة و اتمنى ان تمروا على جميع الفقرات و اعلامي بالسلبيات و الايجابيات او الملاحظات او اي شيء آخر ☺.

محتوى الكتاب:

المؤلف:-

الكتاب: عماد عارف التوي.

البلد: اليمن، عدن.

نبذة : مصمم 3D، كاتب و مبرمج العاب في محرك Unity3D من عام 2012، اعمل على دراسة هذا المحرك من جميع الجوانب و خاصةً الجانب البرمجي بشكل كبير، احاول تقديم الدروس التفصيلية للجانب البرمجي اكانت بعمل كتب او كورسات (فيديوهات) لمشاركتكم كل جديد في عالم برمجة و تطوير الالعاب ككل.

المدونة: Home OF Games – Dev Games.

القناة على اليوتيوب : Home OF Games

البريد: homeofgamesnews@gmail.com

كتب أنشأتها سابقاً، يمكنك مشاهدتها Online و تحميلها:

- **تصميم و برمجة لعبة Cube Bird.**
- **تعلم برمجة الالعاب في Unity4 UnityScripting(part 1).**
- **تعلم برمجة الالعاب في Unity4 UnityScripting(part 2).**

المقدمة:-

كتاب UnityScripting يعتبر ضمن احد الكتب التي قمت بعملها في السابق و التي تهدف الى شرح البرمجة في محرك Unity3D من الاساسيات، ايضاً قمت بعمل كتاب آخر بعنونا (تصميم و برمجة لعبة Cube Bird) و الذي ناقشت فيه عمل لعبة 2D بسيطة باستخدام ابسط و اسهل الأكواد، بالنسبة لهذا الكتاب فأنه يشرح عمل لعبة 2D شبة ما اقول عنها انها معقدة قليل فهي تحتوي على اكثر من مرحلة و تتضمن اشياء عديدة بالإضافة الى عمل قائمة رئيسية و مصغرة و اشياء كثيرة اخرى سنناقش كل منها في فقرة خاصة و سنفصلها بأذن الله.

في البداية سنتحدث قليلاً عن اللعبة ،اللعبة هي عبارة عن مدفع يقوم بحماية المدينة من المركبات الفضائية التي قامت بتعطيل الكهرباء من المدينة. هنا يقف مدفعنا الوحيد امام رحمة المركبات الفضائية التي تطلق الليزر ، القنابل و الصواريخ ،و مهمة المدفع هو الوصول الى نتيجة معينة لكي تعمل اضواء المدينة في حالة اتممت المهمة بنجاح ،او ينفجر المدفع في حالة لم تتممها.

من هذا الشرح سنقوم بعمل العديد من الاشياء لصنع لعبة متكاملة سنناقشها في الفقرات القادمة ،لنتحدث اكثر عن اللعبة ،اولاً اللعبة تحتوي على 6 مراحل و في كل مرحلة تواجه عدو جديد او تزيد صعوبة المستوى ،ايضاً سيحصل المدفع على طلقة جديدة لمواجهه الاعداء الاقوياء سنخصص فقرة لمناقشة الطلقات و مدى تأثيرها ،اما المرحلة الاخيرة ستواجه فيها زعيم الاعداء (Boss) ومهمته هي توليد مركبات اصغر و لك توسيع الامر لأننا سنجعل اللعبة قابلة للتوسيع نظراً لميكانيكية اللعبة و معماريتها ،و بما انها ستكون لعبة متكاملة ،سيتوفر فيها قائمة رئيسية (main menu) و قائمة مصغرة (min menu) تتواجد داخل المرحلة لتوفير سهولة في التحكم بالإعدادات و سنناقش كل قائمة في فقرتين، ايضاً ستتواجد العديد من الفقرات التي ستغطي كل ما تحتاجه اللعبة لتكوينها.

ان تصنيف هذه اللعبة يقارب الالعاب من نوع Beat 'em Up وفي هذه الانواع من الالعاب تركز على القتال و تصويب الاهداف وغالباً هو قتل جميع الاعداء الى ان تصل الى زعيمهم، او المرور على منطقة مليئة بالأعداء الى ان تصل الى زعيمهم وتنهي اللعبة.

ربما اخذت فكرة اولية عن اللعبة و سنترك الشروحات تحدد مستوى اللعبة و لكن يجب ان تمر على الفقرات الاولية لكي تأخذ فكرة اكبر وتوسع معرفتك في اللعبة و طريقة بنائها.

وثيقة تصميم اللعبة GDD:

Game Design Document باختصار GDD. ان وثيقة تصميم اللعبة اعتبرها المفتاح الانسب لبدأ العمل على هذه اللعبة التي قد ربما تضن انه ليست عامل مهم بسبب حجم اللعبة لكن ستعرف و مع الوقت ان الوثيقة ضرورية لسير عملية التطوير على افضل لحال و لكي لا نتوه في العمل و لكي لا يكون مصيرها مجهول، الوثيقة بشكل عام هي عبارة عن الافكار التي تدور في رأسك بخصوص لعبتك، قمت بإضافة شيء جديد للعبة، او وضعت ملاحظات او افكار جديدة هذا اهم شيء كبداية اولية ووثيقة التصميم ستساعدك على ترتيب افكارك ووضعها في المكان المناسب و حساب الوقت و التكلفة و غيرها، فهي الوثيقة التي سترميها في الهواء بعد اكمال جميع اقسامها، ان وثيقة التصميم مهم جداً ففي حالتنا هذه سنتعامل مع العديد من الاشياء، ايضاً الوثيقة لا تقتصر على تسيير امور التصميم فقط بل البرمجة ايضاً فهي الشيء المعقد و الذي سيأخذ الكثير من الوقت لعمل هذه اللعبة.

في العديد من الحالات يتم تقسيم الوثيقة الى جزئيين وهما: وثيقة الصفحة الواحدة (التي تختصر فيها مفهوم اللعبة و ميكانيكية عملها بشكل عام) ووثيقة العشر صفحات (التي تقوم فيها بتفصيل كل شيء عن اللعبة) و لكن ليس كبداية تقوم بوضع كل شيء فيها، بل ابقيها على تحديث دائم و عند عدة نقاط، ان لم تقم بعمل ترتيب معين لما ستقوم بالعمل علية و تقوم بتفصيله فهذا الشيء مكلف حقاً و قد تستغرق وقتاً طويلاً في صيانة هذا الوثيقة و اعادة كتابتها، هناك طرق اسهل لترتيب نموذج العمل على اللعبة، حسناً لنقم بوضع اول الخطوات و هي ترتيب الوثيقة كبداية اولية، ستكون كالاتي:

ميكانيكية العمل الاولية (الترتيب المباشر)

- **بناء المشهد.**
- **تصميم الشخصيات (اللاعب، الاعداء ...الخ).**
- **المؤثرات المرئية.**
- **المؤثرات الصوتية.**
- **قوائم اللعبة.**
- **برمجة اللعبة.**
- **بناء اللعبة.**

كبداية ستجد ان الامر سطحي جداً، ليس هناك تغيير، فقط عبارة عن ترتيب عادي، اذن دعنا نناقش الامر اكثر و نوضح مفهوم العمل لان هذه الوثيقة تعتبر نقطة اساسية لبناء اللعبة و يجب صيانتها بقدر المستطاع.

لاحظ انا تصميم المراحل و الشخصيات هما امران مهمان في الوقت الحالي، ان عمل اللاعب و الاعداء وتصميم اول مرحلة من خلالها تستطيع وضع تجاربنا عليها، و غالباً لا يتم العمل على جميع المراحل في نفس الوقت بل العمل على مرحلة واحدة و من ثم استنساخها وهذا ماسنطبقه في هذه اللعبة مع اختلافات بسيطة سنجريها في كل مرحلة، هنا لن تكلف نفسك عناء رسم الشخصيات و المراحل فكلها موجودة في الرابط و في حالة كان هناك شيء ناقص فبكل بساطة يمكنك الاستعانة بالمواقع المذكورة في الاعلى، للتوضيح ان العمل سيكون برمجي بشكل كبير، حسناً اجد ان الترتيب الحالي لابأس به لكن لنقم بفرزة اكثر، سنعتمد على النجوم لفصل كل شيء و نعرف على ماذا يعتمد هذا الشيء، طريقة النجوم توضح ماهي الاشياء الهمه والتي يجب العمل عليها كبداية، لاحظ هذا التلميح:

*** نجمة واحدة: ان هذا الشيء غير مهم حالياً.**
**** نجمتان: هذا الشيء يعتمد على انجازات سابقة.**
***** ثلاث نجوم: لن نتحرك حتى ننجز هذا الشيء.**

ارى ان هذا جميل، حسناً لنعد الى الوثيقة و سنفصلها كما ترمز له النجوم:

ميكانيكية العمل الاولية (الاشياء المهم و الغير مهمه اعتماداً على النجوم)

- تجهيز المشهد (العمل على مشهد واحد). ***
- اضافة الاعداء و اللاعب. ***
- تصميم المؤثرات المرئية. **
- اضافة المؤثرات الصوتية. **
- تصميم قائمة العرض. **
- تصميم القائمة المصغرة. **
- تصميم قائمة النجوم. *
- تصميم القائمة الرئيسية. **
- برمجة اللعبة. *
- بناء اللعبة. *

لاحظ ان العملية البرمجية باتت آخر همومنا، طبعاً ليس في هذه اللعبة لان كل شيء جاهز ولم يتبقى الا تركيب الاشياء وبرمجتها، ان كل شيء في الوثيقة متوفر (الشخصيات، المراحل، القوائم، المؤثرات المرئية، المؤثرات الصوتية) و ان لم يتوفر شيء فعليك تنزيله من النت، حسناً و بما ان كل شيء بات متوفر سيتبقى لنا تفصيل العملية البرمجية اي ماهي الاشياء التي يجب علينا عملها كأول خطوة و الاسباب سنذكرها في وثيقة العشر صفحات، العملية البرمجية ستتفرع كالاتي:

العملية البرمجية (الاشياء التي يجب برمجتها بالترتيب)

- برمجة اللاعب Player
- برمجة الاعداء Enemies
- ربط المؤثرات المرئية بينهم Particle System
- برمجة المؤثرات الصوتية
- برمجة قائمة العرض.
- برمجة القائمة المصغرة Min Menu
- برمجة قائمة النجوم Menu Stars
- برمجة القائمة الرئيسية Main Menu

و الى هنا اجد ان الترتيب الحالي كافي و ربما يكون ملل لأنه لم يتم الدخول الى البرمجة، في النهاية هذه الوثيقة مهمه ويجب الاستفادة منها و قراءتها و التدقيق فيها، الى هنا و نتوقف عند هذا القدر من التقسيم و ربما نقسمها لاحقاً و سنبقي الوثيقة على تحديث دائم فكل ما عليك فعلة هو اضافة افكارك و ترتيبها كما هو موضح في التقسيم.

وثيقة الصفحة الواحدة:

في فقرة وثيقة الصفحة الواحدة سنناقش طريقة تصميم اللعبة بشكل مختصر لكي تأخذ فكرة اكبر عن طريقة تصميم اللعبة كبداية اولية الى ان نقوم بتفصيل شرح تصميمها بشكل مفصل، دعنا نبدأ هنا بحيث سأعطيك فكرة عن طريقة تصميم اللعبة و من اين نبدأ بالضبط.

بداية تصميم لعبتنا ستعتمد على تنزيل الملف المضغوط الذي يحتوي على خلفيات اللعبة، كبداية سنقوم بوضع و ضبط الخلفيات بشكل مباشر و سنعمل على مرحلة واحدة و سنضيف فيها كل شيء تقريباً الى ان نصل الى برمجة الانتقال بين المراحل و حفظ آخر مرحلة وصلنا اليها سنقوم بتصميم المرحلة الثانية و باقي المراحل لن تختلف في شيء عن المرحلة الاول في اللعبة، بل هي عبارة عن شكل جديد او تغيير طفيف في الخلفية و لك توسيع الامر حقاً.

ان العمل الاولي سيكون على المدفع و الذي بدورة سيكون اللاعب Player، هذا المدفع له العديد من الوظائف و اهمها هي الحركة، سنعتمد على ثلاثة ملفات الاول لوضع ذكاء المدفع بشكل عام و هذا يشمل الدوران اما الثاني لأطلاق النار و الثالث سيكون خاص بالعجلة، بالنسبة للمركبات الفضائية علينا ترتيبها بشكل كامل (مركبة مركبة) ووضعها على شكل prefabs هذا الامر ضروري في اي لعبة، اما بالنسبة للجانب البرمجي سنقوم بعمل ملف برمجي واحد يحتوي على ذكاء الاعداء و معلومات الصحة و الاطلاق EnemiesInfo وهو الذي سيتحكم بحركة الاعداء و عمل المسافات و غيرها، بالنسبة للأصوات هنا اجد ان وضع جميع الاصوات الضرورية في اللعبة داخل كائن واحد parent و يتفرع منه عدة كائنات children الابناء، و كل كائن ابن يحتوي على صوت خاص لكل شيء يحتاج صوت و هذا يشمل موسيقى الخلفية بكل تأكيد.

قوائم اللعبة سيكون عبارة عن 5 ازرار و كل زر حسب وضيفته، ايضاً سيتفرع من بعض الازرار عدة قوائم و التي تخص الاعتمادات المطور، اعدادات الجودة و تحديد المراحل، الجانبي البرمجي للقوائم سيكون عبارة عن ملف واحد باسم MenuManager و الذي بدورة سيحتوي على العديد من الدوال و المتغيرات التي ستساعدنا على الانتقال من مرحلة الى اخرى ايضاً التعامل مع القائمة المصغرة بشكل مباشرة و سريع و مرتب بأذن الله.

بالنسبة للمؤثرات المرئية فبكل بساطة سنعتمد على انشاء مؤثرات خاصة بنا عبر Particle System الذي سيوفر علينا الكثير و سنقوم بربطة بشكل مباشر في البرمجة و سنتعرف على عدة طرق لإضافة المؤثر المرئي لكل شيء يحتاجه في اللعبة.

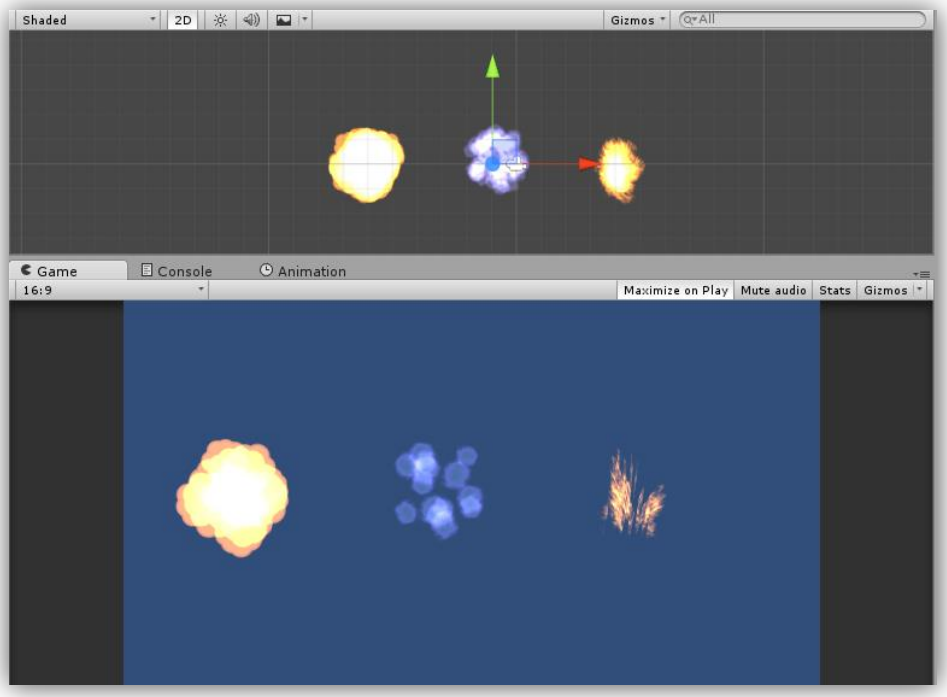
وثيقة العشر صفحات:

في وثيقة العشر صفحات سأفصل مفهوم العمل على اللاعب، الاعداء، المؤثرات المرئية، المؤثرات الصوتية، قوائم اللعبة بشكل عام، مراحل اللعبة و بناء اللعبة، الامر لن يكون مفصل بشكل دقيق لكن سأعطيك فكرة عما يجب عملة في اللعبة كبداية اولية.

بالنسبة للاعب فهو العامل الرئيسي لبدأ العمل على اللعبة اي برمجة و تصميم اللاعب هو امر مهم هو الشيء الوحيد الذي تستطيع وضع جميع التجارب عليك، ما فائدة الاعداء دون اللاعب؟ ايضاً المراحل؟ المؤثرات المرئية و كيف ستظهر دون تأثير اللاعب بجو اللعب؟ هناك اشياء عديدة يمكن ذكرها عن اللاعب لأنه العمل المهم هنا، برمجة اللاعب ستعتمد كبداية على برمجة موقع العجلة مع موقع المدفع هذا الربط العجلة بالمدفع و يتم تحريك العجلة بخواصها الفيزيائية Rigidbody، عندما ترى اللاعب ستلاحظ انه عبارة عن جزئيين فقط (العجلة و المدفع) و يجب ربطهما معاً و ليس من الضروري ان تكون العجلة اب للمدفع و العكس بل يمكن وضعهم كل منهم لوحدة.

الاعداء، بالنسبة لبرمجة الاعداء فأنهم معتمدين على اللاعب في هذه اللعبة، كل عدو سيتجه الى اللاعب و علية توجية مدفعه الية ايضاً اطلاق النار و تختلف في حالات كانت ليزر او صواريخ او يمكنك وضع طلقة خاصة فيك، في حالتنا هذه سنناقش ٣ طلقات للمدفع و هم طلقات الليزر، القنابل و الصواريخ، في حالة لم تكتفي ستضع طلقة اخرى و عدو آخر بكل بساطة، ذكاء سيقسم الى قسمين، الاول التحرك و الثاني اطلاق النار و كل منهم سيعتمد على ملف منفصل، بالنسبة لملف حركة الاعداء فهنا جميع الاعداء ستتشارك في نفس الملف و هذا الملف سيكون قادر على وضع الاعداء في سرعة مختلفة او متشابهه، ملف اطلاق النار سيكون منفصل كلياً عن الاخير بحيث ان لكل عدو سلاحة الخاص:

دور المؤثرات المرئية سيأتي عند التخلص من برمجة اللاعب و الاعداء فهنا يمكنك وضع المؤثرات المرئية بكل سهوله، لن تتطلب العملية منك الكثير فكل ما سنفعلة في المشروع هو عمل المؤثر المرئي و وضعة عند الاصطدامات و الانفجارات او اطلاق النار و غيرها بشكل مباشر.



الاصوات ستعتمد على عمل المؤثرات المرئية كما هو موضح، عملية ربط الاصوات ستكون لها طريقة معينة لكي نستطيع ايقافها و عاداة تشغيلها، فكر في وضع صوت في كل انفجار، هنا ستكون عملية ايقاف جميع الاصوات معقدة، لكن فكرة في ان كل كائن يلزمة صوت يقوم بالبحث عن الصوت الذي يريدة و يقوم بتشغيلة تلقائياً، هنا تستطيع ايقاف الصوت الاب و ستقف جميع الاصوات في نفس الوقت، لاحظ انك كلفت على نفسك عناء الكثير، فهذا ما سنستخدمة في لعبتنا لن نتطرق الى الاشياء المعقدة جداً، دعناً نبدأ بالاسهل و نقوم بتوسيع الامر لاحقاً.

ان قوائم اللعبة دائماً ماتكون الشيء الاخير الذي يتم عملة في اللعبة، صحيح لن نحتاج الى القوائم في الوقت الحالي لكن دعنا نتلكم اكثر عنها، قائمة لعبتنا لن تختلف كثير عن قوائم الالعاب، ستتضمن الازرار الاساسية Start, Select Level, Options, Exit و عندما تأخذ فكرة عن تصميم القوائمة ستجد ان وضع قائمة تتضمن التطوير هي امر سهل وبسيط، القوائم في اللعبة ستكون عبارة عن نوعين، القائمة الرئيسية وهي تتضمن الازرار الرئيسية و التي تقوم عليها اللعبة وهي ستساعدك على التحكم باللعبة و جودتها و معرفة اكثر عنها، قائمة مصغرة و التي تتواجد في داخل اللعبة و ستتيح لك ايقاف اللعب و استئنافة ايضاً ايقاف الاصوات و اعادة لعب المرحلة، شيء ثانوي و هي قائمة الانتقال بين المراحل و التي تظهر ٣ نجوم، تستطيع عملها و هذا الشيء يرجع لك، لكن سنناقشها في هذا الكتاب، و في النهاية جميع القوائم ستعتمد على ملف واحد وهو MenuManager و الذي سيتضمن جميع الوظائف (تشغيل اللعبة، انتقال بين المراحل، اعادة اللعبة، الخروج و غيرها) اما بالنسبة للجودة فأنها ستعتمد على ملف منفصل، و يفضل ان تكون في ملف منفصل.

بالنسبة لمراحل اللعبة فهيا الامر المهم في اللعبة، لاتعتمد على ان كون اللعبة هذه ستحتوي على المتعة المطلوبة في جميع مراحلها بل اجعل هذه اللعبة هي مدخل لعالم تطوير الالعاب، حالياً في لعبتنا ستحتوي على ٦ مراحل هنا لن اكرر ماقلتة سابقاً بالحرف، عملية لعبتنا ستكون في اول مستوى و الذي سيكون مختبر التجارب في المشروع سنحاول تحسين المشهد لكي نستنسخة لاحقاً، هذه العملية لن تكلفنا الكثر و سنختصرها بسرعة.

نأتي اخيراً على بناء اللعبة، ان عملية بناء اللعبة لن تستغرق الوقت الطويل و لكن هذه النقطة لاتعني بناء اللعبة بهذا الشكل بل فحصها ثم بنائها بشكل كامل، قد تستغرق هذه العملية الوقت و ستضطر لمراجعة جميع ملفاتك و التأكد من ان كل شيء بات في مكانة الاصلي، هنا ستبقى الوثيقة كماهي و لن تكون قابلة للتحديث بحيث تعيد ضافة اشياء غير متوقعة و تعيد ترتيب اللعبة و الوثيقة بشكل كامل، هذا لايعتمد على فحص اللعبة داخلياً بل ايضاً يجب تصديرها على شكل اختبار لمدى صلاحية اللعبة للعبو التأكد من عدم وجود اخطاء برمجية او ماشابة، في النهاية عملية بناء اللعبة لن تتم حالياً و سنأجلها الى فقرتها التي سنناقش فيها طريقة بناء اللعبة بشكل مفصل بأذن الله.

ان وثيقة العشر صفحات لن يتم تفصيلها بشكل كامل هنا، سيكون الامر ممل نظراً للكلام الجاف و المعنى المنعدم دون الطبيق لكن سنبقي الوثيقة في حالة تحديث دائم، انه لامر صعب ان نقوم بتفصيل الوثيقة بشكل كامل هنا، هذا الامر صعب فعلاً لهذا دعنا نبدأ بتصميم اللعبة حسب خطة الوثيقة الاولية و التي سنبدأ بها حساب التقسيم الذي وضعناه في البداية، ليس هناك وثيقة تصميم مثالية تاتي من اول فكرة اخترعتها، بل بأبقاء الوثيقة في تحديث دائم ربما تقوم بصنع وثيقة التصميم المثالية للعبتك.

وصف اللعبة:-

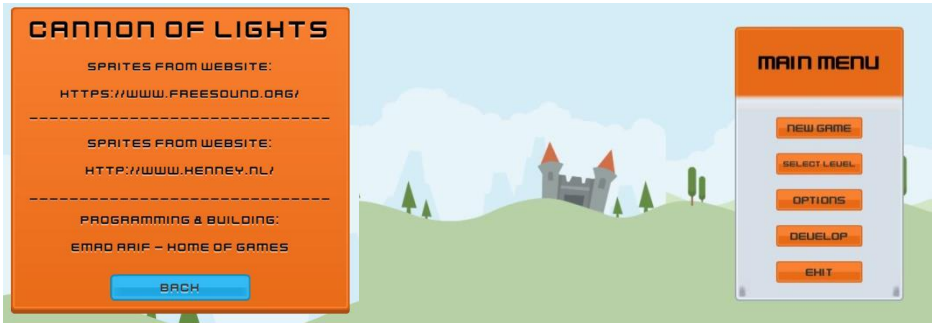
نوع اللعبة عبارة عن لعبة 2D و يمكنك نقلها الى اي منصة تريدها، قصة اللعبة بشكل بسيط هو ان هناك مركبات فضائية قامت بغزو كوكب الارض و قامت بتدمير الكهرباء في المدينة، هنا جاء هذا المدفع ليعيد تشغيل الاضواء في المدينة، في مراحل اللعبة ستواجه ثلاثة انواع من الاعداء و كل عدو لذي قدرة قوية ايضاً سلاح اقوى و درع اقوى و في كل مرحلة ستحصل اما على طلقة جديدة او ستواجه عدو جديد او ستزيد صعوبة المرحلة، تنتهي اللعبة عندما تصل الى عدد معين من الـ Scores نتيجة قتل الاعداء.

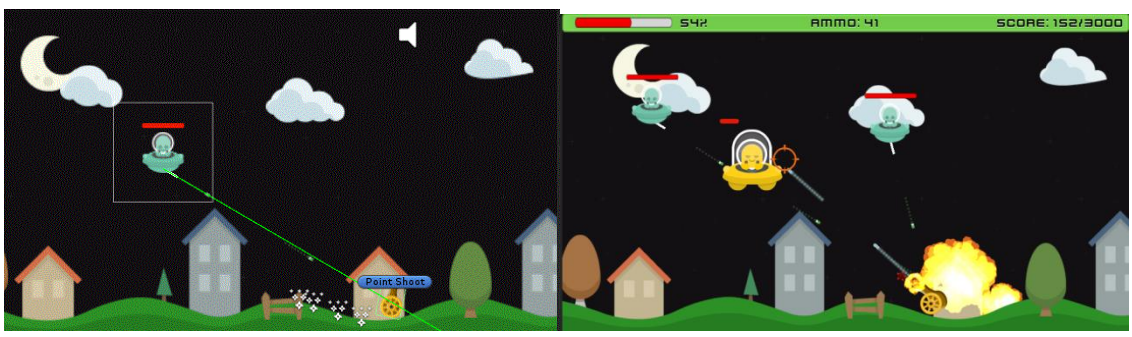
اللعبة ستحتوي على 6 مراحل و في كل مرحلة تزيد الصعوبة و هكذا الى ان تصل الى زعيم الاعداء Boss، زعيم الاعداء عبارة عن عدو ضخم يقوم بتوليد المركبات الفضائية الاصغر حجم تقوم بتشتيتك عن قتال الزعيم، بالنسبة لشكل فهو عبارة عن عدو ضخم محمي بشكل كبير من جميع النواحي، في حالة اردت تتغير شكل العدو فبإمكانك و بكل بساطة تغيير شكلة و هذا لن يضر اللعبة لان اللعبة ستكون مبنية على معمارية برمجية تساعدك على تغيير اللعبة بشكل اسرع و إضافة مراحل اكثر و اضافة اعداء ايضاً التحكم بظهور الاعداء و مدى ضرر الطلقات و صحة الاعداء و غيرها من الاشياء التي سنمر عليها في الشرح.

ملخص اللعبة:

- **نوع اللعبة:** (Beat 'em Up) و في هذا النوع من الالعاب دائماً يحتاج اللاعب الى اسلحة ليتمكن من القضاء على الاعداء.
- **المنصة المستهدفة:** Windows Desktop.
- **الفئة العمرية:** متاحة لجميع الاعمار for Everyone.
- **لعب اللعبة:** يواجه مدفعنا المركبات الفضائية و كل مركبة قوية ستظهر في مرحلة معينة و ستكون لديك طلقة تساعدك على قتال هذه المركبات القوية الى ان تصل الى زعيم الاعداء Boss فهل ستستطيع القضاء علية؟.
- **للاعب:** عبارة عن مدفع يتحرك بشكل افقي وله منطقة تحرك نحددها اعتماداً على مشهد الكاميرا، ايضاً ستحتاج الى الماوس لتوجيه المدفع و قتل المركبات الفضائية، سيكون لذى هذا اللاعب صحة تبقية موجوداً في المشهد و عليك الحفاظ عليها والا خسرت.
- **الاعداء:** لدينا 3 انواع من الاعداء بحيث ان كل عدو يختلف عن الاخر بالطلقات و الذرع و السرعة و هذا لا يعني ان كل عدو اقوى ستضل سرعته كبير، لا بالعكس و سيتحتم عليك الهروب منة لان سيتوفر لدية طلقة اقوى من طلقتك.
- **زعيم الاعداء:** ان زعيم الاعداء سيكون عبارة عن مركبة كبيرة الحجم، وظيفة زعيم الاعداء هو توليد المركبات الفضائية لهذا عليك ان تقوم بتدميره و لكن ستقوم المركبات الاصغر حجم بتشتيتك عن قتالة و بهذا الشكل تزيد صعوبة المرحلة الاخيرة.
- **عالم اللعبة:** كوكب الارض بحيث ستتوفر الجاذبية و بشكلها الطبيعي ولكن المدفع فقط من سيتأثر بها اما المركبات الفضائية ستطفو في الهواء.
- **الاضافات:** ان الاضافات هي اهم شيء في العبتنا هذه، سيتوفر لنا 3 انواع من الاضافات، الاولى تساعدك على حماية نفسك وهي عبارة عن درع حماية يقوم بتغطية المدفع و حمايته من الهجمات، الثانية زيادة في الطلقات (و هناك 3 انواع بحيث ان الطلقة العادية ستحصل على اضافتها من اللعبة تلقائياً عندما تصل طلقاتك الى مستوى اضعف، اما الثانية فسيتوجب عليك قتال المركبات السريعة و هي التي ستعطيك طلقات سريعة، الثالثة قنابل و هنا عليك قتال المركبة الضخمة للحصول عليها) الاضافة الثالثة وهي زيادة في الصحة و التي ستقوم بزيادة صحتك بشكل عشوائي و لكن سيتوجب عليك قتل المركبات الفضائية بشكل عام، في حالة اردت توسيع الامر فلابأس.
- **واجهه اللعبة:** قائمة رئيسية، قائمة مصغرة (ايقاف و تشغيل مؤقت، تحكم بالصوت، اعادة اللعبة و العودة الى القائمة الرئيسية)، قائمة النجوم وهي 3 انوع (ذهبية، عادية، خسارة)، اخيراً القائمة الرئيسية.

صور من داخل اللعبة:





نموذج آخر لوثيقة التصميم:

للعلم ان هناك عدة نماذج لعمل وثيقة تصميم مثالية و ليس بالضرورة ان تكون مثالية او ان تعتمد على فكرة شخص معين في رسم وثيقة تصميم للعبتك، و لكن من الجميل ان تأخذ فكرة اكبر عن وثيقة التصميم، بالنسبة لهذا النموذج فانة يعتمد على وضع معلومات اللعبة بشكل مباشر و الاشياء التي ستضيفها و اتاحة المجال لباقي فريق العمل لإضافة اي افكار جديد و هذا النموذج يقسم اللعبة الى 4 وظائف تقوم بإنجازها وهي **(معلومات اساسية عن اللعبة، نموذج اللعبة، مخطط سير اللعبة، اشياء اخرى)**، بحيث ان هذه النقاط ستساعدك على وضع افكارك بشكل مرتب وواضح، اذن العملية تسري كالاتي:

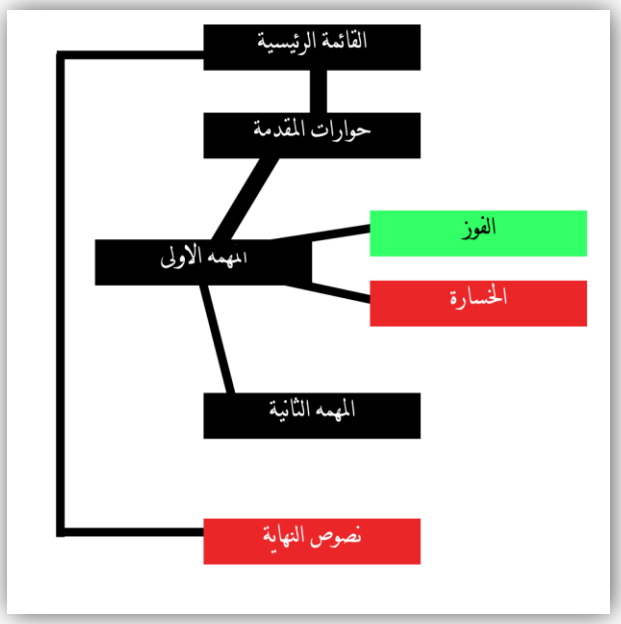
اولاً: معلومات اساسية عن اللعبة:-

- **اسم المشروع الرمزي:** Cannon of Lights Game 2D.
- **وصف مختصر عن المشروع:** لا يتجاوز 200 حرف
- **تاريخ بدأ العمل:** yyyy / mm / d
- **تاريخ الانتهاء من العمل:** yyyy / mm / d
- **محرك الالعاب المستخدم:** Unity3D.
- **المنصات المستهدفة:** الحاسب الشخصي (PC) نظام الويندوز.
- **العاملين على المشروع و ادوارهم:**

الاسم	الدور: (دور كل شخص في تصميم اللعبة)
عماد عارف التوي	برمجة اللعبة، بناء اللعبة.

نموذج اللعبة:-

- **الميكانيكية اللعب:** تعتمد على نظام لعب (Beat 'em Up)
- **سرد اللعبة:** و تقوم في هذه النقطة توضيح هدف اللعبة، القصة، الشخصيات، الرسومات، النصوص، الصوتيات و اخيراً الانظمة البرمجية المستخدمة.
- **مخطط سير اللعبة:** يفضل ان يكون على شكل مخطط مرسوم للتوضيح فقط:-



اشياء اخرى في اللعبة:-

- **اللغة:** الانجليزية فقط.

في الاخير و كل ما اريد توضيحه هو اهمية وثيقة تصميم اللعبة التي ستساعدك على رسم افكارك و رؤيتها قبل العمل على اللعبة و غالباً تستغرق عملية تصميم هذه الوثيقة اسبوع او اكثر و لكن يجب ان تكون مكتملة او شبة مكتملة مع اتاحة الفرصة لوضع اشياء اخرى و رؤية افكار الفريق و هذا العمل اساسي لبقاء وثيقة التصميم موضوعة على مسارها الرئيسي.

نموذج آخر باللغة الانجليزية:
https://docs.google.com/document/d/1axeeBWp683LPU8gCBQQqmquHMYHuG3uhNTN0LjSJBKk/edit

متطلبات الورشة:-

لبدأ العمل في هذه الورشة يجب ان تكون لديك خبرة في اليونتي ولو كانت بسيطة اي فهم لقوائم البرنامج. طريقة انشاء الملفات البرمجية، معلومات عن منظور الـ 2D, 3D في اليونتي، طريقة اضافة المكونات و خبرة بسيطة في الـ C# او JavaScript و لكن هذه الورشة ستناقش البرمجة بالغة الـ C# فقط فان واجهتك مشكلة في تطبيق الاوامر البرمجية و اردت نقلها الى الـ JavaScript فتستطيع نسخها بكل بساطة مع تغيير طرق الكتابة فقط، ان الاوامر البرمجية في الـ JavaScript لا تختلف كثيراً عن الموجودة في الـ C# فلهذا تستطيع اسقاطها دون مشاكل. في حالة لم تكن تملك خبرة في هذا المحرك فيفضل مراجعة دروس الموقع:
http://www.unity3d.com/learn

ايضاً يجب عليك الحصول على نسخة اليونتي 5.1.2f1 وما فوق لكي تطبق محتوى الورشة دون مشاكل، يمكنك تحميلها من هذا الرابط التالي:
http://unity3d.com/unity/download

ملفات الورشة:

بالنسبة للـ Sprits فأنا قمت بأخذها من موقع www.kenney.nl نظراً لضعف خبرتي في التصميم، ايضاً الاصوات قمت بأخذها من موقع https://freesound.org/ ، حقوق هذه الاصوات و الـ Sprits سنقوم بذكر المواقع التي اعتمدنا عليها في عمل هذه اللعبة، للملفات البرمجي فسأقوم بعملها اعتماداً على الخبرة التي توصلت لها في البرمجة على محرك Unity3D، بأذن الله سأقوم بتفصيل كل شيء في الكتاب و في حالة لم يكن هناك شيء واضح ارجوا اعلامي به عبر بريدي المذكور.

اضن انك قد تعرفت على نوع الصور المستخدمة في اللعبة فهي جميلة و تعتمد على خبرتك في جعلها تتحرك واستخدام الاشياء التي تتيحها لك هذه الرسومات بشكل يجعل اللعبة جميلة في حالة لم تعجبك الصور فهناك حلان اما ان ترسم او ان تشتري الرسومات وهناك حل آخر وهو التواصل مع خبراء في هذا المجال و تعطيهم افكار لكي تتحول الى الصور التي تريدها ففي النهاية يجب ذكر المصادر التي اخذت منها الصور او اي شيء آخر.

هنا تجد ملف المشروع بالاضافة الى اللعبة بعد بنائها:

- اللعبة بعد بنائها (نسخة الويندوز):
https://drive.google.com/file/d/0B-W_VzeA116VNVNWOVlOb0V6YjQ/view?usp=drive_web

- الـ Assets المستخدمة
https://drive.google.com/file/d/0B-W_VzeA116VRXNSajRhbnRuUHM/view

لماذا محرك Unity؟

بالحديث عن استخدامي لهذا المحرك فأنا بشكل صريح احب العمل على هذا المحرك فهو سهل التعامل و يوجد له العديد من الدروس بالإضافة الى ان اكثر محبي الالعاب يستخدمون هذا المحرك، Unity يوفر لغة برمجة C# و التي بنظري اراها مرتبة و تساعدك على بناء العابك بشكل جيد و منظم، ايضاً شركة يونتي توفر مرجع Scripting AIP الذي دائماً ساعدنا في مراجعة لغة البرمجة، فانا صراحة اجده مرجع قوي، في حالة اردت العمل على اي محرك آخر فلا بأس لكن تأكد من انه يشبه اليونتي في العمل او قم بأخذ فكرة عن ما تحتويه هذه الورشة و طبقة في المحرك الذي تريده.

المعمارية البرمجية (معمارية لعبتنا):-

قد تتساءل عن ماهية المعمارية البرمجية في الالعاب؟!

ان المعمارية البرمجية هو نظام برمجي يساعدك على التحكم باللعبة دون الحاجة للعبث بالكود البرمجي بشكل كامل او عند تغيير شيء تقوم بإعادة صيانة الكود و هذا مكلف جداً، لهذا يحب انشاء المعمارية البرمجية التي ستساعدنا على نقل اللعبة و اضافة مراحل و اعداء وبشكل عام التحكم بكل شيء في اللعبة، ان هذا سيتيح لك التعامل مع اللعبة بشكل اسرع، هذه المعمارية قد تستغرق منك وقت طويلاً لأنشائها، لابأس فكل مطور العاب يقوم بأنشاء معمارية خاصة به و ليس هناك قوانين معينة لهذه المعمارية بل القوانين وضعت لتكسر.

فكرة في ان لعبتنا ستستغرق وقت طويل لعملها ايضاً مدى تعقيدها، و هنا لن تكون اللعبة عبارة عن ملفات برمجية غير قادرة على تغيير نفسها بشكل سريع، لن ندعها عالقة هاكذا فأن لعبتنا سترتكز على معماريتها الخاصة و التي سوف نقوم بأنشائها، المعمارية البرمجية في النهاية هي الكود الذي يساعدك على التحكم بأي جزء في اللعبة دون اي تعب او الدخول و تغيير الكود كامل بحيث يظهر مشاكل و يخولك الى عمل عملية جراحية لهذا الملف بشكل كامل!.

بالنسبة لجودة هذه المعمارية فصراحة ليس هناك نسبة معينة نطرحها فأن تنظيم الكود و تسهيل عملية تغيير الاشياء هو بحد ذاته يعطيك فكرة عن مدى جودة هذه المعمارية.

فكر في لعبتنا في كيف سيتم ادارة اللعبة اعتماداً على الملفات البرمجية، ان في هذه الحالة سنضطر الى الاعتماد على قيم ثابتة تعيد نقاط تشغيل و اعادة اللعبة او بشكل اوضح ظهور قائمة الانتقال بين المراحل و التي سيتم تشغيلها اعتماداً على نقاطك او صحتك و هي ستعيد لنا عدد النجوم التي حصلت عليها نتيجة لعبك، نفس الشيء ينطبق الى اسلوب اللعب و ظهور الاعداء فهنا سنستخدم نظام الـ Object Pool و الذي سيساعدنا على تحديد على الاعداء الموجودين في المشهد، لاحظ ان هذا عامل اساسي فالوا تركنا الاعداء في الخروج لممتلئ المشهد بالأعداء و لن تكون هذه المشكلة الوحيدة فقط بل ايضاً هذا سيؤثر على ذاكرة تخزين الاعداء التي ستضل غير ثابتة في تخزينهم، لهذا نطام Object Pool سيساعدنا على وضع عدد معين في المشهد.

قد يتباذر في ذهنك سؤال ان اذا كان هناك طريقة معينة لإنشاء هذا النظام فأنا صراحة سأجيبك بانة ليس هناك نظام معين و انا دائماً استخدم طرقي، غالباً يتم الاعتماد على الخوارزميات لكن اجد ان هذه الطريقة معقدة قليلاً، لعبتنا ستعتمد على استخدام قيمة يتم من خلالها تحديد عدد الكائنات في المشهد و في حالة تجاوز الحد المطلوب سيتم ايقاف توليد الاعداء بهذه البساطة او ربما نستخدم دوال تعيد قيم true or false لإيقافها و اعادة تشغيلها و هذا الامر لن يكلفنا الكثير.

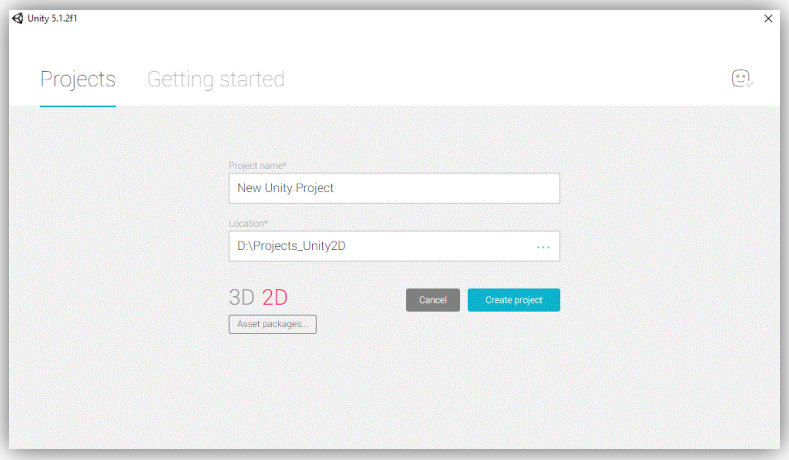
المعماريات التي سننشئها في اللعبة تعتمد على عدد الاشياء المهم في لعبتنا، لاحظ ان لذينا قائمة رئيسية واخرى مصغرة، ايضاً قائمة النجوم (الفوز و الخسارة)، الاصوات، توليد الاعداء و مدير اللعبة، كلها تحتاج الى معمارية تساعد على ربطها مع باقي الملفات البرمجية الضرورية لعملية الربط، لاحظ ان ملف مدير اللعبة Game Manager سيعتمد على Scores لتوقيف اللعبة وهو عامل مهم جداً فيها، ايضاً ملف Player Controller من خلاله سنحصل على قيمة الصحة الحالية للمدفع و التي بدورها هي عامل اساسي لظهور قائمة النجوم، القوائم ستعتمد جميعها على ملف واحد Menu Manager و العديد من الاشياء التي سيتوجب علينا عمل معمارية لها، ان تفصيل كل شيء هنا سيزيد ن تعقيد الامر و لندع الدروس تحدد مصيرنا.

بناء المشهد:-

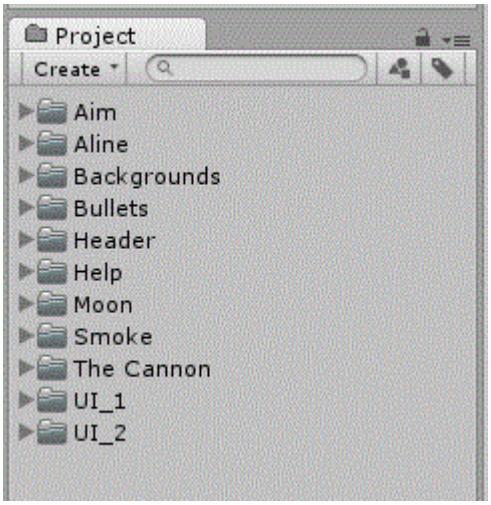
ان عملية بناء المشهد الحالي ستتم عبر ثلاث خطوات بحيث ان الاولى نقوم فيها بترتيب المشهد و الاعداء و الاعب ووضع مكوناتهم، الخطوة الثانية وضع الطبقات للمشهد اما الخطوة الثالثة وهي وضع الاضافات و طبقاتها، كمبدأ اولي ارى ان هذه العملية ضرورية حالياً لكي نرى شكل المشهد الاولي للعبة، بالنسبة للقوائم UI سنتطرق اليها لاحقاً، لان وضع كل شيء في هذه الفقرة قد يزيد من تشعب الامور، دعنا نبدأ بأول خطوة وهي ترتيب المشهد الاولي للعبة.

ترتيب المشهد، الاعداء و اللاعب:

بما اننا وصلنا الى المشهد فهنا نوضح الامر بشكل سريع اولاً قوم بتشغيل اليونتي و قم بأنشاء مشروع وتأكد من انه 2D كما هو موضح في الصورة، عند فتح المشروع تأكد من ان شاشة اللعبة هي (16:9) و ستراها مناسبة مع جميع الشاشات.



هنا ستظهر لك واجهه اليونتي المعتادة، اتركنا منها الان و لنتأكد من انك قمت بتحميل الملف المضغوط (مواد المشروع)، ستلاحظ وجود عدة ملفات و كل ملف خاص بشيء معين، كل ما عليك فعلة هو سحب جميع الملفات الى المشروع.



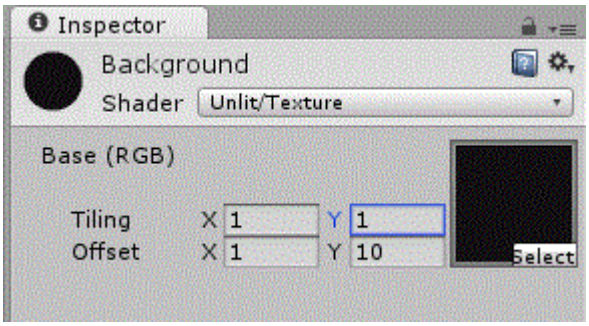
قم بوضع جميع الملفات في ملف واحد يمثلهم مثلاً Project Materials السبب لأنه يتوجب علينا انشاء ملف خاص بالأصوات و ملف آخر خاص بالاسكربتات و ملف اخير باسم Materials و الذي سنضع فيه الالوان و الصور، ملف الاصوات لن نحتاج له الان، اما بالنسبة

لملف الاسكربتات سنحتاج له دائماً، حسناً الى هنا نكون قد انهينا اول خطوة و هي تجهيز الملفات بشكل مرتب و سنتطرق اليها لاحقاً كلاً حسب دورة.

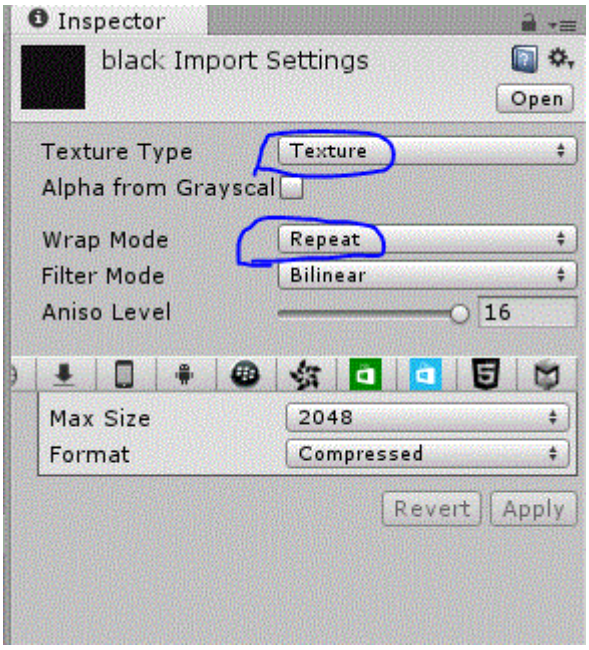
الان كل ما سيتوجب علينا فعلة هو تجهيز المشهد بشكل كامل كأول مرحلة نقوم بتطبيق كل شيء فيها.

١) قم بأنشاء Plane و تأكد من ان موقعة في rotation.x = -90 و هذا الـ Plane سيكون الخلفية.
٢) اختر صورة للخلفية من ملف Background.
٣) قم بأنشاء ماتريل في ملف Materials و قم بتسميته Background.

حسناً هنا سيتوجب علينا التعديل على الماتريل لكي نضيف صورة الخلفية للـ Plane، قم بالضغط على الماتريل و قم بجعل نوعه Unlit/Texture.



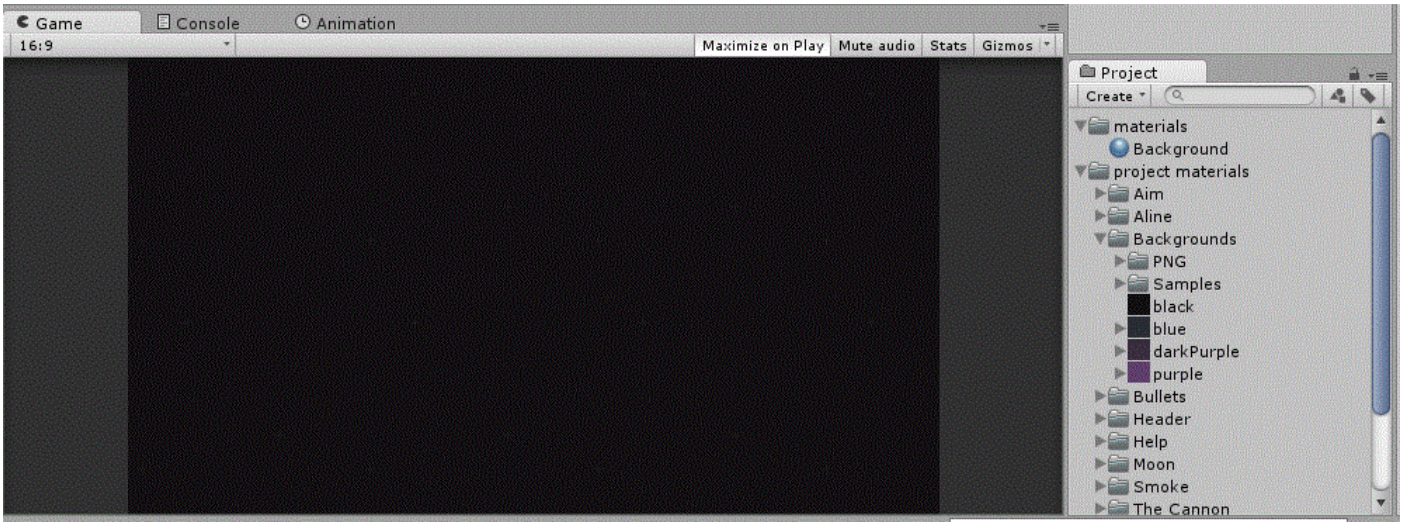
هنا سيطلب منا Texture، الان قم بالرجوع الى صورة الخلفية التي تريدها و قم بضبط عاداتها كما هو موضح في الاسفل.



لاحظ ان هنا حددنا نوع الـ Texture على اساس انه صورة من نوع Texture، الشيء الثاني اننا جعلنا نوع الالتفاف او Repeat اي قابل للتكرار، و هناك نوعان للالتفاف، الاول Clamp و الذي سيضع الصورة على طولها و عرضها و لن تكون قابلة للتكرار اما الثاني Repeat وهو الذي حددناه الان.

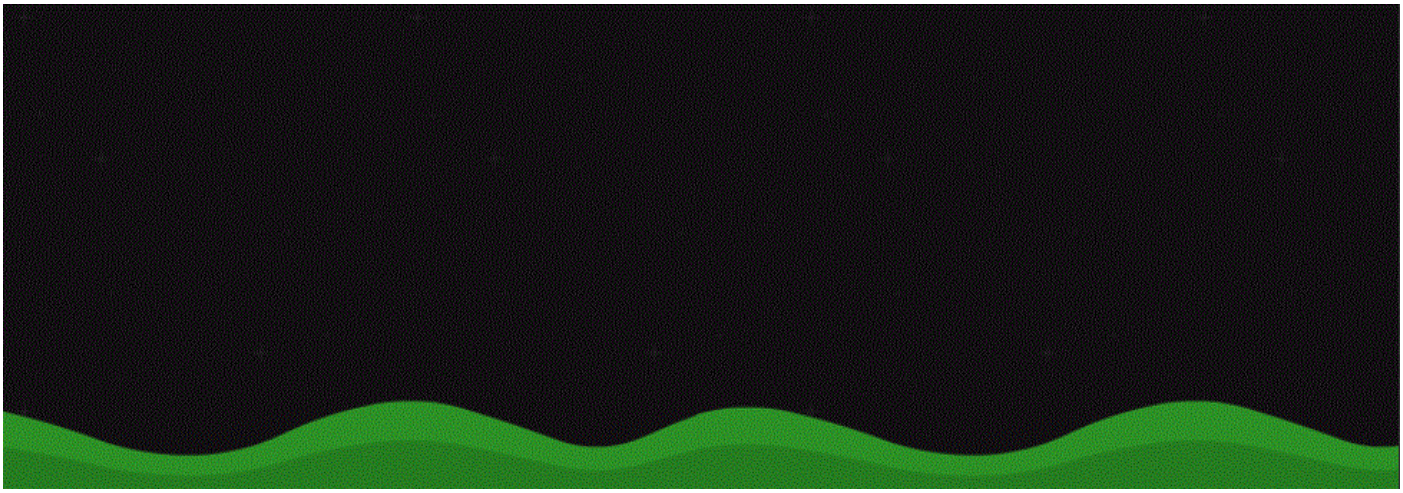
هنا كل ما عليك فعلة هو سحب الصورة الى الماتريل، بالنسبة لإعدادات التكرار ستلاحظ في صورة الماتريل السابقة انه يوجد خانه باسم Tiling X,Y و هذه الخنتان خاصتان بتكرار الصورة على المحورين.

حالياً اجد ان القيمة 2 على المحورين مناسبة مع الخلفية، بشكل سريع قم بجعل الخلفية تتناسب مع الشاشة اكان بنسخ الـ Plane او بتمديده.



هذا هو المشهد بشكلة الاولي، الان علينا وضع الارض و السحب و القمر و ضبط بعض الاعدادات الهامه.

الخطوة الاولى: بالنسبة للأرضية فستكون عبارة عن ارضية خضراء و منحنية بحيث ستطلب منك دفع المدفع بقوة الى الاعلى لكي يصعد و ربما هذا يسبب صعوبة بعض الشيء على اللاعب عند اللعب، لنعد الى ملف Background وستجد ان هناك ملف يتفرع منة باسم PNG قم بفتحة ستجد ملف آخر باسم Flat يحتوي على بعض الصور و جميعها على لون واحد، منة ابحث عن الصورة hills1 ستجد نسخة اخرى باسم hills2 لا حاجة لها حالياً، الان قم بسحب الصورة hills1 الى المشهد و تأكد ان وضعيتها في الوسط في الاسفل بحيث تغطي الجانب السفلي من المشهد لكي تكون لدينا ارض منحنية، للون اللارضية قم بجعل لونه 359B28FF هذا للون الاخضر للأرضية و اجده مناسب، قم بنسخ الارضية جانب الاخرى بحيث تصبح بهذا الشكل:



لاحظ البعد الجميل للأرضية و الذي يجعلها جميلة، كل ما عليك فعلة هو نسخ الارضية و جعل النسخة كائن ابن و قم بتغير تدرج اللون بحيث يتناسب مع صورة للون الأرض (الكائن الاب)، عملية بناء الارض اكتملت نسبياً فهذا شكل اول مرحلة بحيث يكون المدفع محاصر في هذا المشهد فقط، بالنسبة للكولايدر ستلاحظ ان الصور خالية من اي Collider2D فلهذا علينا وضع مكون Collider2D للأرض. بما ان لعبتنا 2D يجب علينا استخدام المكون Collider2D، بشكل سريع قم بإضافة مكون Polygon Collider2D في كل كائن اب و هو تلقائياً سيحدد على الارضية و يضع عليها الكولايدر بشكل مناسب.

الخطوة الثانية: هي اضافة القمر، بشكل سريع قم بالبحث عن الصورة باسم Moon وستجد العديد من الاقمار قم بأخذ ماتريد منها لكن تأكد من ان صورة القمر غير متناسبة في البعد مع الارض اي في حالة كان بعد الارض على محور Z = 0 فبعد القمر يجب ان يكون Z = 10 و نفس الشيء ينطبق على خلفية المشهد (Sky) يجب ان لا تتناسب مع بعد الارض في محور Z.

الخطوة الثالثة: و هي اضافة المباني، الان بالعودة الى ملف Background ستجد ان هناك صورتين للمباني من نوع 2D الاول باسم house_beige_front و الثاني باسم house_grey_front قم بالبحث عنهم و اسحبهم الى المشهد، بالنسبة للحجم ستلاحظ انه صغير جداً لهذا قم بزيادة حجمهم، في حالة لم تعجبك الصور فبكل بساطة قم بتحميل صور اخرى، في حالة اردت تغطية الفراغات ستلاحظ في ملف PNG ان هناك العديد من الصور للأشجار و السحب و اشياء اخرى يمكن الاستعانة بها لكن تأكد من ان جميع الصور في الخلفية لا تحتوي على مكون Collider لأننا فقط نريد من المدفع ان يصطدم بالأرض.

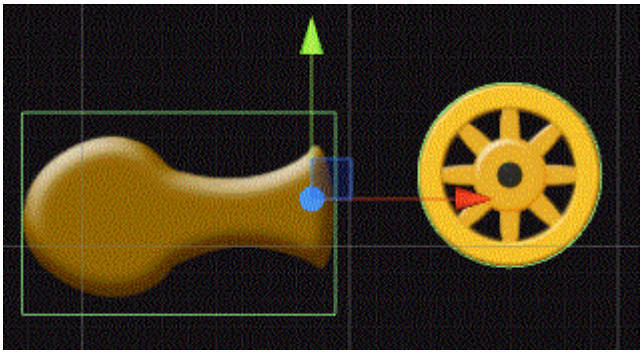
الى هنا اجد ان المشهد قد اكتمل رسمة.

بما ان المشهد قد اكتمل نسبياً علينا اضافة المدفع و هو عبارة عن قطعتين (المدفع و العجلة) تستطيع ان تجدهم في الملف The Cannon ستلاحظ ان هناك 4 صور، قم بسحب الصورتان الملونتان فقط الى المشهد، بالنسبة لربط المدفع بالعجلة و العكس فهذا الشيء سنعتمد علية برمجياً.

هنا علينا وضع الكولايدر في العجلة و المدفع، لنحدد على المدفع و لنضف اليه مكون Box Bollider2D او استخدم Polygon Collider2D لأنه في كلا الحالتين الطلقات لن تهتم لشكل الكولايدر، ايضاً قم بإضافة المكون Rigidbody2D و قم بجعلة Is Kinematic، هذه الوضعية ستجعل المدفع غير خاضع للفيزياء لكن سيكون فيزيائي في بعض الاشياء مثل التصادمات و الاحتكاكات.

لنعد الى العجلة و التي لم نضف اليها اي شيء حالياً، حسناً بما انها عجلة يجب ان تكون خاضعة للفيزياء ككل، قم بإضافة مكون Circle Collider2D وايضاً Rigidbody2D اما للاحتكاك Drag سنضعه على القيم الافتراضية له لأني اجدها مناسبة للعجلة.

الان قم بتجربة المشهد وستلاحظ ان المدفع غير خاضع للجاذبية لكن في حالة الاصطدام فأن العجلة تتأثر به عند الاصطدام .

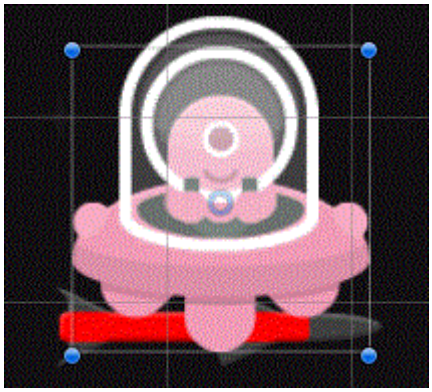


علينا وضع الاعداء في المشهد، بالنسبة للأعداء فأننا كبداية يجب علينا تحديد شكل العدو الاول، شخصياً اجد ان الصورة shipYellow_manned هي مناسبة لتكون العدو الاول قمت بالبحث عنها و اسحبها الى المشهد و اجعل اسمه EnemyOne، اما السلاح فليس هناك شكل محدد له قد تستطيع استخدام الصورة laserGreen1 كما عملت انا بحيث قمت بضبط موقعها و حجمها وجعلتها كائن ابن للعدو الاول:

قم بإضافة المكون Circle Collider2D و المكون Rigidbody2D، الان قم بأنشاء ملف باسم Prefabs بحيث ان هذا الملف سيحتوي على جميع الكائنات من نوع prefab فيه، هذا اول كائن نضعه على شكل prefab وهو العدو الاول EnemyOne.

العدو الثاني EnemyTwo سيكون عبارة عن عدو يطلق القنابل و سيحتوي على درع عادي يحميه من الهجمات لكن هذا ماسيبطئ سرعته في الهواء نتيجة لحجمة الكبير، اذن قم بالبحث عن الصورة shipBeige_manned ستلاحظ شكل العدو قم بسحبه الى المشهد و اعطية اسم EnemyTwo حسناً شيء ثاني و هو الخوذة او الدرع الذي سيغطي الرأس، ابحث عن الصورة dome و اجعلها ابن للعدو الثاني، حسناً قم بإضافة المكون Rigidbody2D الى الكائن الاب ايضاً قم بإضافة Circle Collider2D، الخوذة dome اضف اليها المكون Box Collider2D فقط، بالنسبة للقنابل فهي ستخرج من الاسفل بشكل مباشر و ستنطلق الى المدفع تلقائياً، بما ان هذا العدو يجب ان يكون ضخم قم بجعل قيمته غلى المحوري X,Y في Scale = 1.5 اكبر من العدو الاول بنصف حجمة شيء اخير قم بجعلة prefab، اذن هنا يتكون لذينا الشكل التالي للعدو الثاني:

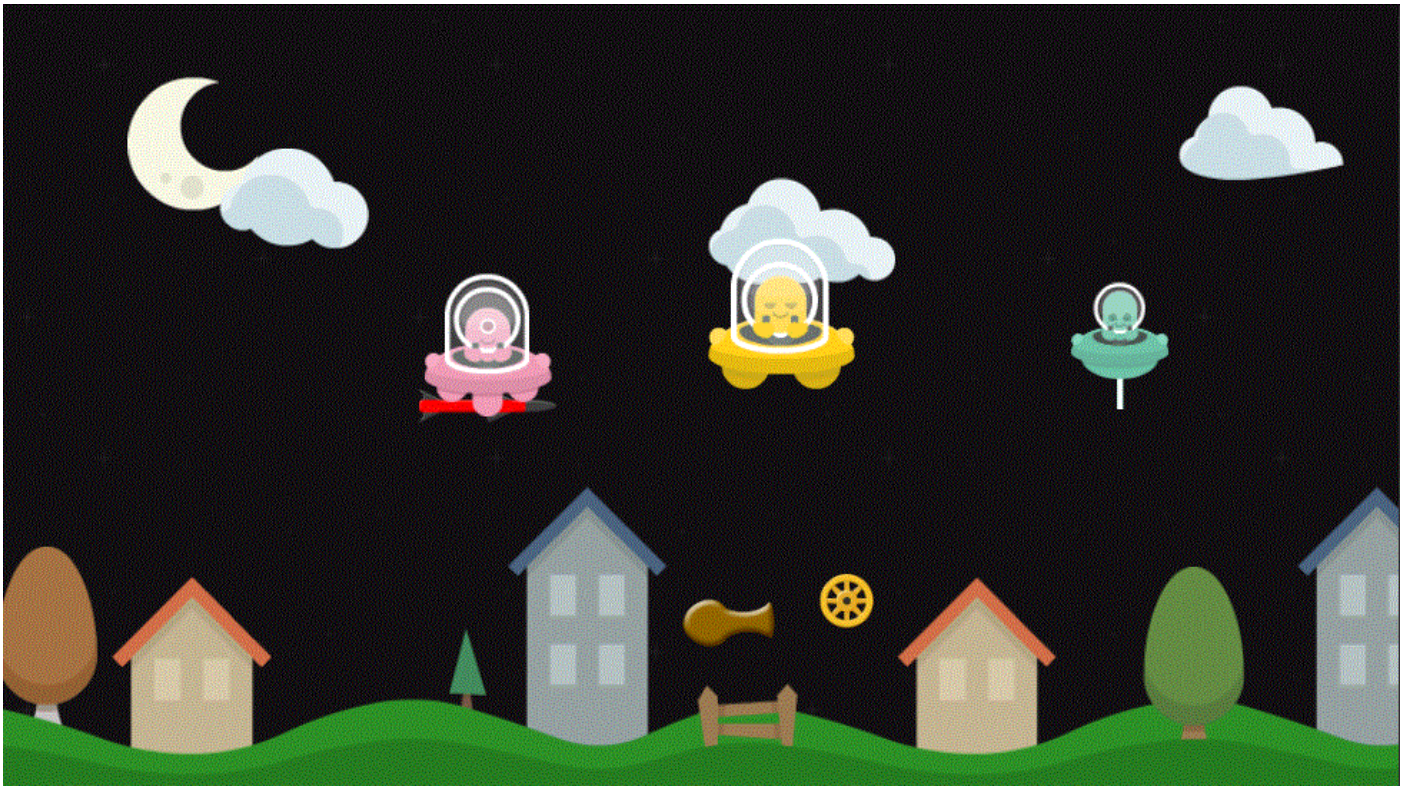
العدو الثالث و سيكون قادر على اطلاق الصواريخ و سيكون سريع جداً و بحماية اقوى، حسناً قم بالبحث عن هذه الصورة وهي تمثل العدو الثالث shipPink_manned قم بسحبة الى المشهد و اجعل حجمة 1.3 في Scale ايضاً قم بإضافة الخوذة dome و اضبط موضعها في العدو، هذا العدو سيكون حامل للصواريخ لهذا علينا اضافة صاروخ في اسفله و لكي تجد الصاروخ قم بالبحث عن الصورة Bullet Enemy2 لا تهتم للاسم فقط قم بسحبها الى العدو بشكل مباشر بحيث يتكون لك هذا الشكل الكامل للعدو الثالث:



الان يجب وضع المكونات لهذا العدو، بالنسبة للخوذة بشكل مباشرة قم بوضع Box Collider2D، الجسم Circle Collider2D و مكون Rigidbody2D فقط، بالنسبة للصاروخ فهو شيء عادي و لن يؤثر في اللعبة اي فقط هو عبارة عن رمز اي ان هذا العدو يطلق الصواريخ.

تبقى لنا عمل زعيم الاعداء، في حالة لم تجد اي مركبة تمثل زعيم الاعداء فبإمكانك تنزيل صورة للزعيم و هذا موقع مساعد (http://opengameart.org/content/spaceship-2d) في الرابط ستجد صورة للزعيم يمكنك تحميلها، حسنا الان قم بسحب الزعيم على المشهد لكي نقوم بتجهيزه، ان مكونات الزعيم لن تختلف ابداً عن مكونات باقي الاعداء، قم بوضع كولايدر يغطي هذا الزعيم، ايضاً Rigidbody2D و قيمة ستكون نفس قيم باقي الاعداء لكن سيتقى لك امر تحديد الاحتكاك، شخصياً ارى ان القيمة ١٠ كافية لتحريك الزعيم بشكل جميل، حسناً، بالنسبة للـ Tag قم بوضع تاج جديد يمثل الزعيم Boss ايضاً الـ Layer سيتكون نفس باقي الاعداء Enemies، حسناً هنا نكون قد انهينا العمل على جميع الاعداء.

الى هنا نكون قد انتهينا من اضافة الاعداء الان بقي لذينا التعديل على المكون Rigidbody2D في جميع الاعداء، بشكل مباشر حدد على الكائنات من الـprefab الخاص بكل واحد و قم بإيقاف الـ Gravity Scale للجميع اي قم بجعل قيمته تساوي الصفر، بالنسبة للاحتكاك Linear Drag فالعدو الاول ستكون قيمته ٤٠ هذا لان حركته ستضل عادية، اما العدو الثاني ٥٠ لان حركته ستكون بطيئة، العدو الثالث ٣٠ لان سيتحمل الصواريخ و يتحرك بسرعة كبيرة، بالنسبة للـ Constraints قم بتوسيعه، ستلاحظ وجود الخانتين Freeze Position و Freeze Rotation ، اما Freeze Rotation ستلاحظ وجود محور Z فقط وهو الذي يتكفل بتدوير الكائنات الفيزيائية، لهذا قم بإيقافه في جميع الاعداء لكي لا يدور اي عدو حول نفسة او في حالة تصادم مع باقي الاعداء او شيء من هذا القبيل، الان هذا هو الشكل النهائي للمشهد

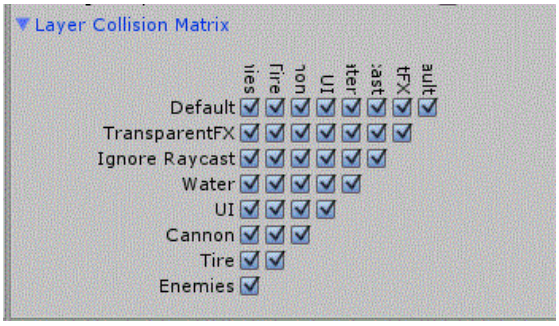


وضع الطبقات للمشهد:

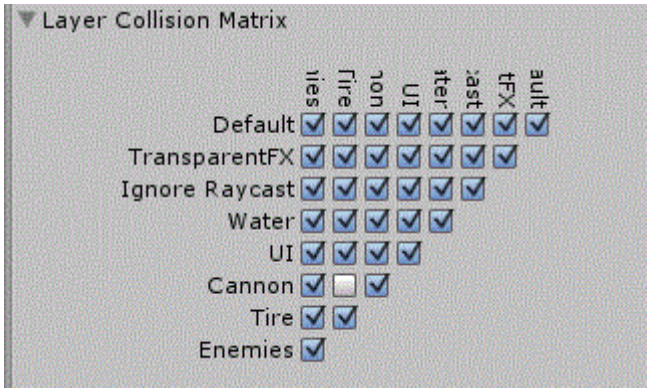
بما اننا وصلنا الى هذه النقطة الجميلة يجب علينا وضع تاج (Tag) يمثل كل كائن، المدفع و العجلة بتاج يمثلهم Player و الاعداء كل تاج بأسمة.

سيتبقى شيء آخر وهو ان علينا وضع Layer بين لكل للأعداء و اللاعب (Layer للعجلة و آخر للمدفع)، اذن للنشء Layer للعجلة باسم Tire و Layer للمدفع باسم Cannon و اما بالنسبة للأعداء فلنجعل Layer واحد يمثلهم جميعاً باسم Enemies.

حسناً دعنا نفكر في ماذا لو تحركت العجلة و ضل المدفع مرتبط فيها؟! ان المدفع كائن يتأثر بالخواص الفيزيائية الى انه ثابت يا في حالة التصادم سيتأثر بالعجلة و ستحصل مشكلة لهذا علينا فصل العجلة عن المدفع او جعلهم لا يتصادمان مع بعضهما البعض لكن كيف نفعل هذا؟ ان اليونتي يوفر قائمة Layer Collision Matrix يساعدك على تحديد الاجسام القابلة للتصادم مع بعضها، تستطيع ان تجده من Edit> Project Setting> Physics2D و حالياً هذه جميع الـ Layers الذي عملناهم :

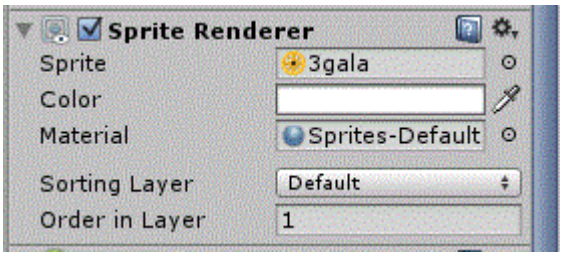


ان هذه هي الحالة الطبيعي للتصادم فكل جسم سيتصادم مع الاخر اي جميع الاجسام قابلة للتصادم مع بعضها، لهذا سنستثني العجلة من الـ Tire لاحظ الصورة التالية:



في هذه العملية استثنينا العجلة من المدفع اي ان المدفع لن يتصادم مع العجلة، و يمكنك ان تجد العديد من الاشياء التي تحتوي على Layer خاص فيها.
ان وضيفة الـ Layer ليس تحديد تصادمات الاجسام فقط، بل ايضاً يساعدك على تقليل نسبة التصادم في الصفحة الواحدة او Layer واحد و بدل ان تكون جميع الكائنات في طبقة واحدة، بهذا الشكل تقوم بفصلها و وضع كل طبقة ضرورية مع التي تريد التصادم بها و هذا يقلل من نسبة التصادم و الضغط في الطبقات.

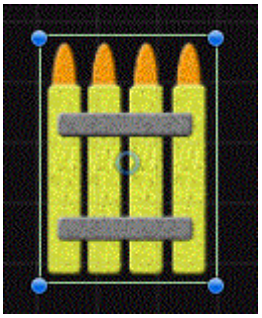
بالعودة الى المشروع قد تلاحظ ان المدفع متقدم على العجلة او العكس اي ان العجلة خلف المدفع و يجب ان نضع المدفع خلف العجلة لكي تظهر العجلة اولاً، بالنسبة لهذه المشكلة فحلها بسيط جداً، قم بالضغط على العجلة، ستلاحظ وجود المكون Sprite Renderer وهو المكون الذي يحتوي على Sprite العجلة ايضاً العديد من الخصائص، في الاخير ستجد الخاصية Sorting Layer و التي تخولك الى انشاء طبقة جديدة تقوم بتحديد مستواها، او Order in Layer (بعد الطبقة) و الذي كلما زادت قيمته زاد بروزه او تقدمة الى الامام، حسناً الان قم بتحديد بعد طبقة العجلة = 1 ستجد ان العجلة باتت اقرب الى المشهد، في حالة حصلت مشكلة ووجدت ان العجلة تضل ظاهره في اول المشهد فبكل بساطة قم بتحديد بعد طبقة لكل شيء اولي في اللعبة.



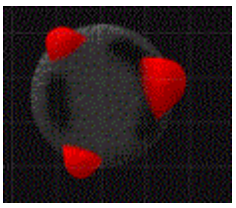
وضع الاضافات و الطبقات:

بالحديث عن الاضافات في اللعبة فالكل يعرف انه ليس هناك لعبة لا تخلوا من الإضافات التي تساعدك على الاستمرار في اللعب، ان لكل لعبة اضافات خاصة و كذلك الامر في لعبتنا هذه، تكلمنا سابقاً عن الاضافات و هي 3 انواع (خاصة بالطلقات، الصحة و الذرع) بالنسبة للطلقات تكلمنا ان لذينا ٣ انواع من الطلقات و جميعها تحتاج اضافة كي تساعدك على الاستمرار في استخدام هذه الطلقات، ايضاً الصحة و التي ستساعدك على البقاء فترة اطول في المشهد، و الذرع الذي سيحميك من طلقات العدو القاتلة، حسناً بما ان لذينا هذه الفكرة عن الاضافات في اللعبة هنا سننتقل الى تعديلها و اضافتها في المشهد.

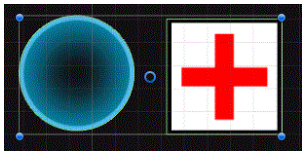
طلقات المدفع ستكون عبارة عن 3 طلقات، الاولى ستكون مخصصة للطلقة رقم 1 و التي هي الطلقة الطبيعية للمدفع، قم بالبحث عن الصورة Bullets ستلاحظ وجود صورة فيها بضع طلقات على شكل حزمة، هنا علينا اعتماد هذه الصورة للحصول على ذخيرة للطلقة رقم 1 و 2، اما بالنسبة للطلقة رقم 3 فسنستخدم نفس الصورة المستخدمة في الطلقة رقم 3 Boom و هي القنبلة، حسناً بالتعديل على الصورة الاولى قم بسحبها الى المشهد، ستلاحظ حجمها الكبير قم بتغييرة الى 0.1f و ستجدها مناسبة مع المشهد، حسناً علينا اضافة المكونات لهذه الذخيرة، بدايةً قم بإضافة المكون Box Collider2D ثم المكون Rigidbody2D منة قم بجعل الـ Gravity = 0 ايضاً خانتا الـ Drag اخيراً قم بإلغاء تفعيل المحور Freeze Rotation = z، شيء ثاني علينا وضع Layer يضم جميع الاضافات ايضاً Tag لكل اضافة تمثلها، قم بإضافة Layer باسم Help لهذا و Tag باسم Ammo_1 و الذي سيحدد ان هذه الطلقة هي رقم واحد، الى هنا سيتبقى لنا سحب الطلقات، اسحب الطلقة الى المجلد Prefabs و قم بتسميتها Ammo_1 و قم بسحبها مجدداً و سمها Ammo_2 لكن هذه ستتطلب Tag باسم Ammo_2



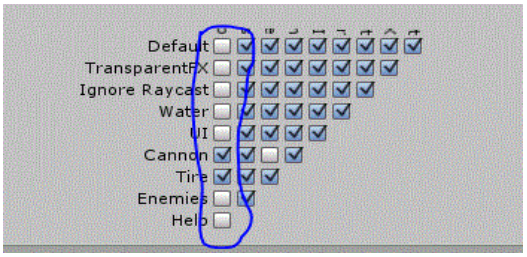
الى هنا نكون قد اكملنا الذخيرة للطلقة الاولى و الثانية، بالنسبة للطلقة الثالثة بشكل سريع قم بالبحث عن الصورة Boom و اسحبها الى المشهد ايضاً ستلاحظ ان حجمها كبير جداً لهذا قم بوضع حجمها على 0.1f نفس الطلقة الاولى، بالنسبة للمكونات لن اعيد كلامي اكثر لهذا قم بإضافة نفس المكونات في الطلقة الاولى ايضاً الـ Layer و استبدل Box Collider بـ Circle Collider2D و تأكد من ان قطرة متساوي مع الصورة ، لكن بالنسبة للـ Tag فهنا يجب وضع تاج باسم Ammo_3 بشكل بسيط و بعيد عن التعقيدات.



تبقى لذينا الصحة و الذرع، بالنسبة للصور قم بالبحث عن الصورتان (Health, Force) هنا سيتوجب عليك اضافة نفس الاشياء لكن تأكد من ان الـ Tag تبغ الصحة هو باسم Health و التاج تبع الـ الدرع هو Force اما باقي الاشياء فهي ستبقى في مكانها، عملية ربطها ستكون برمجياً لهذا سنحتاج الى اسم التاج تبعها و الذي سيساعدنا على تحديد نوع الكائن الذي اصطدم بالمدفع.



آخر شيء تبقى لنا وهو تحديد التصادمات بين هذه الاضافات بشكل عام و بين المدفع و باقي الكائنات، اي ان هذه الاضافات لن تتصادم مع بعضها و ايضاً لن تتصادم مع اي شيء آخر عدا المدفع، حسناً بالرجوع على القائمة Edit> Project Setting> Physics2D و التي سنحدد منها التصادمات، من الامتداد ستجد التصادمات قم بوضع الاعدادات كما هي موضحة في الصورة. من الصورة تلاحظ ان الاضافات لن تتصادم مع نفسها او اي Layer آخر ماعدا الكائن Cannon, Tire ،بهذا الشكل تكون عملية التصادم خاصة بالمدفع فقط اي ان الاضافات ستتصادم بالمدفع اما باقي الاشياء ستتجاهلها و تستمر بالحركة امامها دون اي مشكلة تذكر.



برمجة اللاعب (Player):-

ان مصطلح اللاعب في اي مشروع لا يعني ان اللاعب عبارة عن شخصية (انسان) بل مصطلح اللاعب يطلق على الشيء الذي ستلعب به ففي حالتنا هذه اللاعب عبارة عن مدفع و عجلة و كلمة برمجة اللاعب يعني اننا سنبرمج المدفع و العجلة، اي ان اللاعب في هذه اللعبة عبارة عن قطعتين (مدفع و عجلة) و سنبدأ ببرمجتهما تدريجيا، في البداية قم بحفظ المشهد في مجلد باسم Scenes بحيث ان هذا المجلد سيحتوي على جميع مستويات اللعبة، حسناً دعنا نبدأ بالتدريج بحيث اننا سوف نقسم هذه الفقرة الى عدة نقاط و في كل نقطة نقوم ببرمجة شيء معين في اللاعب، ايضاً هذه الطريقة سنعتمدها في برمجة باقي الكائنات في اللعبة، اذن دعنا نبدأ بأول فقرة.

موقع المدفع من العجلة:

حالياً تعرف ان المدفع جزء منفصل من العجلة و العكس اي ان كلاً منهما جزء منفصل و لهذا يجب علينا ربطهما معاً، اذن بنقم بربطهم معاً، تأمل هذه الصورة:

في البداية قم بأنشاء ملف باسم Player داخل ملف Script، بحيث ان هذا الملف سوف يمثل جميع الملفات البرمجية للاعب بشكل عام، منة قم بأنشاء ملف برمجي C# باسم PlayerController ان هذا الملف سوف يمثل جميع خصائص المدفع ماعدا اطلاق النار، حسناً قم بفتح الملف البرمجي و لاحظ هذا الكود:

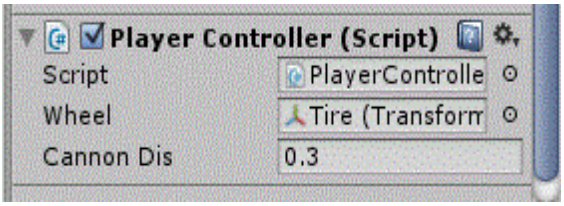
```
1.  public Transform Wheel;
2.  public float CannonDis;
3.
4.  void Update(){
5.
6.          CannonPosition ();
7.  }
8.
9.  void CannonPosition ()
10. {
11.         transform.position = Wheel.position;
12.         float posY= transform.position.y - Wheel.position.y;
13.         float deltaPositionY = transform.position.y + (Mathf.Sin(posY- 90)  * -CannonDis);
```

```
14.
15.         transform.position = new Vector2(transform.position.x, deltaPositionY);
16. }
```

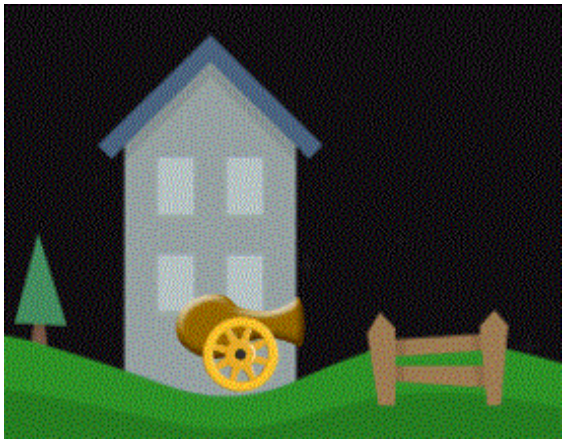
حسناً دعني اوضح لك هذا الكود، بشكل عام هذا الكود يقوم بحساب فارق المسافة بين موقع العجلة و موقع المدفع بينهم فقط، نحتاج الى ان نقوم بانقاص موقع المدفع من موقع العجلة على محور y فقط بينما محور x سيضل على حالة الطبيعي لا نه يجب تحريك المدفع مع العجلة على محور x و لا حاجه لنا لوضع مسافة بينهم، اذن يجب استخدام المحور y و التعامل مع قيمة لهذا سنستخدم الدالة Sin و التي تقوم بأعاده القيمة الموجودة على المحور y بين الكائنين و هذه الدالة تعيد لنا قيمة الناتج بالراديان لهذا الفئة Mathf توفر لنا متغير Rad2Deg و الذي يتكفل بتحويل هذه القيمة من الراديان الى الدرجات و العكس في حالة استخدمت المتغير Deg2Rad اي من الدرجات الى الراديان.

لاحظ في السطر 1 قم بتعريف متغير من نوع Transform باسم Wheel و هذا المتغير يمثل كائن العجلة الذي سنقوم بتحديد موقع المدفع من موقعة، بالنسبة لاستخدامي للـ Transform و لم استخدم GameObject السبب هو ان الاخير سيتيح لي التعامل مع جميع المكونات في الكائن و انا لا احتاج الى المكون Transform لأننا نريد تحديد موقع فقط، في السطر 2 عرفت متغير من نوع float هذا المتغير سيساعدنا على تحديد مسافة المدفع من موقع العجلة على المحور y، في السطر 4 كتبت الدالة Update لان الكود الذي نتعامل معه غير فيزيائي لهذا تشغيله في Update لن يضر ابداً، في السطر 6 هنا قمت تعريف دالة تمثل موقع المدفع Cannon، داخل الدالة لاحظ انني ابتدأت بتحديد موقع المدفع من موقع العجلة هذه القيمة الاولية و التي ستساعدنا على التعامل مع فوارق المسافات، تحته مباشرة في السطر 12 عرفت متغير من نوع float يخزن موقع المدفع من موقع العجلة على محور y لكي نعيد استخدامه و انقاص المسافة منة، اسفله عرفت متغير آخر من نوع float باسم deltaPositionY و الذي يخزن فارق المسافة بين المدفع و العجلة، بشكل منطقي نريد التعامل مع قيم المحور y بين المدفع و العجلة لهذا سنستخدم قيم الـ y بالنسبة لموقع المدفع و نقوم بإضافة فارق المسافة بينهم لكي نقوم بانقاص المسافة بين الكائنين على محاورهم لا على محاور البرنامج لهذا نستخدم الدالة Sin من الفئة الرياضية Mathf و تلاحظ انني اعدت استخدام المتغير posY وقمت بطرح 90 درجة منها لا حاجه لنا بها، ان هذه العملية ستعطينا موقع المدفع من العجلة بشكل مباشر، حسناً يجب استخدام المتغير CannonDis لكي نقوم بتحديد الفارق بينهم و تلاحظ انني استخدم المتغير او بالأصح قمت بضرب القيمة الحالية في المتغير بالسالب لان القيمة ستعود بالسالب فكلما زادت قيمة المتغير CannonDis ستزيد المسافة بين الكائنين بالموجب، الان يجب اعادة استخدام المتغير deltaPositionY و الذي يخزن فارق المسافة بين الكائنين و يجب وضع المتغير في Vector2 بالتحديد على محور y لكي يتسنى لنا التلاعب بقيمة المحور y و هذا بالضبط ماعملته في السطر 12 استخدمت Vector2 لان اللعبة 2D بكل بساطة.

صراحة اجد ان الكود بسيط جداً و لا توجد به تشعبات كثيرة، ان مراجعته اكثر ستزيد من فهمك له، عموماً يجب علينا العودة الى الشروع و نرى القيم التي ظهرت لكن اولاً تأكد من سحب الملف الى المدفع لكي يتسنى لك رؤية المتغيرات، لاحظ هذه الصورة:



لاحظ المتغيرات التي ظهرت، Wheel و الذي يمثل العجلة لهذا قم بسحب العجلة الى هذه الخانة، تحته المتغير CannonDis و الذي أعطيته القيمة 0.3 و التي اراها مناسبة.
اذن لنرى المشروع بشكل كامل، لاحظ موقع المدفع من العجلة مع تأثر العجلة بالجاذبية :



تأمل الصورة، ان موقع المدفع يعمل مسافة قدرها 0.3 راديان من موقع العجلة او منتصف العجلة و عند تحرك العجلة ستلاحظ ان موقع المدفع لن يتغير و سيضل ثابت في مكانة، السبب اننا انقصنا المسافة بين الكائنين بالتحديد بين مراكزهم فقط.

هذا كان بالنسبة لمدفع المدفع و هذا خاص بملفة البرمجي، هو ليس الزاماً كتابة الكود في ملف المدفع بل ايضاً يمكنك كتابته في ملف العجلة ان اردت لكن من باب الترتيب تعمدت فعل هذا.

دوران المدفع (Cannon):

بالحديث عن دوران المدفع في حالتنا هذه على المدفع النظر الى موقع الماوس في فضاء العالم، لهذا يجب استخدام دوال الفئة Camera و هي توفر دوال لوضع موقع معين في فضاء المشهد او العالم و ايضاً الاشعة و غيرها، و في حالتنا هذه يجب تحديد موقع مؤشر الماوس في فضاء العالم في الكاميرا، يوجد 3 طرق فعالة و لكن الطريقة الاخيرة خاصة بالمشاريع الـ 3D، دعنا نناقش الطريقتان و شرح سريع للأخيرة.

الطريقة الاولى (النسب المثلثية):

عند دوران المدفع تلاحظ ان للمدفع محوران x,y في حالة اردنا تدوير المدفع على المحورين فيجب علينا اعادة ناتج قسمة المحورين في المحور z لأنه لو رجعت للبرنامج و قمت بتدوير المدفع ستلاحظ ان المدفع يدور على المحور z فقط و هذه ربما قد تكون مشكلة، حسناً ليس بالفعل لكن غالباً فكر في كيفية تدوير هذا المدفع، بالنسبة لطريقة الاشعة فمن وجهه نظري ارى انها غير مفيدة حالياً و لكن تنفع في الالعاب من منظور 3D لكن لعبتنا من منظور 2D لهذا سيتوجب علينا التعامل مع قيم المحورين لا عادة الناتج في محور الـ z، دعني اقم بتوضيح الامر لك اكثر، ان عملية قسم المحورين x,y يعيد لنا ظل الزاوية الذي تعود قيمته بالراديان بالنسبة لتحويل الوحدة فهذا ذكرناه سابقاً، اذن عملية قسمة المحورين تكون بشكل sin/cos اي جيب الزاوية(جاهـ) على جيب تمام الزاوية(جتاهـ) و تعود القيمة بالراديان، يمكنك مراجعة الامر من هنا، يونتي يوفر لنا دوال مثل MoveRotation و MovePosition و جميعها دوال الفئة Rigidbody بشكل عام و هذا يشمل الـ 2D، بالحديث عن الدالة MoveRotation للفئة Rigidbody2D فأن هذه الدالة تطلب متغير من نوع float يخزن قيمة قسمة الوتر y الى x (ليس فعلينا لكن في حالتنا هذه) و القيمة تعود للمحور z لأنه وبكل بساطة المكون Rigidbody2D يقوم بتدوير الكائن على المحور z فقط، اذن لندخل الى الكود البرمجي في الملف PlayerController و تأمل هذا الكود:

```
1.  private Rigidbody2D rb2D;
2.
3.  void Start()
4.  {
5.          rb2D = GetComponent<Rigidbody2D>();
6.  }
7.
8.  void Update(){
9.
10.         CannonPosition();//دالة موقع المدفع السابقة
11. }
12.
13. void FixedUpdate(){
14.
15.         CannonRotation();
16. }
17.
18. void CannonRotation(){
19.
20.         Vector2 mousePos = Input.mousePosition;
21.         mousePos = Camera.main.ScreenToWorldPoint(mousePos);
22.
23.         float deltaX = transform.position.x - mousePos.x;
24.         float deltaY = transform.position.y - mousePos.y;
25.         float angle = Mathf.Atan2(deltaX,-deltaY) * Mathf.Rad2Deg;
26.
27.         rb2D.MoveRotation(angle + 90);
28. }
```

قد ترى ان هذا الكود معقد و كبير لكن دعني اطمئنك ان هذا الكود بسيط جداً و مفهومة سهل، لاحظ في البداية قمت بتعريف متغير rb2D من نوع Rigidbody2D و هذا المتغير خاص وهو يمثل المكون Rigidbody2D في المدفع، في Start قمت بجعل هذا المتغير يحصل على المكون Rigidbody2D من الكائن لكي يتسنى له الوصول الى الدوال و المتغيرات الخاصة فيه، في السطر 13 كتبت الدالة FixedUpdate لان الكود الذي سنكتبه في الدالة CannonRotation هو كود فيزيائي و عدد اطارات الدالة FixedUpdate ثابت بعكس Update، في السطر 18 قمت بأنشاء الدالة CannonRotation و التي تمثل دوران المدفع بالنظر الى موقع الماوس.، يجب علينا تخزين موقع الماوس في متغير من نوع Vector2 لكي نستطيع ان نحسب المسافة بينة و بين موقع المدفع لهذا في السطر 20 قمت بتخزين موقع الماوس في متغير Vector2، في هذه الحالة من الضروري تحديد موقع الماوس بالنسبة لفضاء العالم على الكاميرا لا عطاء البعد الحقيقي لموقع الماوس، من الفئة Camera اولاً استخدمت المتغير main وهو يعني الكاميرا الرئيسية لتصوير المشهد و هذا المتغير يحتوي على العديد من الدوال التي تساعدك على تحديد بعد موقع كائن من نوع vector3 و داخلياً يتم تحويل كل Vector3 الى Vector2، ستجد العديد من الدوال لتحديد المواقع، انت بشكل مباشر قم باستخدام الدالة ScreenToWorldPoint وهذه الدالة تطلب متغير من نوع Vector3 Position اي متغير يخزن موقع كائن على المحاور الثلاثة فهنا بشكل مباشر معنا المتغير mousePos و الذي يخزن موقع الماوس فيه.

هنا يجب علينا انقاص موقع المدفع من موقع الماوس على محور x و نفس الشيء على محور y و تلاحظ في السطرين 24 ,23 ان هذا ماقمت بعملة بالضبط، يجب علينا قسمة المحورين لكي تعيد لنا قيمة ظل الزاوية لكن بالشكل العادي سيعقد الامر كثيراً لهذا الفئة Mathf توفر لنا دالة Atan2 و هي تطلب متغيرين من نوع float كل منهما يخزن قيمة محور معين و هي تتكفل بتخزين قيمة ظل الزاوية في المتغير angle و هذا ما ستجدة في السطر 25 و لاحظ ايضاً انني قمت بضرب القيمة في المتغير Rad2Deg ليعيد لنا القيمة بالدرجات، اخيراً استخدمت الدالة MoveRotation و كما قلت هي تطلب متغير من نوع float و حالياً المتغير angle يخزن ظل الزاوية المطلوب لتدوير المدفع، بشكل مباشر قم بكتابة المتغير في الدالة لكن عند تشغيل اللعبة قد ترى ان المدفع لا يدور بالشكل الجيد، لهذا تأكد من اشارات القيمة السالبة و الموجبة ايضاً قم بزيادة 90 الى المتغير angle كما هو موضح في السطر الاخير وهي الـ 90 الضائعة من الظل.

الان قم بمناداة الدالة في FixedUpdate و قم بتشغيل اللعبة، ستلاحظ ان المدفع ينظر الى مؤشر الماوس بشكل جيد و خالي من المشاكل.

الطريقة الثانية (الانسيابية):

بالنسبة للطريقة الثانية و هي عمل حركة ناعمة اثناء الدوران اي انتقال من موقعة الحالي الى موقع الماوس بشكل ناعم و قد تستفيد من هذه الطريقة كثيراً لكن لن اعتمدها في المدفع، بالنسبة للحركة الناعمة فيها ليست فيزيائية و تستطيع تشغيلها في Update و لن تلاحظ اي مشاكل، اولاً دعني اعطيك نظرة اقوى الى هذه الطريقة، في هذه العملية لن تضطر لتعريف متغير Rigidbody2D لان جميع القيم سوف تتحول الى الفئة Quaternion وهي التي سنستخدمها بكثر، هذه الفئة توفر دالة Lerp تتيح لك الانتقال من موقع دوران الى موقع دوران، ففي حالتنا هذه سننقل المدفع من موقع دورانه الحالي الى موقع الماوس المخزن في متغير من نوع Quaternion لان الدالة Lerp تطلب متغيرين من نوع Quaternion و الاخير من نوع float يحدد سرعة الانتقال بين المواقع، اذن لنقم بعمل هذا الكود، لاحظ:

```
1.  void Update()
2.  {
3.          CannonRotation();
4.  }
5.
6.  void CannonRotation()
7.  {
8.          Vector2 mousePos = Input.mousePosition;
```

```
9.          mousePos = Camera.main.ScreenToWorldPoint(mousePos);
10.
11.         float deltaX = transform.position.x - mousePos.x;
12.         float deltaY = transform.position.y - mousePos.y;
13.         float angle = Mathf.Atan2(deltaX,-deltaY) * Mathf.Rad2Deg;
14.
15.         Quaternion rotation = Quaternion.AngleAxis(angle + 90, Vector3.forward);
16.         transform.rotation = Quaternion.Slerp(transform.rotation,rotation,Time.deltaTime);
17. }
```

فكر الان كيف تكون هذا الكود وماهي آلية عملة؟
ان هذا الكود بشكل كبير يعتمد على استخدام الفئة Quaternion و التي قمت بتخزين مواقع الحركة و الدوران و الظل فيها، لاحظ انه من السطر 8 الى 13 هو الكود السابق الذي عملنا علية، اما من السطر 15 لاحظ انني عرفت متغير من نوع Quaternion باسم rotation و الذي يخزن موقع الماوس و يتم تحويل الحركة الى الامام forward اعتماداً على محاور البرنامج، المتغير forward يخزن فيه قيم المحور z لو لاحظت في الالعاب الـ3D فأن المحور z هو المحور الذي يقوم بتحريك الاجسام الى الامام و الخلف اما باقي المحاور فأنها 2D و كما نعمل عليها الان (x, y) في المتغير rotation استخدمت الدالة AngleAxis و هي ضمن الفئة Quaternion، ان هذه الدالة تقوم بتحويل قيمة الدوران الى محور معين و هذه الدالة تطلب متغير من نوع float و آخر من نوع Vector3 اي بمعنى آخر (قيمة الدوران تعود الى محور Vector3) و هنا تتوفر معنا قيمة الدوران angle و المحور المطلوب هو Vector3.forward يمكنك اختبار باقي المحاور لتأخذ فكرة اكبر عن الموضوع، الان تبقى لنا السطر الاخير 16 و يجب ان نعيد استخدام الدالة rotation التي تخزن قيمة الدوران و على محور معين، لاحظ بشكل مباشر انني استخدمت المتغير transform.rotation و هذا المتغير يطلب متغير من نوع Quaternion و بما اننا نريد استخدام الدالة Lerp فأننا استخدمنا الفئة Quaternion و منها الدالة Lerp، في الدالة يجب تحديد موقع الدوران الحالي وهو transform.rotation و تحويلها الى متغير الدوران الثاني وهو من نوع Quaternion و الموجود معنا rotation و يخزن موقع الماوس، اما بالنسبة للسرعة فحالياً هي مضروبة في وقت البرنامج لهذا يمكنك استخدام متغير يحدد سرعة الانتقال بكل بساطة.

ان هذه الطريقة غالباً يتم استخدامها في الالعاب الـ 3D لكن بما انها تعمل على الالعاب الـ 2D فليس هناك مشكلة ابداً، في النهاية اتمنى ان تعجبك الطريقة الثانية و ان تكون سهله لكي تفهمها بسرعة .

الطريقة الثالثة (الاشعة):

طريقة الاشعة هي الطريقة التي استخدمتها في اول بداية لتصميم اللعبة ولكن عرفت انها طريقة خاطئة لتدوير المدفع لهذا اردت مناقشة هذه الطريقة و للعلم انها تعمل مع الالعاب الـ 3D بشكل جيد دون مشاكل فيها تعتمد على استخدام الدالة transform.LookAt() و هي غير مستخدمة في الالعاب الـ2D لأنها تدور على جميع المحاور في عملية النظر او بشكل اوضح تطلب متغير من نوع Vector3.

مفهوم الاشعة في اليونتي يعني استخدام فئات مثل Ray و RaycasHit ايضاً Physics تساعد على خلق شعاع صادر من جسم معين.

ان لهذا الشعاع عدة خصائص فهو يقوم بحساب النقطة التي لمسها و هي تعتمد على مكون كولايدر ليحدد المكان الذي خبط فيه و تعيد هذه القيمة في المتغير point، يتم استخدام الفئة RaycasHit لاستخراج معلومات التصادم و المماس و تستطيع من خلالها تحديد طول لهذا الشعاع، ايضاً يتيح لك التلامس مع Layer معين كي لا يختلط مع باقي الـ Layers في البرنامج، اذن الاشعة فعالة في عملية النظر، ليس فقط النظر بل الاشياء اخرى كثير لن يتسع الكتاب لشرحها ولكن سأناقش بعضً منها في الدروس القادمة، على العموم لاحظ هذا الكود:

```
1.  void FixedUpdate(){
2.
3.          LookAtMousePoint ();
4.  }
5.
```

```
6.  void LookAtMousePoint()
7.  {
8.          Ray ray = new Ray();
9.          ray = Camera.main.ScreenPointToRay(Input.mousePosition);
10.         RaycastHit hit;
11.
12.         if(Physics.Raycast(ray, out hit, Mathf.Infinity))
13.         {
14.                 transform.LookAt(hit.point);
15.         }
16. }
```

اولاً يجب ان أشير الى ان هذا الكود فيزيائي بكل معنى الكلمة من السطر 8 الى السطر12 جميعها اكواد فيزيائية و لهذا ناديت الدالة في FixedUpdate بكل بساطة، في السطر 8 لاحظ انني عرفت متغير من نوع Ray و هذا مشيد(Constructor) و يستخدم لإنشاء اتجاه لهذا الشعاع و اشياء اخرى و للعلم ان Ray هو متغير من نوع Vector3، الان لاحظ في السطر اسفله انني استخدمت الدالة ScreenPointToRay و هذه الدالة تطلب متغير من نوع Vector3 Position و بما انني استخدمت المتغير mousePosition فهنا سيقوم بجعل موقع الماوس هو النقطة التي سيطلع منها الشعاع، الان لكي نجلب معلومات التصادم فأننا اولاً نعرف متغير يمثل الفئة RaycastHit، في السطر 12 قمت بالتحقق في حالة من ان هذا الشعاع صادر ام غير صادر بالفعل، الدالة Raycast من الفئة Physics تساعدك على معرفة الشعاع و معلومات التصادم و المماس، ايضاً الطول و ان اردت يمكنك استخدام Layer معين، الان في حالة تحققت هذه الشروط سيقوم الكائن بالنظر الى موقع الشعاع اعتماداً على النقطة التي وصل اليها الشعاع او النقطة التي خبط فيها الكولايدر لهذا تخزن القيمة في المتغير point و كما هو موضح في السطر 14.

اذن ما هو رأيك بخصوص هذه الطريقة؟
غير مفيدة حالياً لكن قد تنفع في بعض الاحيان و انا شخصياً افضل عدم استخدامها في الالعاب الـ 2D و هذا لا يعني ان الاشعة لا تستخدم في الالعاب الـ 2D بل يونتي يوفر فئات مثل Ray2D, RaycastHit2D, Physics2D, BoxCast و اشياء كثيرة و لك امر استكشافها و مراجعتها و خير دليل هو مرجع Unity Scripting API.

برمجة العجلة (Wheel):

بالنسبة لتحريك العجلة فيجب علينا تحريكها فيزيائياً اي باستخدام دوال فيزيائية و يوجد لها طريقتان اكثر من سهله، اولاً دعنا نناقش طريقة حركة العجلة، تلاحظ ان لعبتنا من نوع 2D و ان في هذه الحالة العجلة تتحرك على محور واحد x اي الافقي Horizontal و تحركها يكون بالسهمين الايمن و الايسر او A, D، ايضاً تلاحظ ان حدود حركة المدفع غير موجودة اي انه في حالة خرج المدفع من المشهد سيسقط و لن يعود مجدداً و لهذا يجب علينا عمل حدود لحرك المدفع، قد ربما تخطر في بالك فكرة وضع حدود باستخدام المتغير transform.position.x ووضعة في شرط ومن خلاله تستطيع وضع الحدود التي تريدها، طريقة اخرى هي وضع Box Collider2D في طرفي المشهد بحيث ان المدفع عندما يرتطم بالكولايدر لن يخرج من المشهد، فكر في اي الطريقتين افضل؟

صراحةً لن اعطيك الجواب دون التجربة لهذا لنقوم باستخدام الطريقتين، في البداية سنستخدم الطريقة الاولى و هم عمل حدود باستخدام المتغير transform.position.x، لكن قبل هذا كلة دعنا نبدأ بتحريك العجلة لكي نستطيع اختبار حدودها.

اولاً قم بأنشاء ملف برمجي باسمWheelController بحيث ان هذا الملف البرمجي سيمثل حركة العجلة، اذن قم بفتح هذا الملف و اكتب اكود التالي لكي اقوم بشرحة لك:

```
1.          private Rigidbody2D rb2D;
2.          public float Speed;
3.          public static bool MoveWheel;
4.
5.          void Start()
```

```
6.      {
7.              rb2D = GetComponent<Rigidbody2D>();
8.              MoveWheel = true;
9.      }
10.     void FixedUpdate()
11.     {
12.             if(MoveWheel)
13.             {
14.                     WheelMovement();
15.             }
16.     }
17.
18.     void WheelMovement()
19.     {
20.             float MoveX = Input.GetAxis("Horizontal");
21.             Vector2 movement = new Vector2(MoveX, 0);
22.             rb2D.AddForce(movement * Speed * Time.deltaTime);
23.     }
```

حسناً دعنا نناقش هذا الكود، لاحظ في البداية الثلاثة الاسطر الاولى و التي عرفت فيها متغير من نوع Rigidbody2D يمثل المكون في العجلة اما الثاني من نوع float و هو يحدد سرعة الحركة لهذا للعجلة، و اخيراً الثالث متغير من نوه bool و لكن static لكي يتيح لي التعامل معه خارج هذا الملف لكي نستطيع ايقاف حركة العجلة عندما تنتهي اللعبة، اذن ثلاثة متغيرات اولية ضرورية لتحريك العجلة.

في Start قمت بجلب مكون Rigidbody2D ايضاً اعطيت اول قيمة افتراضية للمتغير MoveWheel ففي حالة true العجلة ستتحرك و false لن تتحرك بكل بساطة، بما اننا سنتعامل مع كود فيزيائي فيجب انشاء الدالة FixedUpdate و التي ناديت فيها الدالة WheelMovement التي تحتوي على اكواد حركة العجلة، دعنا ندخل الى الدالة و نرى مافيها؟

لاحظ في البداية عرفت متغير من نوع float باسم MoveX هذا المتغير يخزن فيه الحركة على المحور x باستخدام المحور Horizontal الافقي و هنا يجب تخزين المتغير MoveX في متغير من نوع Vector2 لكي نستطيع اضافة قيم المتغير الى المحور x، في السطر 21 قمت بتغير متغير من نوع Vector2 و قمت بوضع المتغير MoveX فيه، هنا سيتم تخزين جميع القيم الحالية في المتغير movement و الذي يمثل حركة العجلة على المحور x لكن هنا يجب استخدام هذا المتغير في دالة فيزيائية اي من نوع Rigidbody2D، تجد في اسفلة انني استخدمت الدالة AddForce و التي ستضيف قوة الى العجلة اعتماداً على المتغير movement و للعلم ان هذه الدالة تطلب متغير من نوع Vector2 لتتمكن من وضع القوة على احد المحاور او كلاهما معاً، لاحظ انني استخدمت المتغير Speed لكي نستطيع التحكم بسرعة العجلة بكل بساطة.

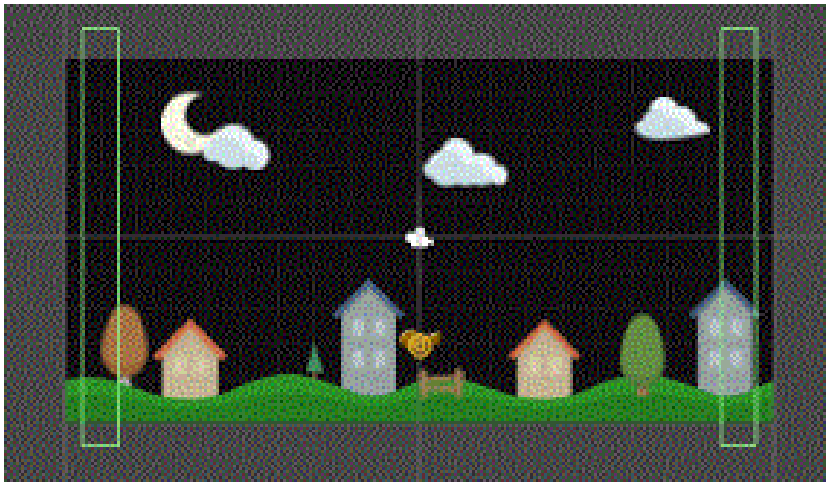
بالنسبة للمتغير MoveWheel فهنا قمت بربطة بالدالة بشكل مباشر و للعلم ان هذا المتغير لن يظهر في الانسباكتور و حتى ان كان public لأنه ثابت و لكن هناك فرق، في حالة كان public static فهنا سيكون ثابت و لكن ستستطيع استخدامه خارجياً اي في ملف آخر عبر فئته مباشرةً، و لكن ان كان private static فلن تتمكن من استخدامه في ملف آخر عبر فئته مباشرةً لهذا سنستخدمه في صيغته العامة لكي نستطيع التعامل معه لاحقاً، القيمة الافتراضي في Start لهذا المتغير هي ضرورية جداً لكي تتمكن من معرفة قيمة الحالة قمت بتشغيل اللعبة. ان هذا الكود سهل التعامل و مفهومة هو وضع الحركة في دالة فيزيائية تعطي قوة على هذا المحور اعتماداً على متغير يحدد السرعة الخطية له فقط!

تبقى لنا وضع حدود لحركة المدفع و لكن اولاً قم بوضع قيمة المتغير Speed كبيرة لكي تستطيع اختبار مدى فعالة الحدود، اذن لنقوم بصنع هذه الحدود، الطريقة الاولى هي باستخدام المتغير transform.position و لنجربها الان، اضف هذه الدالة الى الملف:

```
1.  void WheelLimits()
2.  {
3.          if(transform.position.x <= -8.3f)
4.                  transform.position = new Vector2(-8.3f, transform.position.y);
5.          if(transform.position.x >= 8.3f)
```

```
6.                transform.position = new Vector2(8.3f, transform.position.y);
7.    }
```

الدالة بسيطة جداً فقط عرفت شرطين الاول في حالة كان موقع العجلة اصغر من 8.3f- و هي القيمة التي اراها مناسبة في مشهدي ستضل العجلة في نفس الموقع، اما في حالة كانت اكبر من 8.3f ستصبح العجلة في مكانها، لكن هنا المشكلة تكمن في اننا نقوم بتحريك العجلة باستخدام دالة فيزيائية، لهذا في حالة وصلت العجلة الى اقل او اعلى قيمة ستضل القوة موجودة في العجلة و حتى في حالة وصل الى نهاية المطاف ستضل العجلة تدور و لن تستطيع تحريك المدفع للجانب الاخر الا في حالة توقفت العجلة، هذه المشكلة ربما قد تزعجك اثناء اللعب، لهذا يوجد حل آخر لها و هو بسيط جداً فقط كل ماعليك فعلة هو وضع Box Collider2D في طرفي المشهد كما هو موضح في الصورة:



لاحظ الكولايدران في الصورة و هما يمثلان حدود المشهد على اليمين و اليسار، و عند الاصطدام بهما فأن المدفع سوف يرتطم بهما، قد تسألنا ان كانت هذه هي الطريقة المثالية لوضع حدود للمشهد؟ سأجيبك بان هناك العديد من الطرق و لكن اجد ان استخدام هذه الطريقة في اللعبة هي جميلة، المشكلة الان ان في حالة وضعت الكولايدران بهذه الطريقة فسوف يرتطم به الاعداء لهذا قم بوضع Layer يمثل هذه الحدود و افصل ارتطامه بالأعداء بكل بساطة.

الى هنا اضن ان الامر بات اوضح لك، اذن عملية تحريك العجلة قد انتهت، لهذا سننتقل الى اطلاق النار من المدفع و بعض الخصائص الضرورية له.

مؤشر اطلاق النار (الهدف Aim):

بالحديث ان اطلاق النار فهنا سيتوجب علينا وضع بعض الاولويات الضرورية لكي قوم اضافة الطلقات بشكل جميل، ان مؤشر الهدف ضروري في اي لعبة تصويب و هو الذي يساعدك في تحديد الهدف او التأشير علية، عند تشغيل اللعبة ستوجب علينا اخفاء مؤشر الماوس الافتراضي ووضع مؤشر هدف بدلة، هنا يجب تحديد موقع المؤشر بحيث يكون في موقع مؤشر الماوس الافتراضي في المشهد، اذن لنقم بعمل مؤشر الهدف الخاص بالمدفع، بالنسبة لطرق وضع هذا المؤشر بدل مؤشر الماوس فان هناك طريقتان اجدهما مناسبتان جداً سنناقشهما الان .

اولاً للعثور على المؤشرات قم بالبحث عن الملف Aim ستجد فيه العديد من المؤشرات و التي تختلف في الحجم، شخصياً اخترت المؤشر crosshair_red_large في حالة لم يعجبك فبكل بساطة اختر واحداً آخر، اذن لنقم بعملية وضع مؤشر الهدف بدل مؤشر الماوس.

الطريقة الاولى (Rect):

الطريقة الاولى قم بأنشاء ملف برمجي ثاني باسم PlayerShooting بحيث ان هذا الملف سوف يكون الملف الخاص بأطلاق النار بالنسبة للمدفع لهذا قم بوضعة في مجلد Player مع باقي الملفات و تأكد من سحبة الى المدفع، الان قم بفتح هذا الملف و قم بكتابة الكود التالي فيه:

```
1.  public float heightAim;
2.  public float widthAim;
3.  public Texture2D Aim;
4.  public static bool ShowCursor;
5.
6.  void Start()
7.  {
8.      Time.timeScale = 1;
9.      ShowCursorInLoad();
10. }
11.
12. void OnGUI()
13. {
14.     if(ShowCursor)
15.     {
16.         Rect rect = new Rect
17.             ((Input.mousePosition.x - Aim.width/widthAim/2),
18.              (Screen.height - Input.mousePosition.y - Aim.height/heightAim/2),
19.              (Aim.width/widthAim),
20.              (Aim.height/heightAim));
21.         GUI.DrawTexture(rect,Aim);
22.         Cursor.visible = false;
23.
24.     }else Cursor.visible = true;
25. }
26.
27. void Update()
28. {
29.     ShowCursorInLoad();
30. }
31.
32. void ShowCursorInLoad()
33. {
34.     if(Time.timeScale == 1)
35.         ShowCursor = true;
36.     if(Time.timeScale == 0)
37.         ShowCursor = false;
38. }
```

في البداية دعني واضح لك شيء، ان هذا الكود لا يظهر المؤشر فقط بل ايضاً يحدد متى يظهر و متى يختفي اعتماداً على قيمة Timescale و التي تتحكم بتوقيف و تشغيل المشهد، في البداية قمت بتعريف 4 متغيرات بحيث ان 2 من نوع float يمثلان الطول و العرض للهدف ايضاً الموقع، اما الثالث فهو خاص بصورة الهدف و الرابع خاص بإظهار مؤشر الماوس و إخفائه، اربعة متغيرات ستساعدنا على وضع هذا المؤشر في اللعبة.

في Start حددت ان قيمة Timescale تساوي واحد اي ان الوقت سيشتغل عند بداية الله، ان هذه العملية ضرورية جداً و دائماً تتغير قيمة Timescale عند ايقافه و تشغيله و لن تلاحظ هذا الان و لكن سنحتاج اليه لاحقاً، اسفلة ناديت الدالة ShowCursorInLoad و هذه الدالة وضيفتها اظهار و اخفاء المؤشر اعتماداً على قيمة Timescale، في الدالة OnGUI وضعت شرط و هذا الشرط مربوط بدورة في قيمة Timescale، استخدمت افئه Rect او بالأصح عرفت متغير يمثل الفئة Rect و هذه الفئه تتيح لك ضبط موقع كائن 2D على الشاشة

بالضبط على المحورين X, Y و هي تطلب 4 بارامترات من نوع float بحيث ان الاول يحدد موقع الكائن على المحور x و الثاني على محور y اما لثالث و الرابع فهما مختصان بطول و عرض الكائن، تلاحظ انني مباشر استخدمت الفئه، حسناً دعنا نناقش امرها، تلاحظ في البداية قمت بفتح الاقواس لكي ارتب الامر اي ان شكل الدالة البداية هاكذا **Rect((left),(top),(width),(height))** و الان نستطيع وضع كل قيمة في مكانها بهذا الشكل، في البداية يجب تحديد موقع الهدف على المحور x و لاحظ في السطر 17 انني انقصت الفارق بين موقع الهدف و موقع الماوس و قسمت الناتج على قيمة عرض الهدف و قسمت الناتج الاخير على 2 السبب هو انه كلما زاد حجم الهدف سيتغير موقعة و لهذا قمت بقسمة موقع الهدف على عرضة و ايضاً هذا مافعلته في السطر 18 و تلاحظ انني اقصت الفراق على ارتفاع الشاشة لان الارتفاع يختلف عن العرض لهذا لكي لا تظهر مشكلة تغير موقع الهدف على ارتفاع الشاشة، في السطرين 19, 20 حددت طول و عرض الهدف و قسمت الناتج على طول و عرض الهدف كما هو موضح، ان هذه العملية ستتيح لك تغيير حجم الهدف و عند تشغيل اللعبة ستلاحظ ان موقع الهدف لم يتغير مع تغير حجمة و هذه الطريقة ستختصر عليك عناء ضبط موقعة.

في السطر الاخير 21 استخدمت الدالة DrawTexture و هي متخصصة في اظهار صورة معينة في الشاشة وهي تطلب متغيرين الاول من نوع Rect position يخزن فيه موقع الصورة اما الثاني فهو متغير من نوع Texture2D و بكل بساطة هي الصورة التي تريد اظهارها، تبقى شيء اخير و هو اخفاء المؤشر الافتراضي (مؤشر الماوس) لكي تستطيع رؤية الهدف فقط، في السطر 22 استخدمت الفئة Cursor و هي توفر العديد من الدوال و المتغير التي تختص بالمؤشر و هي ايضاً توفر متغير من نوع bool يساعد على اظهار و اخفاء مؤشر الماوس اعتماداً على قيمته الحالية و كما تلاحظ في السطر 22 جعلت قيمة المتغير visible = false اي المؤشر سيختفي، اما امر اظهار المؤشر فهو على قيمة Timescale، اذن ماهو رأيك بهذه الطريقة؟ البست صعبة قليلاً؟ لابأس و لكن اردت مناقشتها لكي تستوعب طريقة وضع مؤشر الماوس باستخدام الدالة OnGUI و لكن هذه الطريقة كانت تستخدم في النسخ القيمة لهذا في النسخة الجديدة (من Unity4.6 الى Unity5) ستجد دالة توفر عناء وضع مؤشر الماوس بدل المؤشر الافتراضي و هذا ماسنناقشة في الطريقة الثانية.

الطريقة الثانية (Cursor):

بالنسبة لهذه الطريقة فأنا اجدها طريقة مختصرة لوضع هدف بدل مؤشر الماوس الافتراضي، ان الفئة Cursor توفر لنا بعض الدوال و اهمها المتغير visible وهو من نوع bool كما شرحته سابقاً، ايضاً الدالة SetCursor و هي تساعدك على تبديل المؤشر الحالي او بالأصح الحصول على مؤشر، حسناً سنستخدم الدالة SetCursor و هي ستختصر لنا الامر.

ان هذه الدالة تطلب 3 متغيرات بحيث ان الاول من نوع Texture2D و الذي يحتوي على مؤشر الهدف، المتغير الثاني من نوع Vector2 و هو يحدد موقع الهدف اعتماداً على المحورين x, y، اما الاخير فهو من نوع CursorMode يحدد نوع او حالة المؤشر و توجد فيه حالتين، الاولى Auto اي الحالة الاعتيادية في للمؤشر اي سيكون متاح للظهور في الشاشة، اما الثاني ForceSoftware يعتمد على دقة المؤشر، لكن في حالة تغير موقع الهدف فسوف نقوم بضبط موقعة، اذن لنجرب هذه الدالة، تأمل هذا الكود:

```
1.  public float aimWidth;
2.  public float aimHeight;
3.  public Texture2D Aim;
4.  public static bool ShowCursor;
5.  public CursorMode cursorMode = CursorMode.ForceSoftware;
6.
7.  void Start()
8.  {
9.      Time.timeScale = 1;
10.     ShowCursorInLoad();
11. }
12.
13. void Update()
14. {
15.     ShowCursorInLoad();
16.
17.     if(ShowCursor)
```

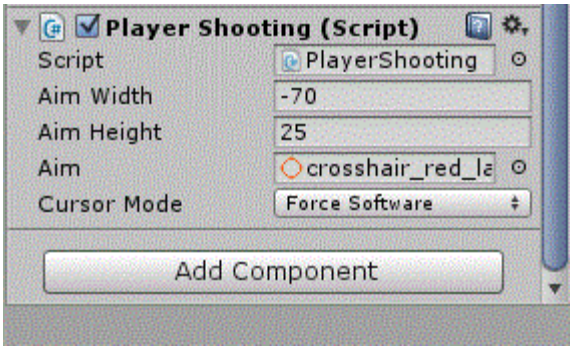
```
18.                 {
19.                         Cursor.SetCursor (Aim,new Vector2(aimWidth, aimHeight),cursorMode);
20.                 }else{
21.                         Cursor.SetCursor (null, Vector2.zero, cursorMode);
22.                 }
23.         }
24.
25.         void ShowCursorInLoad()
26.         {
27.                 if(Time.timeScale == 1)
28.                         ShowCursor = true;
29.                 if(Time.timeScale == 0)
30.                         ShowCursor = false;
31.         }
```

حسناً دعنا نناقش هذا الكود، في البداية تلاحظ انني متغيرين من نوع float يحددان موقع الهدف على المحورين، ايضاً عرفت متغير آخر من نوع Texture2D و هو الهدف، اسفلة عرفت متغير من نوع bool و شرحته سابقاً، المتغير الاخير من نوع CursorMode وهو يحدد نوع المؤشر، اذن 5 متغيرات اولية ضرورية.

في Start تجد القيمة الاولية نفسها، اما في Update فتلاحظ انني استخدمت الفئة Cursor و منها الدالة SetCursor حسناً هذه الدالة يجب ان تحدد موقع المؤشر الحالي و تستبدله بالهدف، فيها حددت اول قيمة و هي صورة المؤشر من نوع Texture2D، اما المتغير الثاني فهو من نوع Vector2 و تلاحظ انني استخدمت متغيري الطول و العرض فيه لان في بعض الاحيان تجد ان المؤشر لا ينطبق على موقع الماوس لهذا سيتوجب علينا تغيير موقعة اعتماداً على المتغيرين، في الاخير يطلب منا نوع او حالة المؤشر و بشكل مباشر نستخدم المتغير cursorMode و افتراضياً هو على الوضعية ForceSoftware و اجدها مناسبة مع اللعبة.

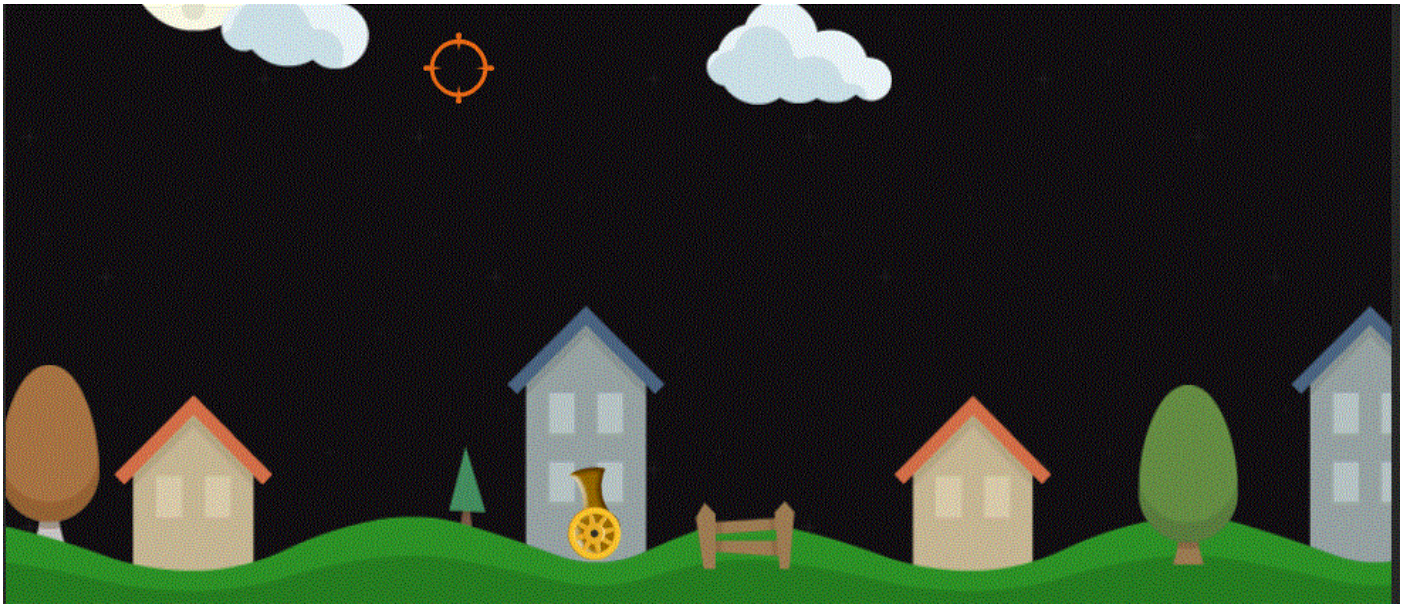
هذا كان في حالة تم التحقق من ان قيمة المتغير ShowSursor اصبحت true، ففي حالة لم تصبح true فهنا يجب اخفاء مؤشر الماوس و بشكل مباشر نضع نفس الدالة و نستبدل المتغير Aim بـ Null و هذه القيمة تعني فارغ اي لاوجود لأي مؤشر او هدف لهذا سيضع المؤشر الافتراضي للماوس ايضاً اعدت موقعه الى منتصف الشاشة و بهذا الشكل نعيد المؤشر الافتراضي للماوس.

ان هذه الطريقة سهله و جميلة ايضاً تستطيع التحكم بها بسهولة، الان بالعودة الى المشروع ستجد المتغيرات التي أنشأناها:



لاحظ نوع المؤشر يظهر على شكل enum و يمكنك تغييرة و هو حالتان، ايضاً لاحظ انني سحبت صورة المؤشر الى مكانة، و بالنسبة لموقع المؤشر فاجد ان الارقام الظاهرة هي مناسبة في اللعبة، و تستطيع تغييرها حسب ما يتناسب معك.

الان قم بتجربة اللعبة و ستلاحظ ظهور المؤشر في الشاشة بشكل جيد ☺.

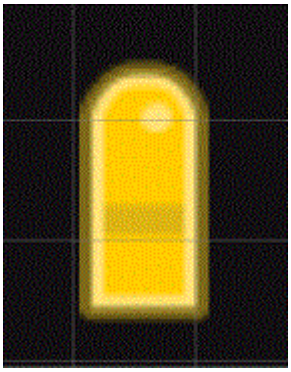


تجهيز و برمجة الطلقات:

في هذه الفقرة سنناقش فكرة اطلاق النار من المدفع، سيتوفر لذينا ثلاث انواع من الطلقات بحيث ان لكل طلقة مدى تأثيرها الخاص في الاعداء و البيئة المحيطة، اذن لنقم بتحضير الطلقات.

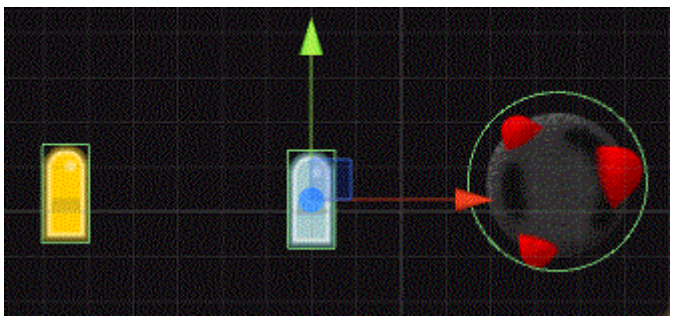
تجهيز الطلقات:

تجهيز طلقات اللاعب فقط، اذن لنقم بالبحث عن طلقات اللاعب و نضع لها المكونات، بالنسبة للطلقة الاولى قم بالبحث عن الصورة icon_bullet_gold_long هذه ستكون الطلقة الاولى، الان اسحبها الى المشهد لكي نعدل عليها و يجب ان تكون هذه:

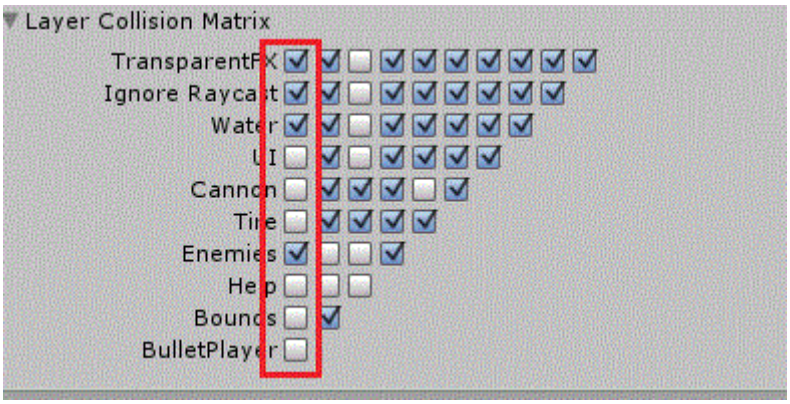


اذن هذا هو شكلة الطلقة الاولى، حسناً دعنا نضيف اليها المكونات، في البداية قم بإضافة Box Collider2D، ثم قم بإضافة المكون Rigidbody2D، اجعل الجاذبية (Gravite) تساوي صفر، بالنسبة للـ Layer قم باضافة Layer جديد باسم BulletPlayer، و هذا سيمثل جميع الطلقات للاعب، ايضاً اضف تاج جديد باسم b_Player_1 اي الطلقة الاولى للاعب، و سننشئ باقي الطلقات الان، سيتبقى لنا تعديل التصادمات لكن سنتركه لأخر الفقرة.

بالنسبة للطلقة الثانية قم بالبحث عن الصورة icon_bullet_silver_long الان قم بوضع نفس المكونات بالاضافة الى الـ layer و لكن تأكد من ان التاج هو b_Player_2 لأنها الطلقة الثانية، الطلقة الاخيرة ستكون عبارة عن قنبلة و هي فعالة جداً لهذا يجب علينا وضع صورة قنبلة، اذن قم بالبحث عن الصورة Boom و ستجد صورة لقنبلة، جيد قم بوضع نفس المكونات و قم بتغيير الـ Box Collider2D الى Circle Collider2D، بالنسبة للـ Tag تأكد من انه b_Player_3 اما بالنسبة للـ Layer فأتركة.



الى اهنا نكون قد انهينا تجهيز طلقات اللاعب و سيتبقى لنا تحديد التصادمات و برمجة الطلقات، اذن لنقم بتحديد التصادمات، بالدخول الى Edit> Project Setting> Physics2D ستظهر لك النافذة التالي:



لاحظ القائمة المحددة باللون الاحمر، ان هذه القائمة هي قائمة Layer الطلقات (BulletPlayer) و تجد انها لا تتصادم مع الـ Cannon, Tire, UI, Help, Bounds, BulletPlayer، و الاهم من هذا انها تتصادم مع الأعداء او لاير الاعداء (Enemies)، بهذا الشكل نكون قد انهينا العمل على طلقات اللاعب و سيتوجب علينا برمجتها، و لكن تأكد من ان جميع الطلقات على شكل prefab اي كل طلقة لها prefab لكي نستطيع استنساخها عند اطلاق النار.

اطلاق النار:

وصلنا الى الفقرة الاهم و التي تعتمد بشكل كبير على الفقرة السابقة و من خلال كل طلقة سنستطيع تحديد قوة كلاً منها، في هذه الفقرة سنناقش طريقة اطلاق النار ايضاً التبديل بين الطلقات و عمل الذخيرة و غيرها من الامور التي سنمر عليها، اطلاق النار سيعتمد على الملف PlayerShooting و اسمة يدل على وضيفته لهذا سيكون هو ملف اطلاق النار، اولاً و قبل كل شيء علينا وضع كائن فارغ من خلاله ستظهر الطلقات و يمكننا التحكم بدورانه، اذن مايجب فعلة هو انشاء كائن فارغ (Create Empty) اجعل هذا الكائن ابن للكائن Cannon و تأكد من ان موقعة في **Position = (2.45, 0, 0.1)** بالنسبة للدوران اجعل دورانه في **Rotation = (0, 0, -90)** قم بتسمية هذا الكائن الفارغ Point Shooting اذن قمنا بتجهيز مكان اطلاق النار.

الان بالنسبة لطلاق النار فهناك عدة طرق و شخصياً اجد ان طريقتان هما الاكثر استخداماً و شيوعاً، حسناً دعنا نناقش الطريقتان، بالنسبة للطريقة الاولى فهي عبر استخدام حلقة تتأكد من انك قمت بالضغط على زر معين و تقوم بتطبيق الاطلاق، اما الطريقة الثانية فهي الطريقة العادية بحيث انك في حالة ضغطت على الماوس يتم انقاص الوقت و في حالة كان الوقت صفر سيتم اطلاق النار، صرحةً اجد ان الطريقتان جميلتان و الاختيار يعود اليك، اذن دعنا نناقش الطريقة الاولى.

الطريقة الاولى:

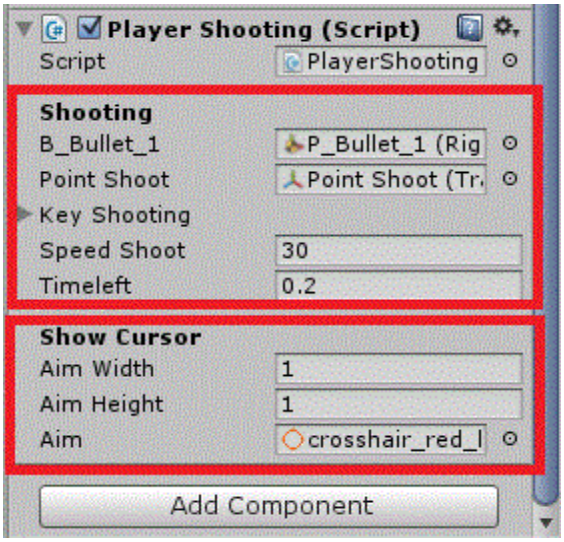
في هذه الفقرة سنناقش اطلاق النار باستخدام الحلقة التكرارية او الدورانية foreach هذه الحلقة تقوم بالدوران على جميع الكائنات التي يتم تعيينها فيها و تتحقق منها و في حالة شرط معين يتم تطبيق الشرط على جميع الكائنات التي تحددنها، بالنسبة لأطلاق النار فسنقوم بالتحديد بالتأكد من اننا ضغطنا على الماوس و في حالة تم التأكد سيتم تطبيق اطلاق النار لكن لن تعمل هذه الحلقة الا في حالة تم التأكد من ان قيمة وقت الاطلاق اصغر او يساوي صفر، ان الشرح بهذا الشكل ربما يشعب الامور عليك لهذا دعنا نناقش الكود التالي:

```
1.  [Header("Shooting")]
2.  public Rigidbody2D b_Bullet_1;
3.  public Transform pointShoot;
4.  public List<KeyCode> keyShooting;
5.  public float SpeedShoot;
6.  public float timeleft = 0.2f;
7.  private float timeShoot = 0f;
8.
9.  void PlayerShoot()
10. {
11.         timeShoot -= Time.deltaTime;
12.
13.         if(timeShoot <= 0)
14.         {
15.             foreach(KeyCode keyCode in keyShooting)
16.             {
```

```
17.
18.                         if(Input.GetKey(keyCode))
19.                         {
20.                             timeShoot = timeleft;
21.                             Rigidbody2D b_prefab Instantiate
22.                             (b_Bullet_1, pointShoot.position, Quaternion.identity) as Rigidbody2D;
23.
24.                             b_prefab.velocity = SpeedShoot * transform.right;
25.
26.                             b_prefab.transform.rotation = transform.rotation;
27.                             break;
28.                         }
29.                     }
30.                 }
31.             }
```

اولاً ان هذا الكود فقط لإطلاق النار بشكل مباشر.

في البداية لاحظ انني وضعت جميع القيم في Header لكي لا تتشعب علي باقي المتغيرات او بالأصح لكي افصل كل وظيفة لوحدها و متغيراتها تضل مفصولة و تستطيع معرفة لمن تعود فلو رجعت لليونتي ستجد ان المتغيرات ظهرت بهذا الشكل:



هذه العملية ستساعدك كثيراً لهذا قم بفصل المتغيرات التي تمثل وظيفة واحدة، ستحتاج لإنشاء العديد من الملفات البرمجية و ستحتاج لفصل وظائفها كلاً على حدة.

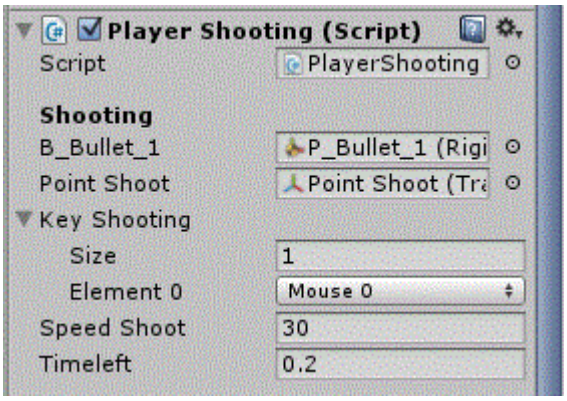
الان لاحظ الستة المتغيرات التي انشأتها و هي ضرورية لعملية اطلاق النار و سأناقشها معكم الان.

في البداية تجد انني عرفت متغير من نوع Rigidbody2D وهو يمثل الطلقة وبما ان طلقتنا تحتوي على مكون Rigidbody2D فيفضل تحريكها بدوال الـ Rigidbody2D. ايضاً لاحظ استخدامي للمتغير Transform وهو يمثل النقطة التي ستنطلق منها الرصاصة. اسفلة تلاحظ انني عرفت متغير من نوع List و هذه القائمة تحتوي على زر الادخال لعملية اطلاق النار و لكن في البداية يجب استيراد نطاق الاسم ليتاح لك استخدام الفئة List و نطاق الاسهم هو **using System.Collections.Generic**؛ و عند استيراده سيتيح لك اليونتي التعامل مع القوائم و اشياء اخرى، هنا انتهينا من المتغيرات الاساسية سيتبقى لنا انشاء متغيرات تمثل سرعة الاطلاق، و الزمن المتبقي بين كل طلقة و اخرى، اذن 3 متغيرات من نوع float تجدها آخر المتغيرات التي انشأتها، فمتغير SpeedShoot يمثل سرعة الطلقة، اما المتغيران timeleft، timeShoot يمثلان الزمن المتبقي و زمن الاطلاق لبدأ الاطلاق من جديد، اذن انتهينا من عمل المتغيرات الاساسية و سيتبقى لنا استخدامها في عملية الاطلاق.

في السطر 9 قمت بأنشاء دالة منفصلة تمثل الاطلاق و بالنسبة لك قم بمناداتها في Update، الان تلاحظ اول كود في الدالة ووظيفته يقوم بانقاصِ الزمن الحالي و في حالة كان الزمن الحالة اقل او يساوي صفر و هو كما موضح في السطر 13 سيتم تطبيق الحلقة foreach و وظيفة هذه الحالقة هو الدوران على جميع الازرار و التأكد من ان الزر في المتغير keyShooting تم الضغط علية سيتم تطبيق عملية الاطلاق اي ان المتغير keyShooting هو وسيط لتحديد نوع زر الادخال لعملية الاطلاق. داخل الحلقة تجد ان سيتم اعادة قيمة المتغير timeShoot الى قيمة المتغير timeleft و هذا المتغير سيكون متاح للظهور في الانسباكتور و تحديد قيمته سيساعدك على معرفة الزمن المتبقي بين كل طلقة و اخرى.

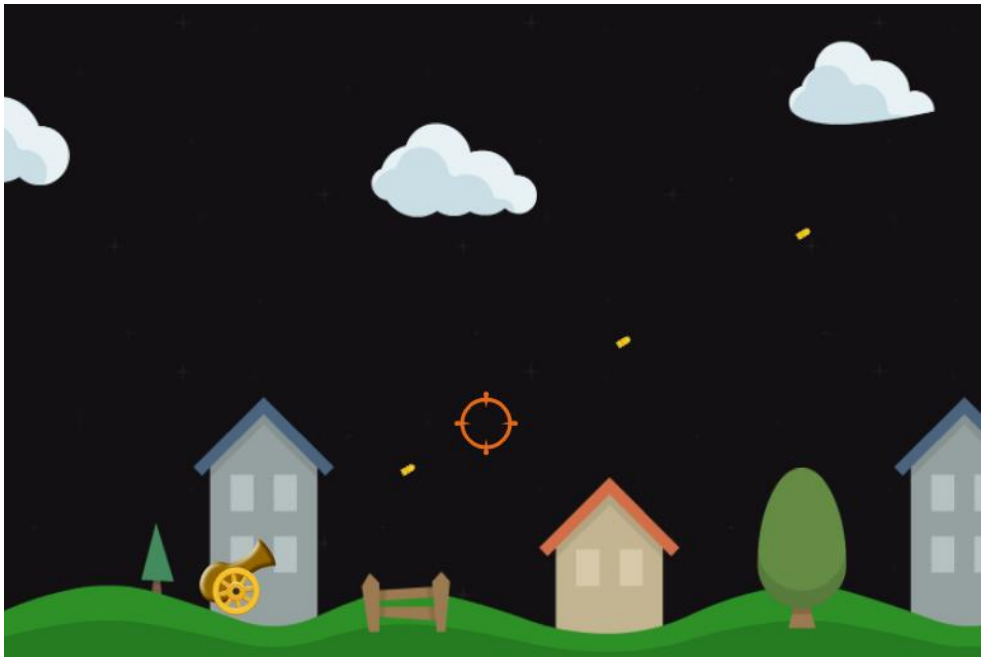
بالنسبة للإطلاق تلاحظ انني استخدمت الدالة Instantiate و هي خاصة باستنساخ الكائنات و لكن تجد انني خزنت قيمة الدالة في متغير من نوع Rigidbody2D السبب لان الطلقة تحتوي على الخواص الفيزيائية و يجب تحريكها بخواصها. الدالة Instantiate تطلب 3 متغيرات بحيث ان الاول يعتبر عن الطلقة و نوع الطلقة يجب ان يتناسب مع نوع الدالة Instantiate كيف هذا؟ لاحظ في نهاية الدالة انني عاملت الدالة كدالة من نوع Rigidbody2D (as Rigidbody2D) و بهذا الشكل سيكون الكائن المستنسخ من نفس النوع، المتغير الثاني الذي تطلبه الدالة هو من نوع Vector3 وهو يحدد مكان طلوع الكائن المستنسخ، اما المتغير الاخير فهو من نوع Quaternion و يحدد دوران الكائن المستنسخ اعتماداً على الكائن الحلي، ان المتغير Quaternion.identity يعبر على ان هذا الكائن سيبقى على محاور التدوير الخاصة به فقط.

اسفله اعدت استخدام المتغير b_prefab وهو يمثل الكائن المستنسخ، الان يجب وضع سرعة لهذا الكائن المستنسخ لعملية الانطلاق، و هذا ما عملة بالضبط استخدمت المتغير velocity بحيث ان هذا المتغير سيعطي سرعة للطلقة عند خرجوها و هذا المتغير يطلب متغير من نوع Vector2 يحدد موقع الكائن عند الانطلاق، لاحظ انني قمت بضرب السرعة في المتغير transform.right و السبب استخدامي لهذا المتغير هو انه سيتم تحديد موقع الطلقة اعتماداً على محاور الكائن نفسة، اما السطر الاخير فهذا اختياري نظراً لشكل الطلقة عندي فأني عينت دورانها مع دوران الكائن لكي تظهر بشكل مناسب، اخيراً استخدمت Break ليعمل توقف مؤقت بين كل طلقة و اخرى.



بالعودة الى المشروع ستجد ظهور المتغيرات التالية، الان قم بملئها كم هو موضح و تأكد من تحديد زر اطلاق النار فالخيار Mouse 0 تعني الزر الايسر للماوس اما 1 فهو الزر الايمن، بالنسبة لسرعة فهذه القيمة اجدها مناسبة عندي ايضاً الوقت المتبقي، الى هنا تأكد من القيم التي تراها مناسبة و استمتع بأطلاق النار ☺

.



الطريقة الثانية:

الطريقة الثانية لإطلاق النار هي عبر تحديد زر معيناً برمجياً و تحديد وقت معين فهي لا تختلف كثيراً عن الطريقة الاولى و لكنها سهل كثيراً، ميكانيكية عمل هذه الطريقة تعتمد الضغط على الزر لتتم عملية انقاص الوقت و في حالة اصبح الوقت اصغر او يساوي صفر فهنا سيتم تطبيق عملية الاطلاق، اذن لاحظ هذا الكود لتتعرف على هذه الطريقة:

```
1.  void PlayerShoot()
2.  {
3.
4.      if(Input.GetMouseButton(0))
5.      {
6.          timeShoot -= Time.deltaTime;
7.          if(timeShoot <= 0)
8.          {
9.              Rigidbody2D b_prefab = Instantiate
10.             (b_Bullet_1,pointShoot.position, Quaternion.identity) as Rigidbody2D;
11.             b_prefab.velocity = SpeedShoot * transform.right;
12.             b_prefab.transform.rotation = transform.rotation;
13.             timeShoot = timeleft;
14.         }
15.     }
16. }
```

حسناً ماذا ترى من هذا الكود؟ لاحظ انه من السطر 9 الى السطر 13 هو نفس الكود السابق، الشيء المختلف قليلاً ان هذا الكود يبدأ بشرط يتحقق من انه تم الضغط على الزر الايسر للماوس سيتم اناقص الوقت، اي ان عملية انقاص الوقت مربوط بتحقق الشرط الاول كما هو واضح، الان في حالة اصبحت قيمت المتغير timeShoot اصغر او يساوي صفر سيتم تطبيق عملية الاطلاق و سيتم اعادة قيمة الوقت الى قيمة المتغير timeleft فقط.

ان الكود فعلاً سهل و ربما لا يحتاج الى شرح مفصل فمن رويتك للاكواد تتكرر يدل على انه لم يتم تغيير شيء الا القليل، و بهذا الشكل نكون قد انينا العملي على اطلاق النار و سيتبقى لنا تجهيز الذخيرة و هذا ما سنناقشه في الفقرة القادمة ☺ .

الذخيرة:

في العاب اطلاق النار تجد ان الذخيرة هي عامل اساسي لتحفيز اللعب، ربما تضل عالق في اللعبة بسبب انك تبحث عن الذخيرة و هذا شيء جميل لكونك زدت من صعوبة اللعبة بحيث انك استخدمت الذخيرة كعذر لإطالة عمر المرحلة.

بالنسبة للعبتنا فلن نحتاج الى تعبة الذخيرة يدوياً اي سيتم تعبة الذخيرة تلقائياً و سيتوجب عليك قتل الأعداء للحصول على الذخيرة و بعض الاضافات التي تخرج بشكل عشوائي، حالياً سنقوم ببرمجة الذخيرة و تهيئتها لاستدعائها لاحقاً و التعامل معها، في اول طريقة سيكون اقصى عدد للذخيرة هو 60، و تعبئة الذخيرة الاولى سيتم تلقائياً عبر اللعبة اي في حالة نقصت قيمة الذخيرة سيتم الحصول على ذخيرة من اللعبة دون الحاجة لقتل الاعداء الى ان هذه الطلقة ستكون غير فعالة و مهمتها فقط هي تغطية الجانب المتعب لعملية البحث عن الذخيرة لهذا ستستخدمها للحصول على ذخيرة للطلقات الاخرى.

اذن لقد اخدت فكرة عن وظيفة الذخيرة للطلقة الاولى، اذن في البداية سيتم تعرف متغير من نوع enum يتحقق من نوع الطلقة لكي يتم تنفيذ ذخيرتها. اذن بهذه الطريقة نستطيع تنظيم الكود و التحقق من نوع الطلقة الحالة للاعب. حالياً يجب تعريف متغيرات جديدة تمثل الطلقة و اقصى حد للطلقة و نوع الطلقة لكي نستطيع وضع الذخيرة فيها، لاحظ هذا الكود:

```
1.  [Header("Ammo Count")]
2.  public static int Ammo_1;
3.  public int maxAmmo = 60;
4.  public enum TypeBullet{
5.          Bullet_1,
6.          Bullet_2,
7.          Bullet_3
8.  }
9.  public TypeBullet typeBullet;
10.
11. void Start()
12. {
13.         Ammo_1 = maxAmmo;
14.         typeBullet = TypeBullet.Bullet_1;
15. }
16.
17. void PlayerShoot()
18. {
19.         timeShoot -= Time.deltaTime;
20.
21.         Ammo_1= Mathf.Clamp(Ammo_1, 0, maxAmmo = 60);
22.
23.         if(typeBullet == TypeBullet.Bullet_1)
24.         {
25.                 maxAmmo = 60;
26.                 Timeleft = 0.2f;
27.
28.                 if(Ammo_1> 0)
29.                 {
30.                         if(timeShoot <= 0)
31.                         {
32.                                 foreach(KeyCode keyCode in keyShooting)
33.                                 {
34.
35.                                         if(Input.GetKey(keyCode))
36.                                         {
37.                                                 timeShoot = timeleft;
38.                                                 Rigidbody2D b_prefab = Instantiate
39.                                 (b_Bullet_1 ,pointShoot.position, Quaternion.identity) as Rigidbody2D;
40.                                                 b_prefab.velocity = SpeedShoot * transform.right;
```

```
41.                                                 b_prefab.transform.rotation = transform.rotation;
42.
43.                                                 Ammo_1-= 1;
44.
45.                                                 break;
46.                                             }
47.                                         }
48.                                     }
49.                                 }
50.                             }
51.         }
```

لاحظ ان الكود بدأ يتوسع و ايضاً يترتب، حسناً دعني اوضح لك آلية عمل هذا الكود، اولاً لاحظ انني استخدمت الطريقة الاولى لإطلاق النار، بالنسبة للذخيرة فلاحظ في السطرين الاول و الثاني انني عرفت متغيرين من نوع int بحيث ان الاول static لكي نستخدمه في ملف آخر لاحقاً هذا المتغير يمثل الذخيرة الاولى، المتغير الثاني باسم maxAmmo وهو يمثل اقصى عدد لجميع الذخائر (جميع الذخائر). في السطر 4 عرفت enum يحتوي على 3 انواع للطلقات و هي واضحة، هنا يجب تخزين هذا الـ enum في متغير لكي نستطيع التعامل مع قيمة، و كما تلاحظ في السطر اسفلة خزنت قيم المتغير في متغير جديد باسم typeBullet اذن الى هنا اتممنا اول خطوة و هي تعريف المتغيرات اما بالنسبة لذخائر باقي الطلقات فهي ستكون عبارة عن متغير جديد يمثل الذخيرة لكل طلقة من Ammo_1 و باقي الأكواد عبارة عن استنساخ.

في Start قمت بتعريف اول قيم و هي وضع الذخيرة في اعلى قيمة و ايضاً تحديد اول قيمة افتراضية لنوع الذخيرة و هي الذخيرة الاولى للاعب و طبيعي ان يبدأ بها و سيتمكن من تغييرها حال اللعب، في الدالة PlayerShoot كل ماقمت بفعلة هو وضع قيمة الذخيرة الدنيا و العليا و القيمة الحالية لها و هذا ماتوفرة لنا الدالة Clamp من الفئة Mathf و كما تلاحظ في السطر 21 قمت بوضع هذه القيم في المتغير Ammo_1 و سيتم تطبيق العملية في الدالة بشكل كامل، في الشرط قمت بالتحقق من حالة الطلقة اي نوع الطلقة التي يستخدمها اللاعب الان و في حالة كانت الطلقة الاولى ستكون قيمة maxAmmo = 60 ايضاً سيتم اعادة قيمة الوقت المتبقي للطلقة نظراً لنوع الطلقة، شرط آخر يجب التحقق من قيمة Ammo_1 لكي يتيح للاعب اطلاق النار في حالة كانت هناك طلقات، الان جاء شرط الوقت و سيتم التحقق من قيمة الوقت المتبقي لكي تتم عملية الضغط على زر الاطلاق ففي حالة تحقق الشرط سيتم تمكينك من اطلاق النار و في حالة اطلقت النار سيتم انقاص الذخيرة كما هو موضح في السطر 43، اذن لاحظ ان الكود سهل و غير متشعب فكل ماعليك فعلة هو التأكد من مناداة الدالة في Update و سيتم تشغيل الدالة و التأكد من قيمها عند اللعب.

انواع الطلقات:

هذه الفقرة ستختص بمناقشة طريقة التبديل بين الطلقات و سنتحقق من حالة الطلقة و تطبيق قيمها. هنا يجب تحديد enum جديد يحتوي على انواع الطلقات المتاحة في المستوى و يتم ربطهم في الطلقات، اجد ان هذه الطريقة ستسهل علينا الامور و لكنها تتيح لك اختيار الطلقات المتاحة في المستوى يدوياً مثلاً احد الانواع هو B1_B2 سيتيح لك استخدام الطلقة الاولى و الثانية، و شيء آخر يجب ان نقوم برمجة جميع الطلقات الان، الامر ليس معقد بس هو عبارة عن استنساخ للكود السابق لكن سأترك هذه المسألة لأخر الفقرة لأننا ايضاً سنناقش طريقة التعامل مع ذخيرة كل طلقة. الان لاحظ هذا الكود الذي يتيح لك تحديد نوع الطلقات في المستوى:

```
1.      public enum UseBullets{
2.          Bullet_1,
3.          B1_B2,
4.          AllBullets
5.      }
6.      public enum TypeBullet{
7.          Bullet_1,
8.          Bullet_2,
9.          Bullet_3
10.     }
```

```
11.	public TypeBullet typeBullet;
12.	public UseBullets useBullets;
13.
14.	void ChangingBullets()
15.	{
16.		if(useBullets == UseBullets.Bullet_1)
17.		{
18.			if(Input.GetKeyDown("1"))
19.			{
20.				typeBullet = TypeBullet.Bullet_1;
21.			}
22.		}
23.		if(useBullets == UseBullets.B1_B2)
24.		{
25.			if(Input.GetKeyDown("1"))
26.			{
27.				typeBullet = TypeBullet.Bullet_1;
28.			}
29.			if(Input.GetKeyDown("2"))
30.			{
31.				typeBullet = TypeBullet.Bullet_2;
32.			}
33.		}
34.		if(useBullets == UseBullets.AllBullets)
35.		{
36.			if(Input.GetKeyDown("1"))
37.			{
38.				typeBullet = TypeBullet.Bullet_1;
39.			}
40.			if(Input.GetKeyDown("2"))
41.			{
42.				typeBullet = TypeBullet.Bullet_2;
43.			}
44.			if(Input.GetKeyDown("3"))
45.			{
46.				typeBullet = TypeBullet.Bullet_3;
47.			}
48.		}
49.	}
```

عند التأمل في الكود ترى انه عبارة عن شروط و في حالة تحققت هذه الشروط سيتم وضع شروط اخرى قابلة للتغيير! لاحظ في البداية انني عرفت enum باسم UseBullets ووظيفته هي اتاحة الطلقات للاستخدام و تجد انه يحتوي على 3 نواع بحيث ان الاول باسم Bullet_1 و هذا يحدد انك ستستخدم الطلقة الاولى فقط في المستوى، اما النوع الثاني وهو B1_B2 و هو يتيح لك استخدام الطلقة الاولى و الثانية، اما الاخير AllBullets سيتيح لك استخدام جميع الطلقات، الان لاحظ في السطر 12 عرفت متغير يمثل الفئة UseBullets و هذا المتغير يتم التعامل معه يدوياً اي يدوياً يتم تحديد نوع الطلقات في هذا المستوى و هذا الشيء سيساعدك على تحديد نوع الطلقات في المستوى الذي تريده.

الان بالذهاب الى الدالة ChangingBullets ستجد انها عبارة عن شروط و هذه الدالة تعمل بشكل مباشر في Update بحيث تتيح لك ضغط الازرار و لكن يجب التحقق من حالة المتغير useBullets و الذي يحقق من حال الطلقة، اول شرط تلاحظ انه يجب التحقق من نوع المتغير على انه النوع Bullet_1 لكي يتيح لك استخدام الطلقة الاولى و يمكنك استخدامها عبر الضغط على الرقم واحد، بالنسبة للشرط الثاني فهو يطلب التحقق من ان حالة المتغير useBullets هي النوع B1_B2 لكي يتيح لك استخدام الطلقتان الاولى و الثانية و يمكنك

التغيير بينهم عبر الضغط على الرقم 1 او 2، اما بالنسبة للشرط الخير فهو يتحقق من اتاحة جميع الطلقات و في حالة تم التحقق من الشرط فبشكل مباشر سيتم تطبيقه و سيتيح لك التغيير بين الطلقات عبر اضغط على 1 او 2 او 3.

ماذا الان؟ اليس فعلاً سهل، ان كل ماتحتوية هذه الدالة هو شروط تتحقق من نوع الطلقات المتاحة في المستوى المطلوب و شيء آخر لاتضع قيمة افتراضي لهذا المتغير لأنه سيتم تطبيقات على جميع المستويات حال بدأ اللعب لهذا دع وضع القيم الافتراضية يدوياً في كل مرحلة، فكل ماستفعلة في النهاية هو تغيير حالة المتغير في كل مستوى.

الان يجب علينا تحديد الذخيرة لكل طلقة و تعريف متغيرات تمثل الطلقات اي لكل طلقة متغير من نوع Rigidbody2D، ولاحظ انني قمت بجمع بقايا الفقرتين الاولى و الثاني التي تهتم بأطلاق النار هنا، اذن في البداية يجب تعريف المتغيرات التالية:

```
1.          [Header("Ammo Bullets")]
2.          public static int Ammo_1;
3.          public static int Ammo_2;
4.          public static int Ammo_3;
5.          public int maxAmmo = 60;
6.
7.
8.          [Header("Bullets")]
9.          public Rigidbody2D b_Bullet_1;
10.         public Rigidbody2D b_Bullet_2;
11.         public Rigidbody2D b_Bullet_3;
```

لاحظ في البداية عرفت 3 ذخائر و كل ذخيرة تمثل اما maxAmmo فهو متغير واحد سيتم ربطة بباقي الطلقات، الان انظر الى الطلقات من نوع Rigidbody2D هي عبارة عن 3 انواع و كل طلقة تمثل كائن الطلقة لكي نعرف ماهي الطلقة التي يجب استخدامها و ربطها.

حسناً هذه القيم الاولية ايضاً علينا التعديل على الدالة PlayerShoot و يجب على كود نسخ الطلقات ان يكون في حالة تعيد بارامتر من نوع Rigidbody2D يمثل الطلقة و آخر من نوع float يمثل سرعة الطلقة، و سنقوم بمناداة الدالة حال اطلاق النار و هذا سيوفر علينا نسخ الكود مراراً و تكراراً، اذن لنقم بوضع كود نسخ الطلقات في دالة لوحدة بحيث يصبح هذا الشكل:

```
1.  void Shoot(Rigidbody2D bullet, float speedShoot)
2.          {
3.                  timeShoot = timeleft;
4.                  Rigidbody2D b_prefab =
5.                  Instantiate(bullet, pointShoot.position, Quaternion.identity) as Rigidbody2D;
6.
7.                  b_prefab.velocity = speedShoot * transform.right;
8.                  b_prefab.transform.rotation = transform.rotation;
9.          }
```

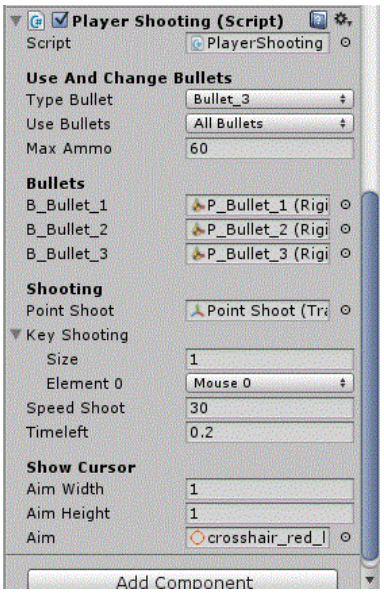
اذن لاحظ هذه دالة منفصلة لن يتم استدعائها الا في حالة تم التأكد من الضغط على زر الماوس الايسر، و ادا نظرت الى البارامترات ستجد ان هذه الدالة تطلب بارمتران بحيث ان الاول يمثل الطلقة و الثاني يمثل سرعة الطلقة، اما داخل الدالة فهو نفس الكود الذي استخدمناه سابقاً لنسخ الطلقات، لكن هنا ستلاحظ ان الطلقة هي البارامتر الذي يمثل الطلقة و السرعة هي البارامتر الذي يمثل السرعة لكي نستطيع تغييرهم لاحقاً عند تشغيل الدالة.

الان يجب علينا تنظيم كود اطلاق النار PlayerShoot و يجب وضع جميع الطلقات و التحقق من الطلقات و كما فلت سابقاً ان لدينا 3 طلقات و كل طلقة لها وظيفتها و الان عبر استخدام هذه الدالة سنقوم بتغير الطلقة و السرعة و ستظهر لنا الطلقة في المشهد و كل طلقة لها سرعتها ووظيفتها التي سنناقشها لاحقاً، شيء اخير يجب علينا التحقق من حالة المتغير maxAmmo في كل شرط TypeBullet لتحديد اعلى قيمة لكل طلقة، فمثلاً الطلقة الاخيرة يجب ان تكون اقصى ذخيرة لها هي 3 و لهذا يجب علينا التأكد من ان قيمة maxAmmo لهذه الطلقة هي 3، اذن لنقوم بترتيب الدالة PlayerShoot، لاحظ الكود الاخير لها:

```
1.  void PlayerShoot()
2.  {
3.      timeShoot -= Time.deltaTime;
4.
5.      Ammo_1 = Mathf.Clamp(Ammo_1 ,0, maxAmmo = 60);
6.      Ammo_2 = Mathf.Clamp(Ammo_2 ,0, maxAmmo = 90);
7.      Ammo_3 = Mathf.Clamp(Ammo_3, 0, maxAmmo = 3);
8.
9.      ////////////////////// Bullet1 ///////////////////////
10.     if(typeBullet == TypeBullet.Bullet_1)
11.     {
12.         maxAmmo = 60;
13.         timeleft = 0.2f;
14.
15.         if(Ammo_1 > 0)
16.         {
17.             if(timeShoot <= 0)
18.             {
19.                 foreach(KeyCode keyCode in keyShooting)
20.                 {
21.
22.                     if(Input.GetKey(keyCode))
23.                     {
24.                         Shoot(b_Bullet_1, SpeedShoot = 30);
25.                         Ammo_1 -= 1;
26.
27.                         break;
28.                     }
29.                 }
30.             }
31.         }
32.     }
33.
34.     ////////////////////// Bullet2 ///////////////////////
35.     if(typeBullet == TypeBullet.Bullet_2)
36.     {
37.         maxAmmo = 90;
38.         timeleft = 0.1f;
39.
40.         if(Ammo_2 > 0)
41.         {
42.             if(timeShoot <= 0)
43.             {
44.                 foreach(KeyCode keyCode in keyShooting)
45.                 {
46.
47.                     if(Input.GetKey(keyCode))
48.                     {
49.                         Shoot(b_Bullet_2, SpeedShoot = 30);
50.                         Ammo_2 -= 1;
51.
52.                         break;
53.                     }
54.                 }
55.             }
```

```
56.                     }
57.                 }
58.                 ////////////////////// Bullet3 //////////////////////
59.                 if(typeBullet == TypeBullet.Bullet_3)
60.                 {
61.                     maxAmmo = 3;
62.                     timeleft = 0.4f;
63.
64.                     if(Ammo_3 > 0)
65.                     {
66.                         if(timeShoot <= 0)
67.                         {
68.                             foreach(KeyCode keyCode in keyShooting)
69.                             {
70.
71.                                 if(Input.GetKeyDown(keyCode))
72.                                 {
73.                                     Shoot(b_Bullet_3, SpeedShoot = 3);
74.                                     Ammo_3 -= 1;
75.
76.                                     break;
77.                                 }
78.                             }
79.                         }
80.                     }
81.                 }
82.         }
```

قد يتبادر في ذهنك سؤال و هو لماذا الدالة كبيرة بهذا الشكل؟ صرحةً سأجيبك بكل شفايفة ان هذه الدالة عبارة عن اكواد مستنسخة و لاحظ انه قبل كل شرط هناك تعليق يدل على نوع الطلقة اما بالنسبة للحلقة foreach ستجد انني استخدمت الدالة Shoot و مررت البارامترات و بحيث اعطيت لكل شرط طلقته الخاصة ايضاً سرعتة و تلاحظ انني جعلت قيمة SpeedShoot في الاخير هي 3 نظراً لسرعة الطلقة الاخيرة و هي القنبلة، قد تتسائل لماذا حددت قيمة للمتغير SpeedShoot و لم اكتب القيمة تلقائياً اي كنت سأكتب 3 بدل SpeedShoot=3، السبب هو انني اريد ان التأكد من ان قيمة SpeedShoot تتغير فقط، اما في حالة وضعت قيمة للسرعة بشكل مباشر فهذه ليست مشكلة، الان تجد انني عرفت الشروط التي تخص كل طلقة بحيث ان السطر 9 يمثل الطلقة الاولى و السطر 34 يمثل الطلقة الثانية، اما السطر 58 يمثل الطلقة الثالثة و لو قمت بمراجعتهم لرأيت انهم نفس الأكواد و هي مستنسخة و التغييرات طفيفة. الان لاحظ في السطر 5, 6, 7 لقد قمت بتعريف اعلى و اقل قيمة لكل Ammo و كلها تخلف في اقصى عدد للطلقات، و بهذا الشكل نكون قد هيئنا الدالة PlayerShoot و اصبحت مرتبة.



في الدالة Start قم تحديد قيمة كل Ammo بحيث تساوي اقصى قيمة maxAmmo كما فعلنا في الطلقة الاولى، و الى هنا نكون قد انتهينا من عملية اطلاق النار و تجهيز الطلقات و اتاحة تغييرها، حالياً لن تكون بحاجة الى الملف البرمجي لتغيير الطلقات بل ستتمكن من تغييرها من قائمة الانسباكتور كما هو موضح.

برمجة الطلقات:

وصلنا للجانب الاسهل وهو وضع معلومات للطلقات عند التصادم، صراحةً ليست معلومات بكل معنى الكلمة اي بحيث ان الطلقة تأخذ المعلومات من الاعداء عند التصادم لا بل ستقوم الطلقة بتدمير نفسها حالياً و سنضع لها عمر لبقائها في المشهد، اذن مفهوم الطلقات سهل و كل مايجب عليك فعلة هو برمجة طلقة واحد و استنساخ الكود الى باقي الطلقات. اولاً قم بأنشاء 3 ملفات برمجية بحيث ان كل ملف برمجي يمثل الطلقة الذي سيوضع عليها كمكون، قم بفتح الملف البرمجي و اضف هذا الكود:

```
1.  public float timeleft;
2.
3.  void Start()
4.  {
5.          Destroy(gameObject, timeleft);
6.  }
7.
8.  void OnCollisionEnter2D(Collision2D Col)
9.  {
10.         if(Col.collider.gameObject)
11.         {
12.                 Destroy(gameObject);
13.         }
14. }
```

اذن هذا الكود سيمثل الطلقتان الاولى و الثانية ،لاحظ في البداية عرفت متغير من نوع float يمثل عمر الطلقات في المشهد، في Start سيتم تفجير الطلقة اعتماداً على الدالة Destroy و لاحظ انني اضفت وقت تدمير الطلقات و هذه الدالة كما تعرف تتيح بارامتر اختياري من نوع float يمثل عمر الكائن قبل التدمير، اخيراً استخدمت الدالة OnCollisionEnter2D و سيتم تدمير الطلقة في حالة اصطدمت في اي كائن يحتوي على Collider، اذن بهذا الشكل نكون قد عرفنا او برمجنا شكل ذكاء الطلقة الاولي.

الان قم بوضع هذا الكود في ملف الطلقة الاولى و الثانية فقط، اما بالنسبة للطلقة الثالثة فهي قنبلة و يجب على هذه القنبلة عمل نطاق محدد عندما يدخل فيه اي عدو يتأثر ضرره او يموت بشكل مباشر نظراً لأنها قنبلة و بالحديث عن هذه المسألة فأننا سنناقش طريقتان لوضع مسافة و تفجير الاعداء و كلتا الطريقتان تشبهان بعض، اذن قم بفتح ملف الطلقة الثالثة.

- الطريقة الاولى :حسناً بالحديث عن الطريقة الاولى و التي سنناقش فيها استخدام الاشعة في عمل نطاق معين بحيث ان هذا الشعاع يعيد لنا معلومات التصادم مع الكائنات في المشهد و في هذه الطريقة سنستخدم الفئة RaycastHit2D لنستطيع الحصول على معلومات التصادم مع الكولايدرات في المشهد و من خلال هذا التوضيح البسيط يتضح ان هذا الكود هو كود فيزيائي و سنستخدمه في الدالة FixedUpdate، اضف الكود التالي:

```
1.	public float timeleft;
2.	public LayerMask enemiesLayer;
3.
4.	void FixedUpdate()
5.	{
6.		RaycastHit2D[]  hits =
7.		Physics2D.CircleCastAll (transform.position, 5f, transform.position , 0f, enemiesLayer);
8.		foreach(RaycastHit2D hit in hits)
9.		{
10.			if(hit.collider.gameObject.CompareTag("EnemyOne"))
11.			{
12.				Destroy(gameObject);
13.			}
14.		}
15.	}
```

حسناً لاحظ في البداية انني متغيرين الاول من نوع float يمثل الوقت المتبقي لتفجير القنبلة، اما المتغير الثاني فهو من نوع LayerMask و قمت بتعريفة لتحقق من ان القنبلة تلامست مع Layer الاعداء، الان في الدالة FixedUpdate عرفت مصفوفة RaycastHit2D بحيث ان هذه الفئة ستقوم بتحديد جميع الكائنات التي تلامست معها وقمت بتسميتها hits، الان هذا المتغير يخزن قيم الدالة CircleCastAll من الفئة Phsics2D و هي خاصة بالدوال الفيزيائية، ان هذه الدالة تعيد 5 بارامترات بحيث ان الاول يطلب منك اصل الشعاع Origin و الثاني قطر الشعاع و كما تلاحظ ان القطر عندنا هو 5 و الثالث يحدد الاتجاه ونحن نحتاج الى هذا الشعاع في نقطة الاصل للقنبلة، بالنسبة للمتغير الرابع هو يطلب المسافة و لاحظ انني وضعتها صفر فلا حاجة لوضع مسافة للدائرة الشعاعية و لهذا سيتم وضع الدائرة في منتصف القنبلة، اما المتغير الاخير فهو يطلب Layer معين للتأكد من ان الشعاع تلامس مع هذا الـ Layer و ستجد انني وضعت اللاير الذي عرفته، اذن بهذا الشكل نكون قد هيئنها الدائرة الشعاعية لكي تظهر، الان يجب علينا وضع حلقة تتحقق من تلامس الشعاع مع الكائنات المحددة باللاير و هذا ماقمت به في السطر 8 استخدمت الحلقة التكرارية foreach و من خلال فئة باسم RaycastHit2D سيتم تخزين جميع قيم المصفوفة hits في المتغير hit و الان يجب على المتغير التأكد من تلامسه مع الاعداء عبر مقارنا التاج الذي في الكود مع التاج الذي يحمله العدو، وكما هو موضح في الشرط قمت بالتأكد من ان التاج هو EnemyOne و سيتم تدمير القنبلة في حالة تحقق الشرط.

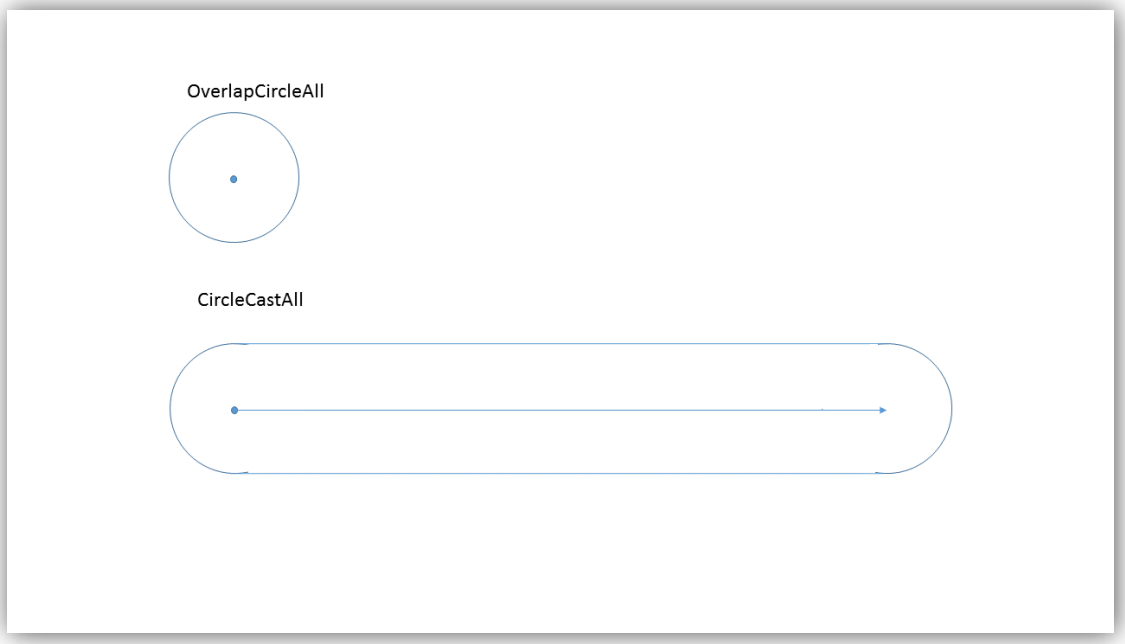
- الطريقة الثانية: بالنسبة لهذه الطريقة فهنا سنستخدم الفئة Collider2D و سنضعها على شكل مصفوفة تخزن جميع التلامسات و سنستخدم الفئة Physics2D و لكن هذه المرة لن نستخدم الاشعة بل سنستخدم الدالة OverlapCircleAll ووظيفتها انها تقوم برسم دائرة و تتحقق من عدة شروط تضعها و لا يختلف عملها عن عمل الاشعة و لكن افضلها حالياً لان الاشعة مكلفة برمجياً و القنبلة في النهاية لن تبقى فترة طويلة في المشهد لهذا سيتم تدميها و سيتوجب علينا رسم دائرة لتحقق من تلامسها مع الاعداء، لاحظ هذا الكود:

```
1.	void FixedUpdate()
2.	{
3.		timeleft -= Time.deltaTime;
4.
5.		if(timeleft <= 0)
6.		{
7.			Collider2D[] col = Physics2D.OverlapCircleAll(transform.position, 5f, enemiesLayer);
8.			foreach(Collider2D hit in col)
9.			{
10.				if(hit.gameObject.CompareTag("EnemyOne"))
11.				{
12.					Destroy(gameObject);
13.				}
14.			}
15.		}
16.	}
```

حسناً دعنا نناقش هذا الكود، قد تلاحظ ان هذا الكود شبيه بالكود السابق اما في حالة لم تلاحظ تابع هذا الشرح، لاحظ في السطر 7 استخدمت الفئة Collider2D و هي على شكل مصفوفة كما في السابق و على هذه الفئة ان تعيد جميع التصادمات و هذه الفئة او المتغير col الذي يمثل هذه المصفوفة سيتم خزن قيم الدالة OverlapCircleAll بحيث ان هذه الدالة تطلب 3 بارامترات فقط و هي موقع الدائرة، قطرها و الـ Layer الذي ستتلامس معه، و كما تلاحظ ان القيم هي نفسها لم تتغير، في السطر 8 استخدمت الحلقة foreach و يجب ان تخزن معلومات المصفوفة في المتغير hit الذي يمثل الفئة Collider2D الموجودة في الحلقة، الان سيتم التأكد من ان التاج الذي تلامست معه الدائرة هو نفسة الموجود في الاعداء و في حالة تحقق سيتم تدمير القنبلة.

اليس الامر سهلً؟ ان محور الموضوع يدور حول وضع دائرة تتحقق من تلامسها مع الاعداء لكي نستطيع جلب معلومات التصادم لاحقاً عند برمجة الاعداء وسيتم وضع الضرر لكل عدو عند انفجار القنبلة ووقوعه داخل الدائر، في النهاية لديك الطريقتان و يمكنك استخدامهما.

تأمل هذا الرسم الذي يوضح الاشياء التي توفرها الدالتان CircleCastAll و OverlapCircleAll :



لاحظ الدائرة الاولى وهي تمثل الدائرة المرسومة عبر الدالة OverlapCircleAll و هي تحدد موقع الدائرة و كما تلاحظ مركزها ثابت و قطرها، اما بالنسبة للدائرة الثانية فهي المرسومة عبر الدالة CircleCastAll ولاحظ مركز الدائرة وقطرها الذي لا يختلف عن سابقتها ولكن اتجاه نصف القطر الثاني الى الجهة اليمين و هذا البارامتر Direction الذي توفر الدالة و هو يحدد اتجاه لهذه الدائرة و كما هو موضح و هو يعتمد على قيمة البارامتر Distance و يمكنك وضع المسافة صفر و الاتجاه هو موقع القنبلة و سيتم وضع الدائرة كما في الشكل الاول، و لكن كما تلاحظ ان هناك اختلافات بين الدوال و استخدم الدوال التي تحتاجها و التي توفر البارامترات المناسبة لوضع الشيء الذي تريده في المشهد بكل دقة.

الصحة و الضرر:

هذه الفقرة مهم لأننا سنقوم ببرمجة الصحة و الضرر للاعب لكي نهيئ استخدامهم لاحقاً، الان في ملف PlayerController سنعرف دالتين بحيث ان الاولى ApplyHealth و هي تتحقق من صحة اللاعب و موتة، اما الثانية ApplyDamage وهي تتحقق من ضرر الاعب و هذه الدالة ستكون public int لكي نستطيع استخدامها لاحقاً عبر فئتها، اذن اولاً قم بتعريف هذان المتغيرات اللذان يمثلان الصحة و اقصى حد لها :

```
[Header("Health")]
public static int Health;
public int maxHealth = 100;
```

في Start اجعل الصحة تساوي اقصى قيمة للصحة maxHealth. الان قم باضافة الدالتين :

```
void ApplyHealth()
{
        if(Health <= 0)
        {
                Destroy(gameObject);
        }
}


public static int ApplyDamage(int damage)
{
        Health -= damage;
        return damage;
}
```

حسناً، لاحظ الدالة الاولى و هي تمثل الصحة و تتأكد من ان قيمة الصحة اصغر او تساوي الصفر لكي تقوم بتدمير اللاعب، اما الدالة الثانية فهي من نوع int اي تحتوي على بارامتر من نوع int و هذا البارامتر يمثل الضرر و لاحظ ان الدالة static لكي نستطيع استخدامها عبر فئتها مباشرة و لان اللاعب فقط من تمثله و ليس هناك كائن آخر مربوط بالضرر و في النهاية هذه الدالة ستعيد لنا البارامتر damage و الذي ستعتمد عليه الصحة في انقاصها، الى هنا و قم باضافة الدالة ApplyHealth في Update الى ان نستخدم الدالة ApplyDamage لاحقاً.

موت اللاعب:

سيترتب على موت اللاعب العديد من الاشياء مثل توقف الحركة و اطلاق النار ايضاً تغيير لون المدفع و العجلة و هنا صور توضح تأثر الشكل في حالة الانفجار و يمكنك ان تجد الصورتان و التان تحملان اسم Destroy للمدفع و العجلة في الملف The Cannon و سنقوم باستخدامهما و هنا سنتعرف على العديد من الاشياء مثل المكون SpriteRenderer و استخدام الفئة التي تمثله ايضاً Sprite لكي نقوم بتغيير صورة اللاعب حال موتة و سنقوم بتعريف بعض المتغيرات التي تتيح للاعب استخدام وظائفه.

بالنسبة لتدمير اللاعب سنقوم بالتعديل على بعض الاشياء مثل الدالة ApplyHealth و علينا وضع صورة التأثير في حالة موت اللاعب، اولاً قم بتعريف هذه المتغير:

```
public Sprite spriteDeath;
private SpriteRenderer Sprite_r;
Private static bool ApplyMove = true;
```

حسناً، لاحظ المتغير الاول وهو من نوع Sprite وسيمثل صورة العدو عند الموت، اما الثاني فهو للحصول على المكون SpriteRenderer و استخدام المتغير Sprite لتغيير شكلة، بينما الاخير و الذي سيتم ربطة في جميع الدوال اعتماداً على صحة اللاعب ففي حالة كان اللاعب لازال موجود في المشهد ستكون قيمة المتغير false و في حالة كان غير موجود او ميت يتكون القيمة true و بدورها سيتوقف كل شيء في اللاعب من حركة و دوران للمدفع، اذن سنقوم بالتعديل اولاً اجعل المتغير Sprite_r يحصل على المكون SpriteRenderer ، الان بالذهاب الى الدالة ApplyHealth قم باضافة الكود التالي:

```
        if(Health <= 0)
        {
```

```
3.                      Sprite_r.sprite = spriteDeath;
4.                      ApplyMove = false;
5.                      WheelController.MoveWheel = false;
6.                      PlayerShooting.ApplyShooting = false;
7.                      rb2D.isKinematic = true;
8.                      Health = 0;
9.              }
```

اذن لاحظ ان في حالة مات اللاعب سيترتب علية العديد من الاشياء و هي تغيير صورة المدفع كما هو موضح في السطر 3 بحيث انني استخدمت المتغير Sprite، بالنسبة للسطر 4 قمت بوضع قيمة المتغير false و سيتم توقيف كل شيء في حالة مات اللاعب و لكن ليس الان لأننا لم نربطه بعد، بالنسبة للسطر 5 تلاحظ انني استخدمت المتغير MoveWheel و قد عرفناه سابقاً وربطنا الحركة فيه لهذا سيتم الغاء تفعيلة في هذا الشرط، اما اطلاق النار سيتم عبر انشاء متغير في الملف PlayerShooting و بحيث ان هذا المتغير سيحدد لنا هل المدفع يستطيع اطلاق النار ام لا، بالنسبة للإعادة الشروط الى حالتها الطبيعية سيتم عبر الملف GameManagers عند انشائه لاحقاً.

الان علينا ربط المتغير ApplyMove في الدوال فكل ماستقوم بفعلة هو ربط الدوال بهذا المتغير...حظ شكل الشرط في الدوال:

```
1.      void Update(){
2.
3.              CannonPosition();
4.
5.              if(ApplyMove)
6.              {
7.                      ApplyHealth();
8.                      ForcePlayer();
9.              }
10.     }
11.
12.     void FixedUpdate()
13.     {
14.             if(ApplyMove)
15.             {
16.                     CannonRotation();
17.             }
18.     }
```

لاحظ ان ليس هناك من شيء صعب فكل ماقمت بفعلة هو ربط هذه الدوال بالمتغير ApplyMove و قد تلاحظ انني جنبت الدالة CannonPosition() و السبب انه في حالة تحقق الشرط سيتفصل المدفع عن العجلة و انا لا اريد عمل هذا لان شكل المدفع سيكون غريب بحيث ان عجلة في مكان و مدفع في مكان و هذه مشكلة فبكل بساطة قم بتجنيب الدالة و ستحل المشكلة.

اخيراً سيتبقى لنا الملف PlayerShooting و علينا التحكم بأطلاق النار ووضع الشروط في حالة مات اللاعب، اولاً قم باضافة هذا المتغير:

```
1.  private bool ApplyShooting = true;
```

اذن هذا المتغير سيحدد لنا هل اللاعب سيطلق النار ام لا بهذه البساطة فكل ماستقوم بفعلة هو التعديل على الدالة Update بحيث تصبح بهذا الشكل:

```
1.      void Update()
2.      {
3.              if(ApplyShooting)
4.              {
5.                      ShowCursorInLoad();
6.                      PlayerShoot();
7.                      ChangingBullets();
8.              }
```

```
9.          }
```

كما تلاحظ ان المتغير ApplyShooting يقوم بربط جميع الدوال و لن تعمل الدوال الا في حالة تم التحقق من ان قيمة المتغير هو true و بهذا الشكل نكون قد انهينا العمل على ربط الدوال كبداية اوليه الى ان نقوم بإعادتها الى قيمها الاولية لأنها static.

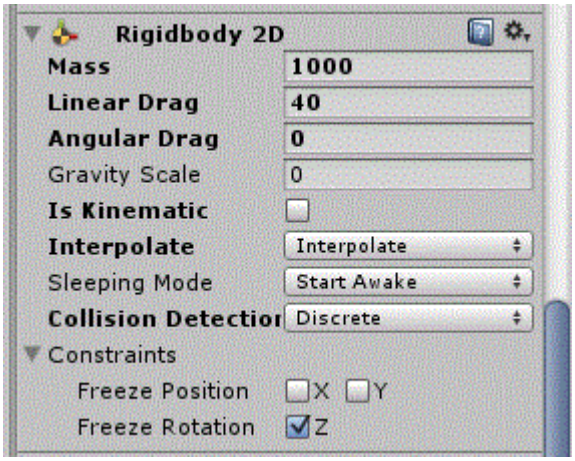
برمجة و تعديل الاعداء (Enemies)

الاعداء، انا مسألة الاعداء مهمة حالياً و من خلالها نستطيع اضافة المؤثرات المرئية و الصوتية و اختبار الذكاء الاصطناعي للأعداء وهو بسيط وفكرة هي ان الاعداء يتجهوا الى اللاعب بحيث ان اللاعب يعمل مسافة مع الاعداء و يحد من وصولهم اليه، شيء آخر على الاعداء ان يتشاركوا الصفات اي سنستخدم ملف برمجي للوراثة فقط باسم EnemiesInfo وهو سيحتوي على العديد من المعلومات و اذكر اهمها و هي (الحركة، الاسلحة، الطلقات، الصحة و الضرر) و جميعها سنضعها في ملف واحد و سيتم وراثتها و بشكل مباشر الى باقي الملفات ستكون عملية طبع للدوال و اعادة قيم، هناك العديد من الطرق لبرمجة الاعداء و لكن في هذه الحالة و بما ان لذينا العديد من الاعداء و هم يتشاركون بعض الصفات سيتم عمل ملف للوراثة منة و سيختصر علينا الامور.

من جانب آخر علينا تعديل طلقات الأعداء و ايضاً التعديل على الاعداء نفسهم و سنضيف العديد من الاشياء لهم، الى هنا لم نلمس الاعداء و لم نقم اضافة شيء فيهم لهذا سيتم التعامل معهم الان، هذه الفقرة ستتقسم الى عدة فقرات متفرعة سنناقش فيها فكرة الاعداء و آلية عملهم و كيف سنقوم بتحريكهم و جعلهم يتشاركون الصفات فيما بينهم، الفقرة القدامة ستختصر بتعديل الاعداء و الطلقات، اما الجانب البرمجي سنخصص له فقرات عديدة و سنناقشه بالتفصيل.

التعديل على الاعداء و الطلقات:

بالنسبة للأعداء فأننا قمنا بتعديلهم سابقاً لهذ سنقوم بتعديل مكون الـ Rigidbody2D فقط، اما بالنسبة للطلقات فهنا لم نناقش فكرة الطلقات مع العلم ان مفهومها سهل و ربما نتحدث عن الطريقة المستخدمة في لطلقات اللاعب، عموماً بالحديث عن الاعداء و احتكاكهم بالهواء اي الاحتكاك بشكل عام Linear Drag سنضيف شيء آخر وهو وضع ثقل يساعد العدو على البقاء في مكانة دون التأثر بطلقات اللاعب، اذن بالعودة الى مكون Rigidbody2D الخاص بالأعداء تأكد من وضع قيمة كما هو موضح:

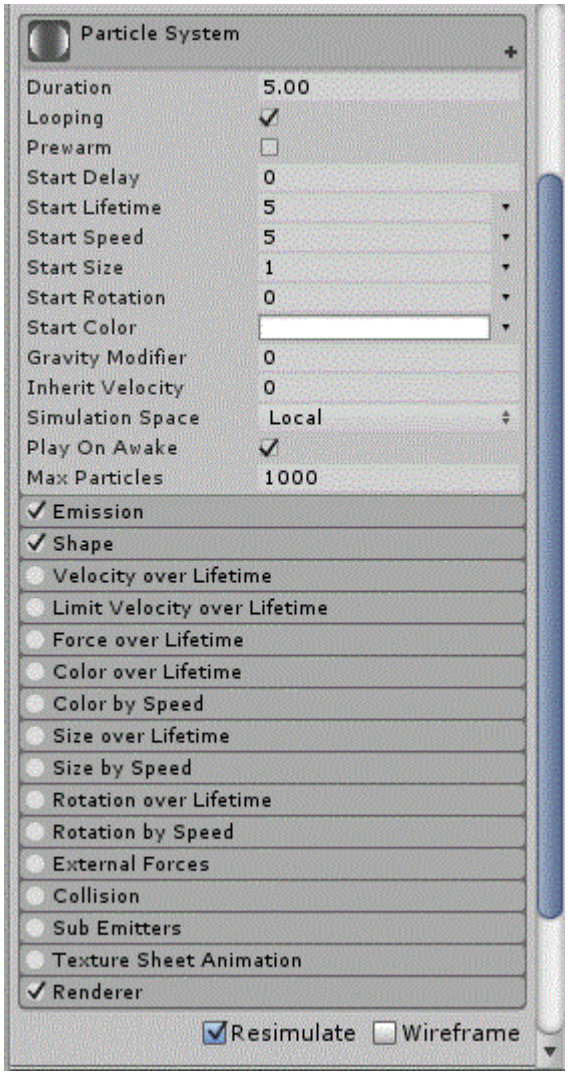


اذن الوزن هو 1000 و الاحتكاك الخطي هو 40 في العدو الاول اي لم نمس الاحتكاك فكل ماقمت بفعلة هو وضع وزن للعدو فقط، الان قم بتطبيق نفس الفكرة على جميع الاعداء، لكي ننتقل الى تعديل الطلقات.

بالنسبة للطلقات فسنتحدث عنها بشكل سريع، اما لعدو الاول فطلقاته ستكون شبيهه بطلقات للاعب و في ملف Bullets ستجد العديد من الطلقات و انا شخصياً اخترت الصورة bulletGreenSilver_outline، اولاً يجب ان تعرف ان المدفع سيقاتل مركبات فضائية و لهذا على الطلقة ان تكون طلقة ليزر او شبيهه بطلقة الليزر، ليست مشكلة فالبارتكل سيستم يحل المشكلة بسهوله و ان اردت يمكنك عمل تأثير لطلقة اللاعب باستخدام الـ Particle System و اقصد بالتأثير هو شعاع خلفي او شيء من هذا القبيل، دعنا نعود الى طلقة العدو الاول، قم بسحبها الى المشهد و اضف اليها مكون Rigidbody2D و Box Collider2D ايضاً قم بعمل Layer لها باسم BulletEnemies و هذا الـ Layer سيتمثل كل الطلقات ايضاً قم بعمل tag لهذه الطلقة و يفضل ان يكون b_Enemy_1 اي طلقة العدو الاول وكل عدو ستضيف له في النهاية الرقم الذي يمثله.

الان علينا وضع تأثير يخرج من الطلقة عند تحركها في الهواء و هنا الـ Particle System يوفر لنا خاصية الانبعاث و لها نوعان وهما Time و Distance.

الان قبل الحديث عنهما دعنا نضف Particle System الى المشهد و اجعله ابن لطلقة العدو الاول، قم بضبط موقع الـ Particle System و دورانه بحيث ترى الانبعاث الى الاسفل، الان اضغط على الـ Particle System و ستظهر لك هذه النافذة:

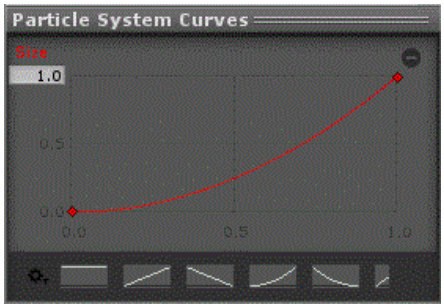


في هذه النافذة تستطيع ان ترى العديد من الخصائص فانا لن اناقشها كلها و لكن المهم سأقوم بتوضيحه، بالنسبة للخانة الاولى Duration فهذه الخانة تحدد المدة الزمنية بين كل موجة تأثير و اخرى و ترى القيمة الافتراضية 5 ثواني للموجة الواحدة و اتركها كما هي، الخيار الثاني Looping و هو يتيح لك عمل تكرار لكل موجة انبعاث او ايقاف التكرار و يمكنك اختباراها عبر تفعيلة و الغاءه، الخيار Prewarm يقوم بعمل دورة كامل للتأثير و هذا ايضاً يخص الخيار Start Delay و الذي يأخر عمل التأثير الى ثواني محددة، الخيارة Start Lifetime يقوم بإطالة مدة بقاء البارتكلز الواحد في المشهد و افتراضيً هو 5 ثواني، الخيار اسفلة واضح وهو يقوم بتحديد سرعة معينة للبارتكلز الواحد، اما الخيار Start Size فهو يحدد حجم البارتكلز الواحد، الخيار Start Rotation يقوم بعمل دوران للبارتكلز، الخيار Start Color يتيح لك تحديد لون معين للبارتكلز، اما الخيار Gravity Velocity فهو يعمل سرعة و هي تمثل جاذبية لهذا التأثير اي في حالة كانت القيمة 1 فحال التأثير يكون كحال اي جسم يتأثر بالجاذبية و سيتم وضع جميع البارتكلز الى الاسفل اي باتجاه الجاذبية، بالنسبة للخيار Simulation space وهو يحدد لك اسلوب محاكاة فضاء هذه الجسيمات فحالياً ضعها على قيمتها الافتراضية و لاحقاً سنعدلها، Play On Awake يقوم بتشغيل البارتكلز عند تشغيل اللعبة، اما الخيار الاخير فهو يحدد اقصى عدد للبارتكلز.

لاحظ ان جميع هذه القيم ابتدائية لعملي البارتكلز و ستلاحظ عن بجانب كل خانة يوجد سهم و هو يتيح لك توسيع التحكم بهذه الخيارات و يمكنك تجربتها،

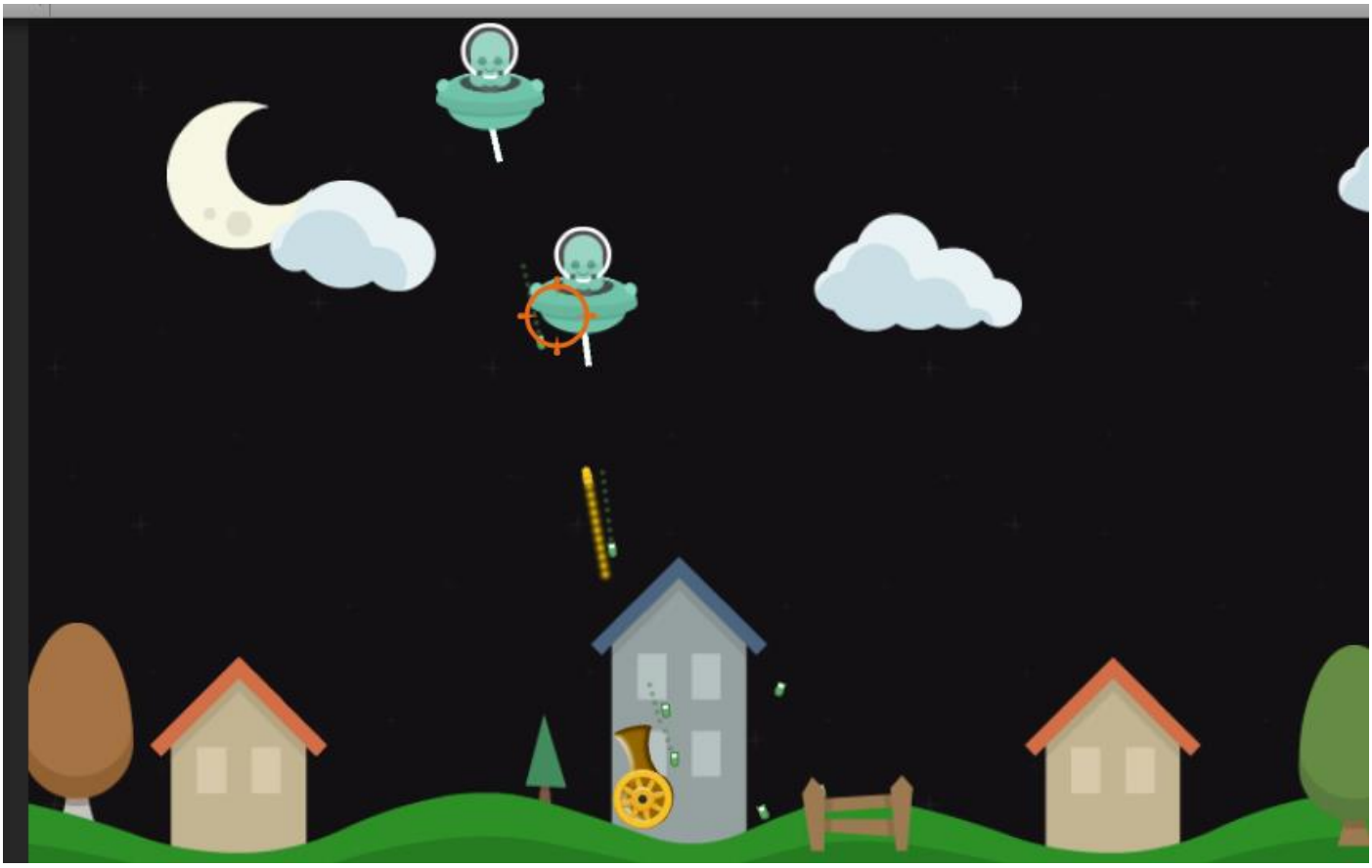
ان شرح القائمة بهذا الشكل ربما يصيبك بالملل لهذا دعنا ندخل الى صلب الموضوع وهو عمل تأثير للطلقة.

بالتعديل على هذا البارتكل، قم بوضع قيمة Start Lifetime = 0.1 ايضاً قيمة Start Speed = 1 اما الخيار Start Size = 0.1 بالنسبة للون قم بجعل اللون يميل الى لون الطلقة اما بتحديد قيمة نظرية او بأخذ اللون عبر مرشح الالوان او لاقط الالوان، بالنسبة للخيار Simulation space فأجعله World لان البارتكلز عبارة عن ابن للطلقة و وضعنا هذا الخيار الان لأننا سنقوم بجعل هذا البارتكلز يعمل عبر حساب المسافة بينه وبين الطلقة، بالانتقال الى القائمة Emission قم بتوسيعها و ستجدها على الحالة الافتراضية وقم بتغيير وضعها الى Distance ان وجدت مشكلة في هذه فاعدها الى القيمة الافتراضية لكي نستطيع ضبط باقي الخيارات جيداً، بالنسبة للقائمة Shape و ستجدها مفعله، بشكل مباشر قم بإلغاء تفعيها و ستجد ان البارتكلز تظهر على شكل خط عمودي، بشكل سريع انتقل الى القائمة Size by Speed و هذه القائمة تحدد حجم البارتكلز اعتماداً على السرعة، و عند فتحها ستجد خيار رسم بياني و يمكنك فتحة و سترى خط باللون الاحمر و عدة خطوط افتراضية، الان قم بوضعة كم هو موضح في الصورة.



ان هذا الرسم سيقوم بجعل هذه المجسمات تقل سرعة و حجم عند وصولها الى نهاية المطاف و لكي تطوف البارتكل في الهواء و يظهر تأثير واضح لها قم بتحديد هذا الرسم.

الان انتقل الى قائمة Renderer و هذه القائمة خاصة بوضع صور للبارتكلز و اشياء اخرى مثل تأثره بالظلال ووضع الصور على شكل كائنات و غيرها من الخصائص يمكنك تجربتها، حالياً قم بفتح هذه القائمة وستجد خانة باسم Material و هذه الخانة خاصة بوضع صور بدل البارتكلز الافتراضي في المشهد، هنا لا اريد استخدام صورة لعمل تأثير للطلقة و لكن عند الانتقال الى المؤثرات المرئية اعدكم بأننا سنناقش البارتكلز و العديد من الاشياء فيه، حسنا بالعودة الى المشهد ستجد الطلقة خالية من التأثيرات و السبب اننا وضعنا الانبعاث عند حساب المسافة Distance و لكن سترها تظهر عند الاطلاق، و هذه صورة اولية لكي تأخذ فكرة عن آلية عمل هذا التأثير:



حسناً بالنسبة طلقة العدو الثاني و هي القنابل ستجد الصورة Bullet Enemy3 مناسبة، لايهم الاسم و لكن ابحث عنها، في حال لم تعجبكم فبكل بساطة قم بتغييرها، اذن اضف اليها نفس مكونات الطلقة الاولى و قم بوضع تاج لها باسم b_Enemy_2 و هذا سيمثل الطلقة الثانية، حسناً سيتبقى لنا وضع التأثير لكن ساترك الامر لك و ان اردت يمكنك وضع تأثير بارتكاز او الغائها.

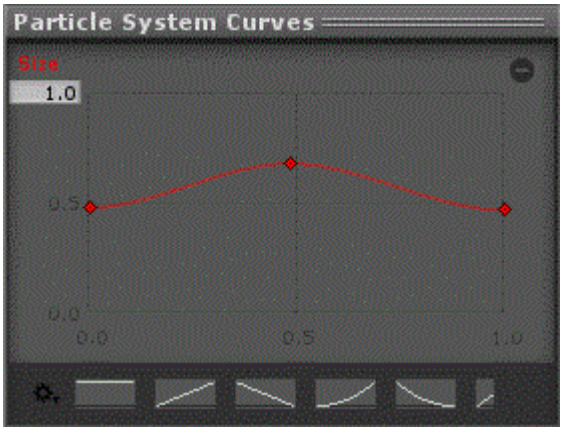
بالنسبة للطلقة الثالثة و هي الصاروخ و التي تختص بالعدو الثالث حامل الصواريخ و يجب علينا العثور اولاً على صورة الصاروخ و ستجد صورة للصاروخ باسم Bullet Enemy2 اسحبه ال المشهد و اضف نفس المكونات و تأكد من ان الـ Tag فيه باسم b_Enemy_3، الان الصاروخ يجب عمل مؤثر مرئي له لهذا اضف Particle System و قم بضبط إعداداته كالاتي:

Start Lifetime = 1
Start Speed = 1
Start Size = 1

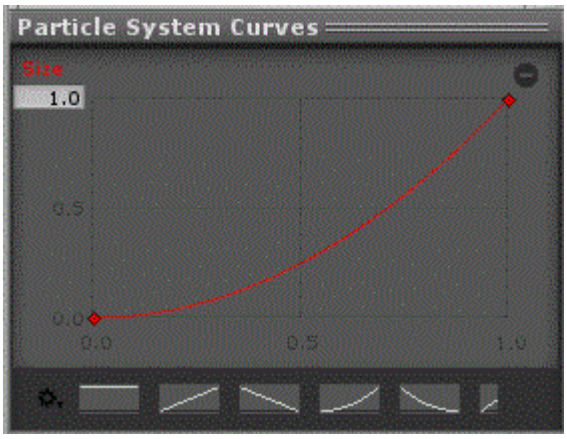
بالنسبة للون قم بجعلة يتناسب مع لون الصاروخ.

Simulation Space = local

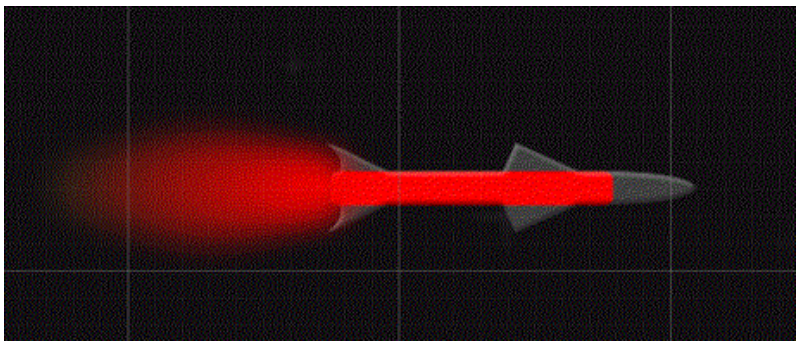
بالنسبة للقائمة Size Over Lifetime قم بضبط اعدادات المخطط البياني كما هو موضح:



اما بالنسبة للقائمة Size By Speed اضبط اعدادات المخطط البياني كما هو موضح:



حسناً الان بالعودة الى المشروع سترى ان البارتكلز يظهر بهذا الشكل:



اذن و بهذا الشكل نكون قد انتهينا من تجهيز الطلقات و ضبط اعدادات الاعداء، الان قم بجعل كل طلقة prefab لكي نستطيع استنساخها لاحقاً، سيتبقى لنا الجانب البرمجي وهو الجانب الذي سنناقشه في الفقرة القادمة، لهذا تأكد من ضبط اعدادات الطلقات ووضع المؤثرات الاولي و سيتم وضع باقي المؤثرات في فقرة تشملها.

انشاء الـ EnemiesInfo:

في هذه الفقرة سنتعرف على ملف الوراثة الخاص بالأعداء و سيحتوي هذا الملف على جميع خصائص الاعداء من اطلاق نار، صحة، اضرار و غيرها من الامور التي تخص الاعداء، ايضاً سيتم تعريف انواع الاعداء فيه و كل نوع من الاعداء سوف يحمل خصائص معينة يرثها من هذا الملف بكل سهوله.

ان هذه العملية ستساعدنا على التقليل من كتابة الأكواد المتكررة في اللعبة، استخدامي للوراثة في هذه اللعبة هو للتقليل من حدة استخدام الأكواد المتكررة و لكي استطيع ان ارى سير العمل على افضل حال، في حالة رأيت ان مشكلة الوراثة معقدة فبكل بساطة قم بنقل الأكواد الى ملف آخر و اللصقة في الاعداء.

حسناً دعنا نناقش ماهي الوظائف التي سيتوجب على الاعداء تشاركها؟ اولاً علي تذكيرك ان لدينا 3 انواع من الاعداء و لكل عدو وظيفته الخاصة و هناك اشياء عديدة سيتشاركها الاعداء مثل الحركة، الصحة و الضرر و سيتم تغيير قيمة الضرر من عدو لآخر و المسافة العشوائية و اطلاق النار و سيتم تغيير شكل الطلقات و سرعتها اعتماداً على دالة تقوم بذلك، بالنسبة للأشياء التي لن يتشاركها الاعداء و هي الاسلحة و للعلم ان العدو الاول له سلاح تستطيع ان تراه ووظيفه هذا السلاح هو النظر الى اللاعب لكي يطلق النار و سنناقش بعض الطرق لجعل هذا السلاح يطلق النار، بالنسبة للعدو الثاني فهو يطلق القنابل وبشكل مباشر سيتم اخراج القنابل من أطرافه، و اخيراً العدو الثالث سيطلق الصواريخ و على الصاروخ تتبع الحذف و سيتوجب علينا عمل ملف آخر للطلقات.

اولاً يجب علينا انشاء الملف EnemiesInfo و هذا الملف سيحتوي فقط على معلومات الاعداء، اذن في مجلد Scripts قم بأنشاء ملف جديد باسم EnemiesInfo و انشاء 4 ملفات اخرى تمثل الاعداء بحيث ان لكل عدو ملف و اجعل هذه الملفات ترث من الملف EnemiesInfo و لا تنسى اضافة abstract للملف EnemiesInfo، اذن هذا هو شكل الملف الرئيسي الاولي:

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public abstract class EnemiesInfo : MonoBehaviour {
5.
6.          [Header("Shooting")]
7.          protected float timeShoot = 0f;
8.          [SerializeField] protected float timeleft = 0.2f;
9.          [SerializeField] protected float SpeedShoot;
10.
11.         [Header("Health")]
12.         protected int health;
13.         [SerializeField] protected int maxHealth = 100;
14.
15.         [Header("AI Enemy")]
16.         [SerializeField] protected float speed;
17.         [SerializeField] protected float maxDistance;
18.         [SerializeField] protected float maxPosY;
19.         protected Rigidbody2D rb2D;
20.         protected Transform Player;
21.
22.
23.
24.         public virtual void Start()
25.         {
26.                 rb2D = GetComponent<Rigidbody2D>();
27.                 Player = GameObject.FindGameObjectWithTag("Player").transform;
28.                 health = maxHealth;
29.
30.         }
31.
```

```
32.         public virtual void FixedUpdate()
33.         {
34.                 EnemyMovement();
35.                 ApplyHealth();
36.         }
37.
38.
39.
40.         void EnemyMovement()
41.         {
42.
43.                 if(transform.position.y <= maxPosY)
44.                 {
45.                         Vector2 maxPositionY = new Vector2(transform.position.x, maxPosY);
46.                         rb2D.MovePosition(rb2D.position = maxPositionY);
47.                 }
48.
49.                 if(Player != null)
50.                 {
51.                         if(Vector2.Distance(Player.position, transform.position) > maxDistance)
52.                         {
53.                                 Vector2 MoveToPlayer = (Player.position - transform.position );
54.                                 rb2D.MovePosition(rb2D.position + MoveToPlayer * speed * Time.deltaTime);
55.
56.                         }else{
57.                                 return;
58.                         }
59.
60.                 }
61.                 else{
62.                         Destroy(gameObject);
63.                 }
64.         }
65.
66.
67.
68.         public void EnemyShoot(Rigidbody2D bullet, float speedShoot,float timeleft,Transform pointShoot)
69.         {
70.                 timeShoot -= Time.deltaTime;
71.                 if(timeShoot <= 0)
72.                 {
73.                         timeShoot = timeleft;
74.                         Rigidbody2D b_prefab =
75.                         Instantiate(bullet, pointShoot.position ,pointShoot.rotation) as Rigidbody2D;
76.                         b_prefab.velocity = speedShoot * pointShoot.up;
77.
78.                 }
79.         }
80.
81.         public float RandomDistance(float max, float min)
82.         {
83.                 maxDistance = Random.Range(max, min);
84.                 return maxDistance;
85.         }
```

```
86.         void ApplyHealth()
87.         {
88.                 if(PlayerController.Health <= 0)
89.                         health = 0;
90.
91.                 if(health <= 0)
92.                 {
93.                         health = 0;
94.                         Destroy(gameObject);
95.                 }
96.         }
97.
98.         public int ApplyDamage(int damage)
99.         {
100.                health -= damage;
101.                return damage;
102.        }
103.
104.        public virtual void OnCollisionEnter2D(Collision2D Col)
105.        {
106.                if(Col.gameObject.tag == "b_Player_1"||
107.                  Col.gameObject.tag == "b_Player_2"||
108.                  Col.gameObject.tag == "b_Player_3")
109.                {
110.
111.                }
112.        }
113.}
```

حسناً دعنا نناقش هذا الملف الضخم، اولاً ان هذا الملف يجمع كل شيء يخص الاعداء من حركة و اطلاق النار و صحة و ضرر و يمكنك مراجعة الدوال المسماة التي تشرح نفسها، في قائمة Shooting لذي 3 متغيرات و لا تختلف عن سابقتها و هي متغير يمثل وقت الاطلاق و آخر يمثل الزمن المتبقي و اخيراً سرعة الطلقة و لاحظ ان آخر اثنين عملت لهم تسلسل ميداني SerializeField لنتمكن من عرض المتغيرات في الانسباكتور و تغيير قيمهم لاحقاً ايضاً لاحظ معرفات الوصول لديهم هي protected فهي تتيح للفئة الحالية و الفئة الوارثة الوصول الى هذا المتغير و التعامل معه بينا private لايسمح الا للفئة الحالية بالوصول اليه، في قائمة Health عرفت متغيرين الاول يمثل الصحة و الثاني يمثل اقصى ضرر للصحة، اما في قائمة AI Enemy هي مختصة بالحركة و الحصول على المكونات و الكائنات، لاحظ قمت بتعريف 5 متغيرات بحيث ان اول 3 متغيرات هم من نوع float وهم يمثلوا السرعة، المسافة بين الاعداء و اللاعب و قصى مسافة للأعداء على محور y سنناقشها لاحقاً، الان بالنسبة للمتغيرين الاخيرين وظيفة الاول هو الحصول على مكون Rigidobdy2D من الاعداء لنتمكن من تحريكم لاحقاً، اما الثاني للبحث عن اللاعب ووضع الشروط علية، الان و بهذا الشكل نكون قد انهينا انشاء المتغيرات، هي متغيرات اولية و سنقوم باضافة المزيد لاحقاً.

في Start سيتوجب علينا وضع القيم الافتراضية للمتغيرات المهم مثل الصحة و استخراج او الحصول على مكون Rigidbody2D و البحث عن اللاعب Player ولاحظ ان الدالة Start هي virtual و هذه العملة ضرورية ليتمكن اللاعب من الوصول الى و على كل virtual ان تكون public فهي وظائف عامة يتشاركها الملف ملف الوراثة و الملف الوارث منة، و من خلال هذا الشيء سيتوجب علينا وضع دوال التحديث على هذا الشكل يتم تشغيل الوظائف في جميع الملفات الوارثة.

بالنسبة للحركة ان الاعداء سيعتمدون على المكون Rigidbody2D للتحرك و لهذا قمت بتعريف المتغير rb2D و الذي يمثل هذا المكون و للعلم ان هذا المكون فيزيائي و يجب علينا تشغيله في دالة فيزيائية FixedUpdate و كما تلاحظ قمت بأنشاء هذه الدالة و فيها قم بمناداة دالتين بحيث ان الاولى تختص بالحركة و الثانية تختص بالصحة، بالنسبة للحركة لاحظ انني عرفت دالة باسم EnemyMovement و هي تمثل الحركة لجميع الاعداء، الان في الدالة ستجد شرط يتحقق من موقع الاعداء على محور الـ y فلن نسمح للأعداء بالوصول الى الارض

و لسمها و لهذا يجب علينا الحد من عمل ذلك، في الشرط سيتم الوصول الى محاور الكائن و سنتلاعب بقيم المحور y بينما قيم المحور x ستضل نفسها، الان هذه القيمة مخزنة في المتغير maxPositionY، اسفلة استخدمت الدالة MovePosition و هي تختص بتحريك الاجسام الفيزيائية بحيث انني اضفت موقع الكائن على محاورة العادية الى محاورة الفيزيائية الحالية. الشرط الثاني وهو يتحقق من وجود اللاعب في المشهد ففي حالة تحقق الشرط سيتم تطبيق الاتي: التحقق من اعلى مسافة بين الاعداء و اللاعب و في حالة تجاوزت الحد المطلوب سيتم وضع الاعداء في أماكنهم او لن يتم تغيير شيء، اما في حالة كانت هناك مسافة كبيرة بين الاعداء و اللاعب فسيتم تحريك الأعداء الى اللاعب اعتماداً على حساب فارق المسافة بين موقع اللاعب و موقع الأعداء و تخزينها في المتغير MoveToPlayer سيتم استخدامها في الدالة MovePosition كما فعلنا سابقاً و يمكن ملاحظة هذا في السطرين 54 ,53، في حالة لم يتحقق الشرط سيتم تدمير العدو.

الدالة EnemyShoot ولاحظ انها EnemyShoot فنحن لن نقوم بوراثتها بكل معنى الكلمة بل سنستخدمها و لاحظ انها تعديد 4 بارامترات بحيث توفر لنا الامكانيات لتغيير قيمها لاحقاً و لاحظ ان آلية عمل هذه الدالة لا تختلف كثير عن الدالة الموجودة في اللاعب و التي تساعده على اطلاق النار، بل هذه الدالة تعيد بارامتر رابع من نوع float timeleft و سيتوجب علينا تغيير سريع الاطلاق من عدو لأخر.

الدالة RandomDistance من نوع float وهي تختص بعمل مسافة عشوائي بين الاعداء و اللاعب بحيث انها تتلاعب بقيم المتغير maxDistance و هي تعيد بارمتران يحددان اقل و اعلى قيمة عشوائية للمسافة.

الدالة ApplyHealth و هي تختص بصحة الأعداء و لاحظ ان اول شرط وضع هو التحقق من صحة اللاعب في المشهد ففي حالة مات اللاعب سيتم تدمير العدو، اما الشرط الثاني فهو يحدد قيمة صحة العدو و سيتم تدميرة في حالة تحقق الشرط.

الدالة ApplyDamage لاحظ انها من نوع float ايضاً هي public فسنحتاج لاستخدامها لجلب معلومات الضرر عبر اصطدام طلقات اللاعب بالأعداء و سيتم تغيير قيمة الضرر من طلقة لأخرى.

ربما اطلت في الشرح لكن من باب توضيح كل شيء و الاجابة عن تساؤلاتك بخصوص هذا الملف، ففي النهاية مسألة الملف سهله جداً بحيث انني قمت بجمع كل الوظائف في ملف واحد و سيتم وراثتها لاحقاً، الان يجب علينا العودة الى ملف العدو الاول و سيتوجب علينا اضافة شيء منفصل وهو تحريك سلاح العدو الاول و تعديل بعض الاشياء البسيطة.

برمجة العدو الاول:

حسناً هنا سيتوجب علينا التعديل على ملف العدو الاول لهذا قم بفتح ملف العدو الاول و اضف هذا الكود كي نناقشه:

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class EnemyOne : EnemiesInfo {
5.
6.          public Transform PointShoot;
7.          private Transform myGunOne;
8.          public Rigidbody2D b_Enemy;
9.
10.         public override void Start ()
11.         {
12.                 base.Start ();
13.                 myGunOne = transform.FindChild("Gun");
14.                 RandomDistance(10, 3);
15.         }
16.
17.         public override void FixedUpdate ()
18.         {
19.                 base.FixedUpdate ();
20.                 EnemyShoot (b_Enemy, SpeedShoot, timeleft, PointShoot);
21.                 GunEnemyOne();
```

```
22.         }
23.
24.         void GunEnemyOne()
25.         {
26.                 if(Player != null)
27.                 {
28.                         Vector2 playerPos = Player.position;
29.
30.                         float deltaX = myGunOne.position.x - Player.position.x;
31.                         float deltaY = myGunOne.position.y - Player.position.y;
32.                         float angle = Mathf.Atan2(-deltaX, deltaY) * Mathf.Rad2Deg;
33.
34.                         Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
35.                         myGunOne.rotation = rotation;
36.                 }else Destroy(gameObject);
37.         }
38. }
```

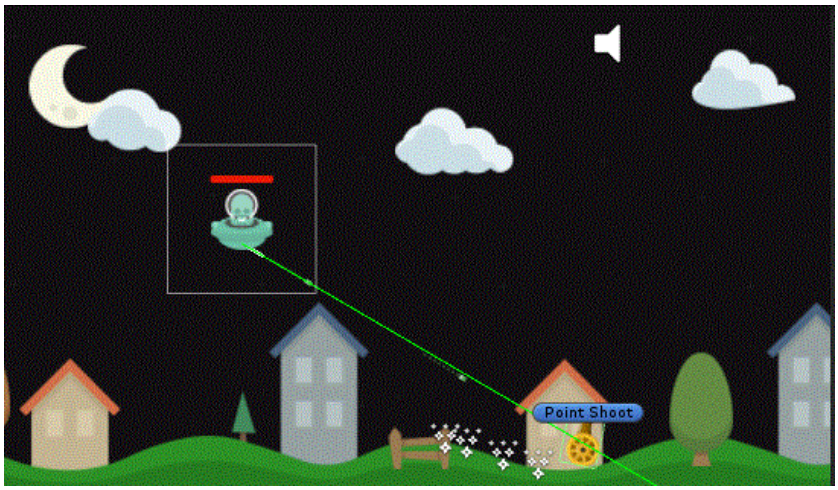
حسناً، بالنسبة للعدو الاول فسيتوجب علينا اولاً البحث عن النقطة التي سيطلق منها النار، و نحن لم نقم بتعريف متغير يقوم بالبحث عن موقع الاسلحة فلكل عدو سلاح خاص به، اذن بشكل سريع قمت بتعريف متغير من نوع transform يمثل السلاح ايضاً متغير آخر يمثل النقطة التي ستخرج منها الطلقات و اخيراً متغير يمثل الطلقة و سنستخدمه في دالة اطلاق النار.

بالنسبة للدالة Start ستجد انني عملت override للدالة و السبب ان هناك كود غير مشترك و عندما تحصل من هذه المسالة تقوم بعمل override للدالة المطلوبة و وضع الشيء الغير مشترك فيها و سيتم تشغيله مباشرة، الان تجد انني استخدمت الدالة RandomDistance و اعطيتها قيمة عشوائية، الان FixedUpdate قمت بعمل override و قمت بإعادة استخدام الدالة EnemyShoot و قمت بتمرير البارامترات(الطلقة، السرعة، الزمن، الموقع) و سيتم وضع الطلقة على هذه الاعدادات، فانا لم اقم بعمل override للدالة لهذا الشيء فقط بل ايضاً لذينا سلاح يجب ان نقوم بتحريكه، الان في الدالة GunEnemyOne سيتم التحقق من موجود اللاعب في المشهد ففي حالة تحقق الشرط سيتم تدوير السلاح، و تلاحظ ان الكود هو الكود الذي شرحته في تدوير اللاعب و هو عبر عمل انسيابية في دوران السلاح، و سيتم توجيه السلاح الى اللاعب و اطلاق النار سيتم مع الدوران.

بالنسبة لأطلاق النار هناك طريقة اخرى تساعدك على عمل ذكاء اصطناعي جميل لعملية اطلاق النار، فهي لن تسمح بأطلاق النار الا في حالة تلامس دوران سلاح العدو متوافق مع دوران اللاعب و لكن كيف يحصل هذا؟ هذه الطريقة ستعتمد على الاشعة بحيث اننا سنرسم شعاع و هذا الشعاع سيتحقق من جميع الكائنات التي تصادم معها و سيتم مقارنة التاج الملامس بتاج الشرط و سيتم اطلاق النار في حالة تلامس هذا الشعاع مع اللاعب، هذه المرة سنستخدم الدالة RaycastAll على شكل مصفوفة و سيتم تهزين جميع الكائنات التي تلامست مع هذا الشعاع في هذه المصفوفة، اذن في الدالة FixedUpdate اضف هذا الكود:

```
1.  public override void FixedUpdate ()
2.  {
3.          base.FixedUpdate ();
4.          GunEnemyOne();
5.
6.          RaycastHit2D[] hits = Physics2D.RaycastAll(myGunOne.position, -myGunOne.up, 20f);
7.          foreach(RaycastHit2D hit in hits)
8.          {
9.                  if(hit.collider.gameObject.tag == "Player")
10.                 {
11.                         EnemyShoot (b_Enemy, SpeedShoot, timeleft, PointShoot);
12.                 }
13.         }
14.         Debug.DrawRay(myGunOne.position, -myGunOne.up * 20f, Color.green);
15. }
```

حسناً تأمل في هذا الكود، انه نفسه المستخدم سابقاً في عمل الدائرة الشعاعية و لكن هنا استخدمت الشعاع RaycastAll و هو عبارة عن شعاع خط مستقيم بحيث ان هذه الدالة تطلب 4 بارامترات بحيث ان الاول يطلب موقع انطلاق الشعاع و الثاني اتجاه الشعاع و الثالث طوله و اخيراً اللاير الذي سيتلامس معه، لاحظ انني تعمدت عدم وضع لاير معين لهذا قمت بعمل الحلقة التي تتحقق من جميع الكائنات المتلامسة مع هذا الشعاع و اعادة تخزين قيم المتغير hit في المصفوفة hits و سيتم التحقق من التاج الموجود في الشرط و سيتم اطلاق النار في حالة تحقق الشرط او فعلاً تلامس الشعاع مع اللاعب! اخيراً استخدمت الدالة DrawRay هي اختيارية ووظيفتها اظهار الشعاع و هذه صورة من داخل المشهد توضح عمل الشعاع.



برمجة العدو الثاني:

بالنسبة للعدو الثاني فانة لن يختلف كثيراً عن العدو الاول لهذا سيتم نسخ نفس الكود و مع القليل من التعديلات، اذن قم باضافة هذا الكود:

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class EnemyTwo : EnemiesInfo {
5.
6.          public Transform[] PointShoot2;
7.          public Rigidbody2D b_Enemy;
8.
9.
10.         public override void Start ()
11.         {
12.                 base.Start ();
13.                 RandomDistance(5,10);
14.
15.         }
16.
17.         public override void FixedUpdate ()
18.         {
19.                 base.FixedUpdate ();
20.
21.                 EnemyShoot (b_Enemy, SpeedShoot, timeleft = Random.Range(3, 5), PointShoot2[0]);
22.                 EnemyShoot (b_Enemy, SpeedShoot, timeleft = Random.Range(3, 5), PointShoot2[1]);
23.         }
24. }
```

دعنا نشرح هذا الكود و نظراً لبساطته فليس هناك الكثير لنقوله، لاحظ في البداية عرفت مصفوفة من نوع Transform و هي تمثل النقاط التي ستخرج منها الطلقات و في العدو الثاني هناك نقطتان فقط. ايضاً عرفت متغير يمثل الطلقة، في Start قمت باستدعاء دالة المسافة ووضع مسافة عشوائية، اما في FixedUpdate قمت بنسخ كود اطلاق النار مرتين بحيث انك لو لاحظت الاول ستجد ان رقمة في المصفوفة هو صفر اما الثاني فهو واحد و وقت اطلاق النار سيتم وضعة بشكل عشوائي بحيث ان كل نقطة تطلقه قنبلة بوقت مختلف، و بهذا الشكل نكون قد انهينا العمل على العدو الثاني:

برمجة العدو الثالث:

ملف العدو الثالث لن يختلف ابداً عن باقي الاعداء بل سيكون اقل استخداماً للاكواد، لاحظ هذا الكود:

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class EnemyThree : EnemiesInfo {
5.
6.          public Transform PointShoot;
7.          public Rigidbody2D b_Enemy;
8.
9.
10.         public override void Start ()
11.         {
12.                 base.Start ();
13.                 RandomDistance(3, 10);
14.
15.         }
16.
17.         public override void FixedUpdate ()
18.         {
19.                 base.FixedUpdate ();
20.
21.                 EnemyShoot (b_Enemy, SpeedShoot, timeleft = Random.Range(3, 5), PointShoot);
22.         }
23. }
```

لاحظ ان لا شيء جديد يستحق الشرح فقط نفس الأكواد، متغير يمثل نقطة الاطلاق، الطلقة، و قمنا بوضع مسافة عشوائية و اخيراً اطلاق النار في الدالة FixedUpdate، بالنسبة لمعلومات التصادم سيتوجب على الطلقة حمل معلومات التصادم و انا اريد ان اضع هذه المعلومات في الطلقة لكي نناقش طريق و اساليب جديد في الحصول على مكونات الكائنات التي تصادمنا معها.

زعيم الاعداء:

زعيم الاعداء بسيط جداً فمهمته هي توليد الاعداء و لكن سنضعه في آخر مرحلة، اذن نشئ ملف برمجي وسمه Boss و اضف الكود التالي:

```
1.          public Transform PointShoot;
2.          public Rigidbody2D[] AddEnemies;
3.
4.          public override void FixedUpdate ()
5.          {
6.                  base.FixedUpdate ();
7.                  EnemyShoot (AddEnemies[Random.Range(0, AddEnemies.Length)], SpeedShoot,timeleft =
    Random.Range(3,7),PointShoot);
8.          }
9.
10.
```

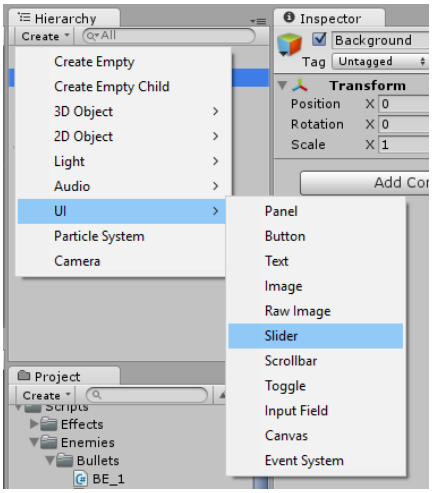
```
11.     void OnCollisionEnter2D (Collision2D Col)
12.     {
13.             base.ApplyDamage(1);
14.     }
15.
```

اذن مصفوفة من نوع Rigidbody2D تمثل الاعداء الذي سيولدهم، اما باقي الأكواد هي نفسها و لكن ترى ان الضرر الذي تحدثه الطلقات في العدو قيمته واحد وهي نسبة ضئيلة جداً وهذا ما سيجعل قتال الزعيم يتطلب وقت.

تصميم وبرمجة الـ Health Slider:

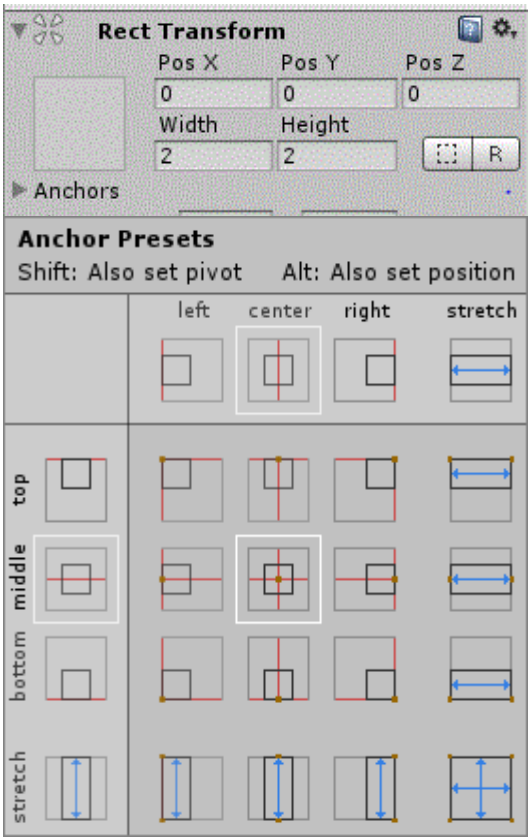
بالحديث عن القوائم فان القوائم في اليونتي تتيح لك عمل العديد من الاشياء و ليس فقط عمل قوائم المهام الرئيسية والثانوية و غيرها بل لها العديد من الوظائف وتتيح لك استعمالها بأكثر من الطريقة. هذه الفقرة سنخصصها لوضع Health Slider للأعداء و هذا الشيء غير ضروري و لكن من باب اعطاء جمالية للعبة، بالنسبة للقوائم في اليونتي ستجدها مرفقة مع النسخة 4.6 الى آخر نسخة وهي توفر العديد الخصائص و لكن لن نناقش جميعها الان و سنخصص فقرة لها.

في النافذة هايركوي (Hierarchy) قم بالضغط على الزر Create و ستجد القائمة UI و فيها العديد من القوائم، منها اختار Slider.



عند انشاء Slider ستجد انه قد اضافة كائنين (EventSystem, Canvas) بحيث ان الـ Sider ابن للكائن Canvas و في حال قمت بخارجة منة ستلاحظ اختفائه اي ان الـ Canvas عبارة عن نطاق لهذه القوائم.

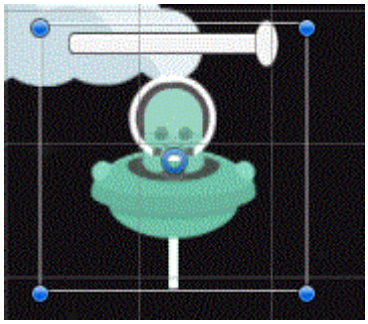
عند انشاءه ستلاحظ ان حجمة كبير جداً و يجب علينا ضبط موضعه مع الاعداء، اولاً قم بوضع العدو الاول في المشهد لكي نستطيع ضبط موقع السلايدر معة. حسناً قم بجعل الـ Canvas ابن للكائن EnemyOne و ستجد ان حجمة لم يتغير بعد، لكن عند الضغط على الـ Canvas تجد العديد من المكونات في الانسباكتور، في Rect transform قم بعمل Reset Position، الان في Canvas ستجد مكون بنفس الاسم و فيه بعض الخصائص مثل نوع العرض و ستجد انها ثلاثة (المساحة العامة، مساحة الكاميراً، مساحة فضاء العالم) قم باختيار World Space و ستجد ان حجم الـ Slider تغير، الان يجب ضبط حجمة مع العدو لهذا قم بضبط اعدادات القائمة Rect Transform كما هو موضح:



كل الذي قمنا بعملة هو تغيير حجم الكونفز وتحديد نوع العرض، لاحظ الخانتين Width و Height ووظيفتهما تحديد طول و عرض الكائنات داخل الكونفز بينما القائمة Scale وظيفتها تغيير حجم الكونفز على جميع المحاور، الان لو لاحظت السلايدر ستجد ان حجمة لم يتغير لأننا لم نغير حجمة، اذن قم بالضغط على Slider و ستجد العديد من المكونات و اهمها قائمة الـ Anchors و ستساعدك على تحديد موقع السلايدر في الكونفز بكل سهوله و ستجد ان هناك عدة طرق ربما لا يوضحها اليونتي بالضبط و لكن عند الضغط على القائمة قم بالضغط على المفتاح Alt و ستجد انه قم تم التحديد على مواقع السلايدر و كل ماعليك فعلة هو تحديد المنقطة التي تريد للسلايدر ان يكون فيها، لاحظ الصورة في الاسفل:

بهذا الشكل نكون قد حددنا موقع السلايدر في وسط الكونفز.

الان بالنسبة لحجم السلايدر ستجد في القائمة Scale ان جميع القيم هي 1 قم بتغييرها الى 0.01 و اجدها مناسبة معي و اخيراً ستجد السلايدر بهذا الشكل:



الان علينا تعديل بعض الاشياء البسيطة مثل مسح الـHandle Slide Area و الـBackground في حالة قمت بهذه الخطوة ستجد ان الـ Fill Area بقى وحيداً في السلايدر، الان يجب علينا تعديل اللون و القيمة، بالعودة الى السلايدر ستجد العديد من المكونات و حالياً سنعمل على المكون Slider، منه ستجد العديد من الخيارات و اولها Interactable قم بإلغائه لكي لا يتم التعامل مع الادخالات، القائمة Transition اجعلها none، الان بالذهاب الى الخيارات الاخيرة ستجد الخانة min value و max value و هي تتيح لك تحديد اعلى و اقل قيمة للسلايدر، اجعل اقل قيم هي الصفر بينما اعلى قيمة هي 100 كما تتناسب مع الصحة، الخيار whole number يساعدك على تحديد نوع القيم اكانت اعداد صحيحة ام كسور عشرية بمعنى ان تكون int ام float في حالة قمت بتصحيح الخانة سيقوم بتحول القيم الى اعداد صحيحة int.

تبقى لنا آخر شيء وهو وضع لون لهذا الشريط، في الشريط Fill Area ستجد كائن ابن باسم Fill قم بتغيير لونة من المكونImage اما باقي الخيارات اتركها كما هي، و بهذا الشكل نكون قد انهينا تصميم الـ Health Slider و سيتبقى لنا برمجته فقط.

بالنسبة لبرمجة السلايدر فكل ماستفعلة هو الوصول الى متغير value و ربط قيمة بقيم الصحة او المتغير health و لكن اولاً يجب علينا استيراد نطاق الاسم UI و الذي منة نستطيع الوصول الى هذه المكونات، اذن قم بفتح الملف EnemiesInfo و قم باستيراد نطاق الاسم using UnityEngine.UI؛

حسناً قم بالذهاب الى القائمة Health التي أنشأناها سابقاً و ستجد متغيرات الصحة، قم باضافة السلايدر كما هو موضح:

```
1.  [Header("Health")]
2.  protected int health;
3.  [SerializeField] protected int maxHealth = 100;
4.  [SerializeField] protected Slider healthSlider;
```

اذن سيستطيع جميع الاعداء الوصول الى هذا المتغير و اردت فعل هذا لأنني سأضع السلايدر على جميع الاعداء، ففي حالة لم ترد اظهاره في جميع الاعداء فبكل بساطة قم بتعريف متغير يمثل السلايدر في العدو الذي تريده، الان قم بالذباب الى الدالة ApplyDamage و اضف الكود التالي:

```
1.  public int ApplyDamage(int damage)
2.  {
3.          health -= damage;
4.          healthSlider.value = health;
5.          return damage;
6.  }
```

لاحظ في السطر 4 استخدامي للمتغير value وهو الذي يختص بتغيير قيم السلايدر و ستجده موضوع في المكون نفسة فكل ماقمنا بفعلة هو الوصول الى هذا المكون وهو على شكل متغير و ربطة في قيم الصحة و التي ستتغير في حالة اصيب العدو. اذن بهذا الشكل نكون قد انهينا العمل على الـ HealthSlider ففي النهاية هو شيء اختياري و ان اردت ان تضفه فضفة.

الطلقات و معلومات التصادم:

كما تعلم ان كل ماقمنا بعملة سابقاً هو تعديل الطلقات و الاعداء و برمجتهم و هذه الفقرة نخصصها للطلقات و كما تعلم ان لذينا 3 نواع من الطلقات و كل طلقة لها وظيفتها الخاصة و لكن معلومات التصادم تبقى هي نفسها و ستجد ان جميع الطلقات في حالة اصطدمت في اللاعب ستقوم بتغيير قيم الدالة ApplyDamage و التي هي static بحيث اننا نستطيع الوصول اليها عبر فئتها مباشرةً.

اولاً دعني اعطيك فكرة عن عمل جميع الطلقات، في البداية لذينا الطلقة الاولى و التي وظيفتها ستكون التصادم فقط ففي حالة تصادمت مع اللاعب ستقوم بانقاص صحة اللاعب، اما الطلقة الثانية فهي القنبلة و يجب عليها البحث عن اللاعب ففي حالة وجدته ستنطلق الية و تتصادم معة الا في حالة تصادمت مع طلقة اللاعب و هذا سيساعد اللاعب على تدمير القنبلة في الهواء، الطلقة الثالثة لن تختلف كثير عن الطلقة الثانية فهي صاروخ و يجب عليها البحث عن اللاعب كسابقتها و لكن ما تعلم على الصاروخ ان يغير دورانه الى اللاعب و سنستخدم الطرق السابقة في عملة التدوير اما معلومات التصادم فستكون مثل سابقتها الثانية بحيث تتيح للاعب تدميرها في الهواء، اذن بهذا الشكل اخذت فكرة عن الطلقة و سيتبقى لنا برمجتها.

قم بإنشاء 3 ملفات بحيث ان كل ملف يمثل كل طلقة خاصة فيه، سنقوم ببرمجة الطلقة الاولى، اضف هذا الكود:

```
1.  void OnCollisionEnter2D(Collision2D Col)
2.  {
3.      if(Col.collider.gameObject.CompareTag("Player"))
4.      {
5.          PlayerController.ApplyDamage(2);
6.      }
7.      Destroy(gameObject);
8.  }
```

لاحظ ان كل ماتقوم به هذه الطلقة هو انقاص صحة اللاعب بمقدار 2 فقط ففي حالة تصادمت مع اي كائن سيتم تدميرها.

بالنسبة للطلقة الثانية سيتوجب علينا تعريف متغيرات تمثل اللاعب و سرعة الطلقة فلن نعتمد على سرعة اطلاق العدو بل على الطلقة في هذا الملف، اذن اضف هذا الكود:

```
1.  private Transform Player;
2.  private Rigidbody2D rb2D;
3.  public float Speed;
4.
5.  void Awake()
6.  {
7.      Player = GameObject.FindGameObjectWithTag("Player").transform;
8.      rb2D = GetComponent<Rigidbody2D>();
9.  }
10.
11. void FixedUpdate ()
12. {
13.     B_Enemy_2();
14. }
15.
16.
17. void B_Enemy_2()
18. {
19.     if(Player != null)
20.     {
21.         Player = GameObject.FindGameObjectWithTag("Player").transform;
22.         Vector2 move = Vector2.MoveTowards
23.                 (transform.position, Player.position, Speed * Time.deltaTime);
24.         rb2D.MovePosition(move);
25.         transform.Rotate(0, 0, 600 * Time.deltaTime);
26.     }else Destroy(gameObject);
```

```
27.         }
28.
29.
30.         void OnCollisionEnter2D(Collision2D Col)
31.         {
32.                 if(Col.collider.gameObject.CompareTag("Player"))
33.                 {
34.                         PlayerController.ApplyDamage(5);
35.                 }
36.
37.                 Destroy(gameObject);
38.         }
```

اذن هذا هو الكود المختص بتحريك القنبلة، هناك اشياء ثانوية مثل دوران القنبلة قد تلاحظها في السطر 25 بحيث انني اعطيت قيمة دوران افتراضية و لم اعرف متغير لها، لاحظ قمت بأنشاء 3 متغيرات تمثل اللاعب و مكون Rigidbody2D و سرعة الاطلاق، بالنسبة للدوال تلاحظ استخدامي للدالة Awake و السبب لكي اتحقق من الاستيقاظ الكلي للطلقة و من ثم سيتم الحصول على المكون و من ثم البحث عن اللاعب.

الان الدالة B_Enemy_2 ووظيفتها تحريك الطلقة وتلاحظ في البداية انني وضعت جميع الأكواد في الشرط و من الضروري التحقق من وجود اللاعب و البحث عنة في حالة وجدة و السبب كتابتي للكود مرتين هو للتأكد من عدم وجود مشكلة اثناء تدمير اللاعب و الطلقة في الهواء، اما التحريك سيتم عبر استخدام الدالة MoveTorards و التي تساعدنا على حساب المسافة و عملها جيد و هذه القيم مخزنة في المتغير move و الذي اعدت استخدامة في الدالة MovePosition في السطر 24، اخيراً في حالة لم يتحقق الشرط سيتم تدمير الطلقة.

بالنسبة لمعلومات التصادم ستجد ان الطلقة ستنفجر في حالة تصادمت مع اي كائن بينما اذا تصادمت مع اللاعب ستقوم بنفاص صحته بشكل مباشر.

اخيراً الطلقة الاخيرة و التي هي عبارة عن صاروخ و وظيفتها لا تختلف كثيراً عن سابقتها فاشيء الجديد المضاف فيها هو الدوران، اذن قم باضافة هذا الكود:

```
1.          private Transform Player;
2.          private Rigidbody2D rb2D;
3.          public float Speed;
4.
5.          void Start()
6.          {
7.                  Player = GameObject.FindGameObjectWithTag("Player").transform;
8.                  rb2D = GetComponent<Rigidbody2D>();
9.
10.         }
11.
12.         void FixedUpdate()
13.         {
14.                 B_Enemy_3();
15.         }
16.
17.         void B_Enemy_3()
18.         {
19.
20.                 if(Player != null)
21.                 {
22.                         Vector2 move = Vector2.MoveTowards
```

```
23.                                         (transform.position, Player.position,Speed * Time.deltaTime);
24.                         rb2D.MovePosition(move);
25.
26.                         Vector2 playerPos = transform.position;
27.                         float deltaX = transform.position.x - Player.position.x;
28.                         float deltaY = transform.position.y - Player.position.y;
29.                         float angle = Mathf.Atan2(-deltaX, -deltaY) * Mathf.Rad2Deg;
30.                         Quaternion rotation = Quaternion.AngleAxis(-angle + 90, Vector3.forward);
31.                         transform.rotation = Quaternion.Lerp(transform.rotation, rotation, 7 * Time.deltaTime);
32.                 }else Destroy(gameObject);
33.
34.         }
35.
36.         void OnCollisionEnter2D(Collision2D Col)
37.         {
38.                 if(Col.collider.gameObject.CompareTag("Player"))
39.                 {
40.                         PlayerController.ApplyDamage(10);
41.                 }
42.
43.                 Destroy(gameObject);
44.         }
```

اذن نفس الكود السابق مع تعديلات طفيفة، لاحظ نفس المتغيرات و نفس الدالة Awake ايضاً الشروط في الدالة b_Enemy_3 و لاحظ من السطر 26 الى 31 هو الكود المختص بدوران الصاروخ وهي الطريقة الانسيابية عملية الدوران اما دالة التحريك فهي نفسها لم تتغير و اخيراً دالة التصادم تجد ان الصاروخ ينفجر في حالة تصادم مع اي جسم و في حالة تصادم مع اللاعب سينقص صحته، و بهذا الشكل نكون قد انهينا العمل على جميع الطلقات و سيتبقى لنا وضع الاعداء في المشهد عبر انشاء الملف EnemiesManager.

برمجة الاضافات:

كما تعلم سابقاً قمنا بالتعديل على الاضافات و لكن لم نبرمجها، هنا سنقوم ببرمجة الاضافات ايضاً علينا عمل درع للاعب بحيث ان في حالة حصل اللاعب على الدرع سيعمل درع الحماية بدورة، اذن العملية سهله فكل ماسنقوم بفعلة هو عمل ملف برمجي واحد يحتوي على انواع الاضافات و بحيث ان كل اضافة تعيد القيم التي تمثلها. اذن قم بإنشاء مجلد باسم Helps بحيث ان هذا الماجد سيمثل الملف البرمجي الخاص بالإضافات، حسناً قم بإنشاء ملف برمجي باسم HelpPlayer و اضف الكود التالي:

```
1.  public enum TypeHelp{
2.
3.      Ammo_1,
4.      Ammo_2,
5.      Ammo_3,
6.      Health,
7.      Force,
8.  }
9.
10. public float Speed;
11. private Transform Player;
12. public TypeHelp typeHelp;
13.
14. void Awake()
15. {
16.     Player = GameObject.FindGameObjectWithTag("Player").transform;
17. }
18.
19. void Update()
20. {
21.     if(Player != null)
22.     {
23.         Vector2 moveToPlayer = Vector2.MoveTowards
24.         (transform.position,Player.position,Speed * Time.deltaTime);
25.         transform.position = moveToPlayer;
26.
27.     }else Destroy(gameObject);
28.
29.     if(PlayerController.Health <= 0 )
30.         Destroy(gameObject);
31.
32. }
33.
34. void OnCollisionEnter2D(Collision2D Col)
35. {
36.     if(Col.gameObject.tag == "Player")
37.     {
38.         if(typeHelp == TypeHelp.Health)
39.             PlayerController.Health += Random.Range(5,10);
40.
41.         if(typeHelp == TypeHelp.Force)
42.             PlayerController.timeleft = Random.Range(10,20);
43.
44.
45.         if(typeHelp == TypeHelp.Ammo_1)
46.             PlayerShooting.Ammo_1 += Random.Range(20,30);
47.
48.         if(typeHelp == TypeHelp.Ammo_2)
49.             PlayerShooting.Ammo_2 += Random.Range(30,40);
```

```
50.                     if(typeHelp == TypeHelp.Ammo_3)
51.                             PlayerShooting.Ammo_3 += Random.Range(1,3);
52.
53.                     Destroy(gameObject);
54.
55.             }
56.     }
```

صراحةً ارى ان الكود سهل جداً فكل ماقمت به هو اعطاء معلومات التصادم لكل enum و تحريك الاضافات سيطبق على الجميع و يمكنك رؤية كود التحريك في Update، اما المتغيرات فهي متغير يمثل اللاعب و آخر يمثل السرعة و الاخير يخزن انواع الاضافات و تلاحظ استخدامي لها في الدالة التصادم، الان كل مستقوم بفعلة هو سحب هذا الملف الى جميع الاضافات و قم بملء المتغيرات و ستلاحظ عملهم بشكل جيد.

برمجة وتعديل الدرع:

بالنسبة للدرع في السرط 42 تجد انني استخدمت المتغير timeleft و أعطيته قيمه عشوائية! ان هذا المتغير يمثل الدرع الذي يقوم بحماية اللاعب و في حالة كانت قيمة المتغير timeleft اكبر من الصفر سيتم تشغيل الدرع، حسناً ربما ترى هذا غريب لأنني لم اشرحه لهذا قم باضافة هذه المتغيرات في الملف PlayerController:

```
1.      [Header("Force Help")]
2.      public GameObject Force;
3.      public static float timeleft;
```

ان هذه المتغيرات ستمثل الوقت المتبقي لتوقف الدرع و متغير يمثل الدرس ككائن، الان اضف هذه الدالة الى نفس الملف:

```
1.      void ForcePlayer()
2.      {
3.              timeleft -= Time.deltaTime;
4.              if(timeleft > 0)
5.              {
6.                      Force.SetActive(true);
7.              }
8.
9.              if(timeleft <= 0)
10.             {
11.                     Force.SetActive(false);
12.                     timeleft = 0;
13.             }
14.     }
```

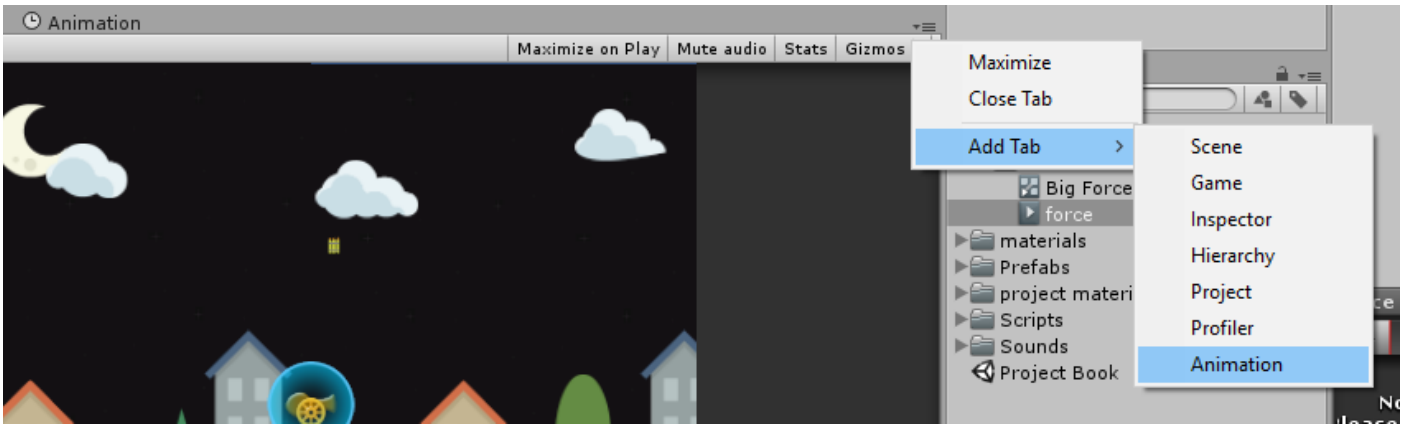
الدالة بسيطة جداً فكل ماتقوم به هو التحقق من حالة الوقت ان كان هناك وقت سيتم تشغيل الدرع وفي حالة لم يتبقى وقت سيتوقف الدرع بهذه البساطة، الان كل ماعليك فعلة هو مناداة الدالة في Update لكي نتجهز لعمل انميشن بسيط للدرع.

بالعودة الى المشروع قم بالبحث عن اضافة الدرع و اسحبها الى المشهد و اتبع هذه النقاط:

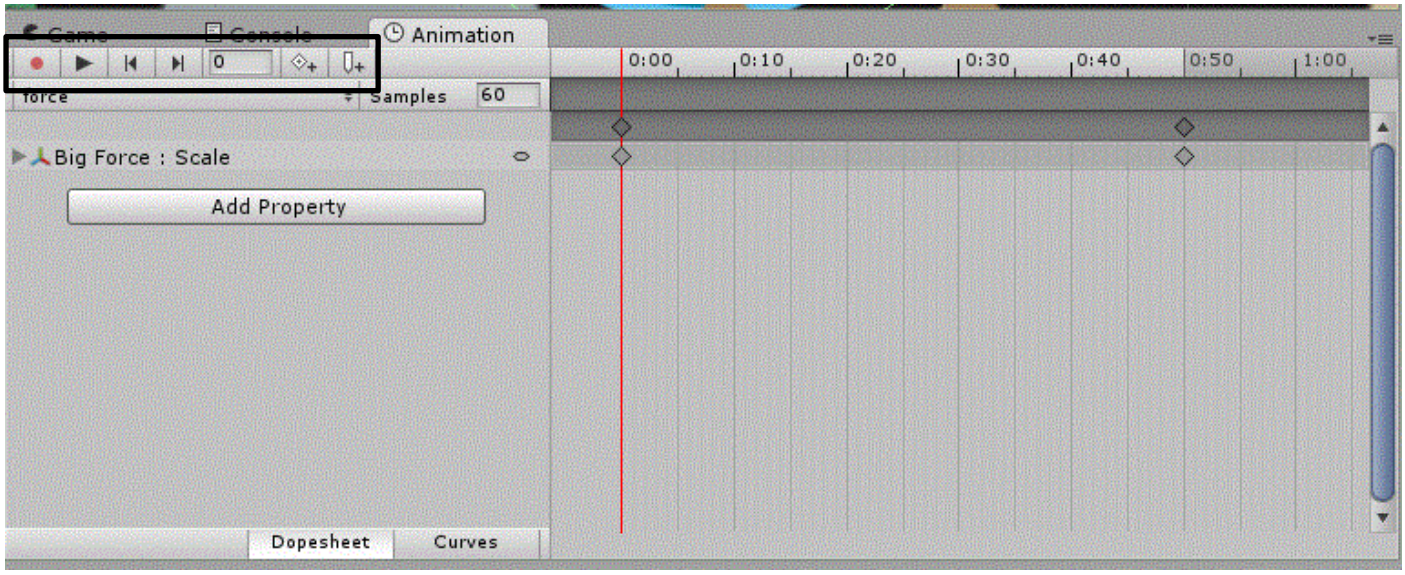
- اجعل الدرع ابن للمدفع.
- قم بتغيير طول و عرض الدرع حسب مايتناسب معك.
- قم بعمل تاج جديد باسم ForcePlayer و اجعلة التاج الرئيسي لهذا الدرع.
- قم بعمل Layer جديد بنفس اسم الدرع بحيث نتيح لهذا الدرع التصادم مع الطلقات فقط.

- اجعل هذا الدرع يتصادم مع طلقات الاعداء فقط.
- بالعودة الى ملف PlayerController قم بسحب الدرع الى المتغير الذي يمثله.

بالنسبة لعمل الانميشن لهذا الدرع فسنعمله بشكل بسيط، اولاً قم باضافة القائمة Animation كما هو موضح في الصورة:



اذن بهذا الشكل ستظهر لك قائمة الانميشن و لكن لن يكون فيها شيء لهذا قم بالتحديد على الدرع و اضغط Create من قائمة الانميشن و سيطلب منك حفظ هذا الانميشن، اولاً قم بأنشاء مجلد يمثل جميع الانميشن في المشروع، و انشئ داخلة مجلد باسم Force و هذا المجلد يمثل انميشن اللاعب، اذن قم بعمل اسم للانميشن و اعمل حفظ و ستظهر لك هذه القائمة:

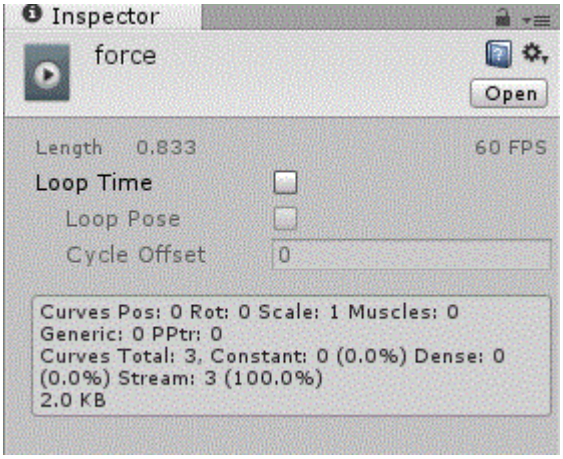


في حالة لم ليكن عندك دراية بهذه القائمة فسأشرحها لك بشكل سريع، لاحظ القائمة المحددة بالمستطيل الاسود، ان هذه الازرار وظيفتها تشغيل الانميشن و إيقافه فقط للتجربة، اما الخانة التي تحتوي الرقم صفر تحدد النقطة التي وصلت اليها في التحريك فلاحظ الخط الاحمر مازال في البداية و تلاحظ ان القيمة صفر و ان قمت بتحريكه ستتغير القيمة حسب موقعة، بالنسبة لآخر ازرار فهي خاصة بوضع حدث معين و وضع مفتاح لتحريك الانميشن و يمنك ملاحظة المفتاح في حالة قمت بتغيير نوع العرض الى Curves اما وضع الاحداث سنستخدمه لاحقاً ووظيفته هو ربط دالة في نقطة معينة في الانميشن. بالنسبة للقائمة Samples فهي تحدد سرعة الانميشن او بشكل ادق تقارب النقاط، اذن هذه الاشياء الاولي التي ستعرفها مع التجربة.

حسناً بالنسبة للدرع فنا سنتعامل مع حجمة فقط و يجب علينا الوصول الى القائمة Scale و ستجد في الصورة زر باسم Add Property و هو خاص بالوصول الى المحاور او المكونات او اي شيء آخر يحتويه الكائن ماعدا الملفات البرمجية، اذن قم بالضغط علية و افتح القائمة

Transform و اختر Scale. في اول نقطة اجعل حجم الدرع هو صفر و قم بسحب الخط الاحمر الى النقطة 50 كما هو موضع و عند هذه النقطة اجعل الحجمة هو 2 او مايتناسب معك و عند تشغيل الانميشن ستجد انه يعمل في حالة كائن الكائن مفعل.

في حالة واجهت مشكلة تكرار الانميشن فبكل بساطة قم بالضغط على مكون الانميشن و الغي تحديد الـ Time Loop. لاحظ الصورة:



اذن هنا نكون قد انهينا العمل على الدرع و هو تلقائياً سيتم الحصول على الوقت في حالة حصل على اضافة الدرع و سيعطينا قيمة عشوائية لتشغيل هذا الدرع و سيتم الغاء تفعيلة في حالة انتهى الوقت وبهذا الشكل نكون قد جهزنا الاضافات، اما الان سيتبقى لنا اخراجها من الاعداء في حالة موتهم.

الحصول على الاضافات من الاعداء:

بما اننا في الفقرة السابقة قمنا بتهيئة الاضافات لاستخدامها برمجياً هنا سنناقش فكرة اخراج الاضافات، و هناك طريقتان لإخراج الاضافات وهما ام عن طريق قتل الاعداء او تلقائياً من منطقة عشوائية وهي تساعدك فقط للحصول على ذخيرة الطلقة الاولى لكي تتمكن من قتل الاعداء، بينما قتل الاعداء سيساعدك على الحصول على ذخائر للطلقات المتاحة و ايضاً الدرع و الصحة في حالة تحققت شروط معينة. اذن عند موت الاعداء سيتم وضع الاضافات، اما الثانية ستكون عن طريق مدير الاعداء الذي لم نعمله بعد و لكن سنضيف الكود فيه في حالة بدأنا بالعمل علية.

للحصول الاضافات من الاعداء قم بتعريف هذه المتغيرات في ملف EnemiesInfo و هي تمثل الاشياء التي سيحتاجها اللاعب لإخراج الاضافات:

```
1.  [Header("Help Player")]
2.  [SerializeField] protected GameObject[] health_force;
3.  [SerializeField] protected GameObject[] Ammos;
4.  protected float timeAddForce;
```

لاحظ معي 3 متغيرات بحيث انهم تمثلان مصفوفتان و هما خاصتان بالطلقات و الصحة و الدرع معاً، و اخيراً متغير يحدد الوقت المنقضِ للحصول على الدرع.

بالنسبة لوضع الشروط سيتوجب علينا تعريف متغير يمثل انواع الطلقات بحيث انه يخزن فيه انواع الطلقات لكي يتمكن العدو من معرفة الطلقات التي يستخدمها اللاعب في المشهد و يضع الاضافات التي تتناسب مع نوع الطلقة. مثلاً ليس من الجيد ان يضع العدو اضافة لطلقة لا يستخدمها اللاعب و لهذا علينا تعريف هذا المتغير الذي يخزن انواع الطلقات. اذن بالعودة الى ملف PlayerShooting سنعرف متغير يمثل المتغير UseBullets، حسناً اضف هذا المتغير الى الملف:

```
1.  public static UseBullets isBullet;
```

الى هنا كون قد انتهينا من الاهم و سنعود الى الملف EnemiesInfo لوضع شروط الحصول على الاضافات.

الان بالعودة الى الملف EnemiesInfo اضف هذا الدالة:

```
1.          void EnemiesHelpPlayer()
2.          {
3.                  if(health <= 0)
4.                  {
5.                          //this code to help player health and force----------------------------------
6.                          if(PlayerController.timeleft <= 0 && timeAddForce <= 0 && PlayerController.Health <= 40)
7.
            Instantiate(health_force[Random.Range(0,health_force.Length)],transform.position, Quaternion.identity);
8.
9.                          if(PlayerController.Health <= Random.Range(30,60) && timeAddForce <= 0)
10.                                 Instantiate(health_force[0],transform.position, Quaternion.identity);
11.
12.                         //this code to help bullet1----------------------------------
13.                         if(PlayerShooting.isBullet == PlayerShooting.UseBullets.Bullet_1 || PlayerShooting.isBullet
    == PlayerShooting.UseBullets.B1_B2|| PlayerShooting.isBullet == PlayerShooting.UseBullets.AllBullets)
14.                         {
15.                                 if(PlayerShooting.Ammo_1 <= Random.Range(20,40))
16.                                   Instantiate(Ammos[0],transform.position, Quaternion.identity);
17.                         }
18.
19.                         //his code to help bullet2----------------------------------
20.                         if(PlayerShooting.isBullet  ==  PlayerShooting.UseBullets.B1_B2  ||  PlayerShooting.isBullet
    == PlayerShooting.UseBullets.B1_B2)
21.                         {
22.                                 if(PlayerShooting.Ammo_2 <= Random.Range(30,70))
23.                                         Instantiate(Ammos[1],transform.position, Quaternion.identity);
24.                         }
25.
26.                         //his code to help bullet3----------------------------------
27.                         if(PlayerShooting.isBullet == PlayerShooting.UseBullets.AllBullets)
28.                         {
29.                                 if(PlayerShooting.Ammo_3 <= Random.Range(0,3))
30.                                         Instantiate(Ammos[2],transform.position, Quaternion.identity);
31.                         }
32.                 }
33.         }
```

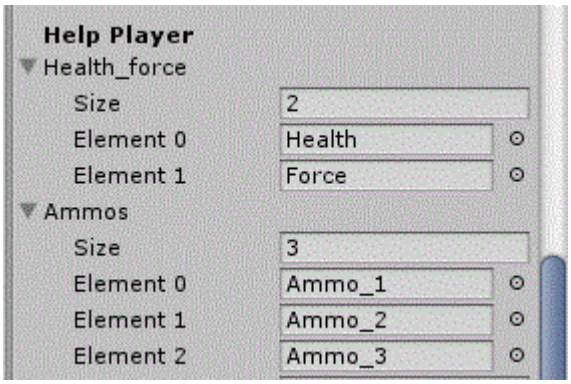
اولاً دعني اقول لك ان هذا الكود سهل جداً فلو لاحظت التعليقات و هي تدل على 4 اشياء قمنا بعملها، ففي اول تعليقات قمت بوضع اول شرطين بحيث ان اول شرط يتحقق من ان الدرع قد انتهى ايضاً يتحقق من وقع وضع الدرع فلن نترك الدرع يخرج بهذه السرعة ففي حالة قام اللاعب بقتل العدو دون ان ينفد الوقت لن يعطيه اضافة الدرع، و اخيراً يتحقق من حالة الصحة، و سيتم تنفيذ احد القيم العشوائية اي درع او صحة و بهذا يعتمد على حظ اللاعب. في الشرط الثاني يتم التحقق من حالة الصحة عبر قيم عشوائية للصحة ففي حالة تحقق الشرط سيتم وضع الكائن رقم صفر وهو يمثل الصحة و لاحقاً يجب عليك وضع الصحة في اول رقم في المصفوفة.

بالنسبة للتعليق الثاني فهو يدل على ان هذا الشرط يتخصص بالتحقق من حالة الذخيرة الاولى و لاحظ استخدامي للمتغير isBullet في الشرط وهذا الشرط يتيح للأعداء اعطاء الذخيرة الاولى في حالة كانت انواع الطلقات اي نوع و سيتم وضع الذخيرة الاولى بعد التحقق من قيمتها و لاحظ ان الرقم صفر في المصفوفة يمثل ذخيرة الطلقة الاولى.

في الشرط الثاني سيتم التحقق من قيمة الذخيرة الثانية و نوع الطلقة المستخدم و تلاحظ انه اما الخيار الثاني (الطلقة الاولى و الثانية) او الخيار الاخير (جميع الطلقات) و سيتم وضع الذخيرة رقم واحد وهي ذخيرة الطلقة الثانية.

الشرط الاخير من تعليقة يدل على انه الطلقة الاخيرة فشرطة لن يتحقق الا في حالة كانت نوع الطلقة هي النوع الاخير و ايضاً التحقق من قيمة الذخيرة فهي الطلقة الثالثة عبارة عن قنبلة ووضعنا عدد القنابل 3 وهذا يفسر القيمة العشوائية التي وضعتها في الشرط، اخيراً سيتم تحقيق الشرط و سيتم وضع الذخيرة رقم اثنين و هي تمثل ذخيرة الطلقة الثالث.

لاحظ ان هذه الدالة بسيطة جداً فهي بشكل كبير تعتمد على الشروط فقط، اخيراً قم بمناداة الدالة في الدالة ApplyHealth اما بالنسبة لملء الخانات الخاصة بالمصفوفات فتأكد انها بهذا الشكل:



اخيراً في حالة سألتنا عن قيمة المتغير timeAddForce فهنا قم بوضع قيمة عشوائية له في Start اما في Update اجعلة يتناقص و هذا سيتم تطبيقه على جميع الاعداء في المشهد.

اذن من الصورة تلاحظ الطريقة التي قمت فيها بملء الخانات في المصفوفات و بهذا الترتيب لكي استطيع تحديد الاشياء التي سيضعها العدو حال موته.

و الى اهنا نكون قد انتهينا من العمل على الاضافات و سيتبقى لنا تعريف المساعدة الاخيرة للاعب و هي الحصول على الطلقة الاولى من اللعبة عبر دالة يتم تشغيلها في الملف EnemiesManager حالة عملة.

انشاء مدير الاعداء EnemiesManager:

بالحديث عن ملف مدير الاعداء وهذا الملف يعتبر ملف مهم جداً فهو الذي يتحكم بالأعداء و له عدة وظائف فليس من الضروري ان يتشابه ملف مع باقي الملفات فهنا مدير الاعداء وظيفته هو اخراج الاعداء الى المشهد اي توليد الاعداء و في كل مرحلة ستستطيع توليد الاعداء الذين تريدهم، على ملف مدير الاعداء ايضاً توزيع دالة مساعدة اللاعب و على الملف ان يقوم بمساعدة اللاعب شخصياً فقط للذخيرة الاولى فنحن لن نترك اللاعب دون طلقات، فكر في ان الذخيرة الاولى للاعب قد نفذت! سيتحتم عليك الموت لتعيد اللعبة و هذه مشكلة و لهذا علينا مساعدة اللاعب على الاقل في الحصول على ذخيرة الطلقة الاولى و هذا تلقائياً سيكون على عاتق مدير الاعداء و الذي سيساعد اللاعب في حالة وصلت طلقته الى مرحلة حرجة.

بالنسبة لنظام اضافة الاعداء في المشهد فهنا فكر معي في طريقة اضافة الاعداء، ماذا ان استخدمنا الدالة Instantiate؟ ان ترك الدالة بهذه الباسطة تعمل قد تكون مشكلة كبيرة في اضافة الاعداء، لاحظ ان تركت الدالة تعمل هنا ستواجه مشكلة الكم الهائل من الاعداء في المشهد، و لهذا علينا الاستعانة بنظام يساعدنا على وضع الاعداء في المشهد بعدد معين بمعنى ان 7 اعداء في المشكلة هم كفاية و لك امر توسيع الرقم لاحقاً و للعلم ان هذا النظام يسمى مسبح الكائنات Objects Pool فهذا النظام سيساعدنا على وضع الاعداء في المشهد بعدد معين.

في حال سالتنا ان كان هناك طريقة معينه لعمل هذا النظام فسأجيبك ان ليس هناك طريقة معينة و انا كما قلت سابقاً استخدم طرقي فبعيداً عن التعقيدات و عمل هذا النظام اعتماداً على الخوارزميات سنقوم بعملة بطريقة اسهل و هي الاعتماد على رقم يتناقص في حالة توليد الاعداء و في حالة موت اي عدو يقوم العدو بإعادة زيادة في الرقم على ملف مدير الاعداء و بهذا الشكل كل عدو سيموت سيدخل بدالة عدو آخر و نكون قد حالينا مشكلة الكم الهائل من الاعداء.

حسناً الان قم بأنشاء ملف باسم EnemiesManager و يفضل ان تضعه في ملف نفصل باسم Managers بحيث ان هذا الملف يمثل جميع ملفات المدراء في اللعبة.

تصميم نظام Object Pooling:

اذن في كبداية سنقوم بتصميم هذا النظام و لكن علينا انشاء كائنات فارغة تمثل النقاط التي سيظهر منها الاعداء، اتبع الخطوات التالية:

- انشئ كائن فارغ وسمه Game و هذا الملف سيضم العديد من الكائنات.
- انشئ كائن آخر وسمه EnemiesManager و اجعلة ابن للكائن Game.
- اخيراً قم بأنشاء 3 كائنات فارغة و قم بتفريقهم بحيث واحد يوضع في الوسط و الثاني على اليمين و الثالث على اليسار لكي نستطيع ان نخرج الاعداء من جميع الجهات، في النهاية اجعلهم ابناء لكائن EnemiesManager.

اذن بهذا الشكل نستطيع برمجة مدير الاعداء، اذن قم بسحب الملف البرمجي EnemiesManager الى الكائن بنفس الاسم، و اضف هذا الكود الى الملف البرمجي:

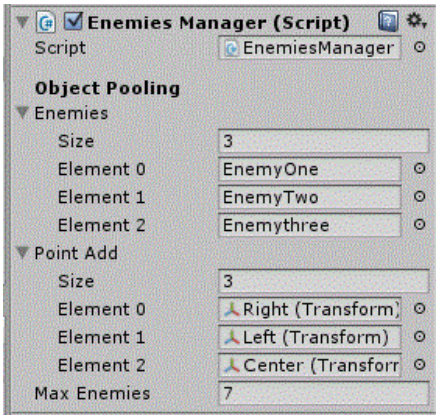
```
1.  [Header("Object Pooling")]
2.  public GameObject[] Enemies;
3.  public Transform[] pointAdd;
4.  private float timeleft;
5.  public static int EnemiesCount;
6.  public int maxEnemies;
7.
8.  void Start()
9.  {
10.         timeleft = Random.Range(2,5);
11.         EnemiesCount = maxEnemies;
12. }
```

```
13.     void Update()
14.     {
15.         ObjectPool();
16.     }
17.
18.     void ObjectPool()
19.     {
20.         if(PlayerController.Health > 0)
21.         {
22.             gameObject.SetActive(true);
23.             EnemiesCount = Mathf.Clamp(EnemiesCount, 0, maxEnemies);
24.             if(EnemiesCount > 0)
25.             {
26.                 timeleft -= Time.deltaTime;
27.                 if(timeleft <= 0)
28.                 {
29.                     timeleft = Random.Range(2,5);
30.                     Instantiate
31.                     (Enemies[Random.Range(0, Enemies.Length)],
32.                     pointAdd[Random.Range(0, pointAdd.Length)].position,
33.                     Quaternion.identity);
34.
35.                     EnemiesCount -= 1;
36.                 }
37.             }
38.         }else gameObject.SetActive(false);
39.
40.         print(EnemiesCount);
41.
42.     }
```

حسناً دعنا نناقش الكود، من النظر الى الكود ستجد انه عبارة عن شروط و ليس هناك امر صعب و ان وجدت اشياء غريبة فسأوضحها لك.

لاحظ في البداية قمت بتعريف العديد من المتغير و اذكر منها مصفوفة تمثل الاعداء و اخرى تمثل النقاط التي سيظهر منها الاعداء و تجد متغير الوقت و في الاخير متغيران يمثلان عدد الكائنات و اقصى عدد ممكن، اذن سنعتمد على المتغير maxEnemies لإظهار الكائنات بعدد معين في المشهد، بينا المتغير EnemiesCount سيختص بتحديد الكائنات الموجودة و التي ماتت. اذن هذه المتغيرات ضرورية حالياً. اذن في ادالة ObjectPool نتحقق من ان اللاعب لازال موجود في المشهد اعتماداً على صحته، ففي حالة تحقق الامر سيتم تفعيل الكائن مدير الاعداء، اسفلة تجد انني اعطيت الحدود للمتغير EnemiesCount و هذا سيوفر علينا وضع الشروط، الان في السطر 24 سيتم التحقق من ان الاعداء اكبر من الصفر اي ان هناك اعداء مخزنين في هذا المتغير ففي الدالة Start ستلاحظ انني وضعت عدد الاعداء بأقصى و هذا سيخزن عدد الاعداء الذي سيتواجدون في المشهد، اذن يجب ان يتحقق الشرط و سيتم وسيتم انقاص الوقت، فلن نترك الاعداء يخرجون بالكم الهائل لا بل سيتم اعطاء وقت عشوائي بين كل عدو و آخر و لاحظ في Start ستجد ان هناك قيم عشوائية لأول عدو و عند اناقص الوقت في الدالة Object Pool سيتم التحقق من ان الوقت يساوي الصفر و سيعطينا قيمة عشوائية اخرى للعدو القادم.

الان سيتم وضع الاعداء في المشهد فلو لاحظت الدالة Instantiate انها تقوم بإعطائنا عدو عشوائي كل فترة و على موقع عشوائي بين الثلاث النقاط و هنا نستطيع التحكم بنوع الاعداء في المشهد، و اخيراً في حالة خرج اول عدو سيؤثر هذا على المتغير EnemiesCount بحيث انه عندما يصل الى القيمة صفر سيتوقف توليد الاعداء، اذن اضف هذا الكود و لاقم بملء المتغيرات في الانسباكتور:



في حالة قمت بتجربة اللعبة و قمت بقتل جميع الاعداء في المشهد ستلاحظ عدم ظهور الاعداء مجدداً و هذه مشكلة كبيرة، و لكن انا لم ابرمج الملف بهذه البساطة بل قد هيئت المتغير EnemiesCount للاستخدام و في حالة اعدت مراجعته ستجد انه Static و هذا يتيح تغيير قيمتة. اذن علينا زيادة قيمة المتغير في حالة تم تدمير كل عدو.

في ملف EnemiesInfo و الذي يحتوي على دالة تتحقق من صحة اللاعب سنقوم باضافة الكود التالي:

```
1.  public void ApplyHealth()
2.  {
3.      if(PlayerController.Health <= 0)
4.          health = 0;
5.
6.      if(health <= 0)
7.      {
8.          health = 0;
9.          EnemiesManager.EnemiesCount += 1;
10.         Instantiate (explosionDead, this.transform.position, Quaternion.identity);
11.         Destroy(gameObject);
12.     }
13. }
```

اذن لاحظ السرط 9 قمت باستعمال المتغير EnemiesCount و تجد انني قمت بزيادة قيمتة في حالة مات العدو و بهذا الشكل نكون قد قمنا بزيادة قيمة المتغير و هذا يعني اضافة عدو آخر.

النظام بسيط جداً و بعيد عن التعقيدات و غالباً عند سماع كلمة نظام نظن ان الامر معقد و لكن كل نظام يختلف في تشكيلته فلاحظ نظاماً شبة مايقال عنة انه نظام و لكن هو عبارة عن كود مثل باقي الأكواد و لكن وظيفته هي التعامل مع عدد الاعداء في المشهد و الحد من تكاثرهم.

وضع الذخيرة الاولى للاعب:

كما تعلم في الفقرة السابقة قمنا ببرمجة الاضافات و تبقى لنا وضع الاضافة الاخيرة للاعب و هي الطلقة الاولى التي سيحتاجها اللاعب لواجهة الاعداء في حالة نفذت الطلقة منة، هنا يجب علينا تعريف متغير يمثل الوقت المنقضِ لوضع الذخيرة و اخيراً متغير يمثل الذخيرة و ننشئ دالة تمثل وضع الاضافة للاعب، اذن ضف هذه المتغيرات اولاً:

```
1.  [Header("Help Player")]
2.  public GameObject Ammo;
3.  public float timehelp;
```

الان قم بكتابة هذه الدالة:

```
1.  void HelpPlayerAmmo()
2.  {
3.      if(PlayerShooting.Ammo_1 <= Random.Range(10,30) && PlayerController.Health > 0)
4.      {
5.          timehelp -= Time.deltaTime;
6.          if(timehelp <= 0)
```

```
7.                          {
8.                                  Instantiate(Ammo,
9.                                  pointAdd[Random.Range(0,pointAdd.Length)].position, Quaternion.identity);
10.                                 timehelp = Random.Range(3,6);
11.                         }
12.                 }
13.         }
```

حسناً، هذه الدالة وظيفتها سهله فهي تتحقق من صحة اللاعب و ذخيرة الطلقة الاولى، و سيتم انقاص الوقت وفي حالة تم التحقق من شرط الوقت سيتم وضع الذخيرة في المشهد بهذه البساطة، في النهاية سيتم اعطاء قيمة عشوائية اخرى للوقت و لكن كبداية قم وضع قيمة عشوائية للوقت في Start و بهذا الشكل نحد من وضع الاضافات بشكل كثيف في المشهد، والى هنا نكون قدا انهينا العمل على ملف EnemiesManager.

انشاء مدير اللعبة GameManager:

بالحديث عن هذا الملف المهم، ان هذا الملف يعتبر من اهم الملفات في اللعبة و سيحتوي على العديد من الاشياء مثل النقاط و تشغيل قائمة النجوم اعتماداً وصول النقاط الى مستوى معين و سيتم حساب قيمة الحياة و اعطائك النجوم التي تستحقها. حالياً لم لنقم بعمل قائمة النجوم لهذا سيتم عمل النقاط كبداية و من ثم عمل قائمة النجوم و برمجتها في هذا الملف.

بالنسبة لطريقة عرض النقاط او Scores سنقوم بعمل طريقة تصاعدية للنقاط اي عند قتل اي عدو سيعطينا نقاط عشوائية و تلقائياً سيتم زيادة قيم المتغير Score الى ان يصل الى الحد المطلوب، اذن المسألة سهله و لأتوجد فيها اي صعوبة.

برمجة النقاط:

قم بأنشاء ملف برمجي باسم GameManager ويفضل وضعة في ملف المدراء Managers. اذن لاحظ هذا الكود:

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class GameManager : MonoBehaviour {
5.
6.          [Header("Score")]
7.          public int HightScore;
8.          private int Score;
9.          public static int maxScore = 0;
10.
11.
12.         void Update()
13.         {
14.                 UpdateScore();
15.         }
16.
17.         void UpdateScore()
18.         {
19.                 Score += 5;
20.                 print("Score :" + Score);
21.
22.                 if(Score >= maxScore)
23.                         Score = maxScore;
24.                 if(maxScore >= HightScore)
25.                         maxScore = HightScore;
26.         }
27. }
```

ان الكود بسيط جداً فكل ماتحتوية هو متغيرات تمثل النقاط، لاحظ ان لذينا 3 متغيرات بحيث ان لذينا متغير يمثل اعلى نتيجة يمكن الوصول لها، متغير يمثل النقاط، و متغير اخير يمثل اعلى نقاط و هي متغيرة و لاحظ انها static لكي نستطيع استخدامها لاحقاً.

في الدالة UpdateScore تلاحظ انني جعلت قيمة الـ Score تزيد و لكن لن نسمح بزيادة القيمة بهذا الشكل فعلينا اعطائها حد معين للزيادة فيه و هذا موضح في السطر 22 بحيث ان الـ Score لن تزيد قيمتها عن قيمة maxScore بينما في السطر 24 لن تزيد قيمة المتغير عن النتيجة الاخيرة و بهذا الشكل نكون قد عملنا حد لزيادة النقاط، الان بمناداة الدالة في Update سننتقل الى الملف EnemiesInfo بالضبط الى الدالة ApplyHealth و الي سنضع فيها قيمة عشوائية للمتغير maxScore، اذن قم باضافة هذا الكود في شرط موت العدو:

```
1.  GameManager.maxScore += Random.Range(100,200);
```

اذن بهذا الشكل سوف يعطينا قيمة عشوائية للمتغير maxScore و بهذا الشكل تتيح للمتغير Score ان يزيد في حدود هذه القيمة العشوائية. اخيراً سيتبقى لنا حفظ النقاط و سنتعرف على فئة جديد في الفقرة القادمة.

الفئة PlayerPrefs:

بالنسبة لحفظ النقاط فأن اليونتي يوفر لنا العديد من الطرق لحفظ النقاط و لاشك ان الطرق الشائعة لحفظ النقاط في العديد من الالعاب هي عن طريق ملفات xml او binary بحيث ان xml ينشئ ملفات قابلة للقراءة و الكتابة بينما binary ينشئ ملفات قابلة للقراءة فقط، بالنسبة لحفظ النقاط هنا سنستخدم الفئة PlayerPrefs فهي توفر لنا دوال للقراءة و اخرى للكتابة و لها 3 انواع (float, int, String) و لك حرية استخدام ايً منهم ففي النهاية لكل دالة قراءة دالة كتاب.

بالنسبة للدالة التي سنستخدمها فهي من نوع int و كما تلاحظ هذه هي دالة القراءة PlayerPrefs.GetInt() و هذه هي دالة الكتابة الخاصة بها PlayerPrefs.SetInt() بحيث ان دالة القراءة ستطلب منك كتاب اسم المفتاح كما يعبر عنه في اليونتي، بينما دالة الكتابة ستطلب منك وضع اسم للمفتاح و القيمة التي تريد حفظها و اخيراً استخدام الدالة PlayerPrefs.Save() لحفظ التغيرات. اذن العملية بسيطة جداً و حالياً سنعرف دالة تخزن كل هذا و سنستخدمها لاحقاً، قم بكتابة الدالة التالي في نفس الملف:

```
1.  void SaveScore()
2.  {
3.      PlayerPrefs.Save();
4.
5.      if(Score > PlayerPrefs.GetInt("Score"))
6.      {
7.          PlayerPrefs.GetInt("Score");
8.          PlayerPrefs.SetInt("Score", Score);
9.      }
10. }
```

لاحظ الشرط، ان هذه الشرط يتحقق من قيمة النقاط و هل اي اكبر من القيمة المحفوظة فشكل منطقي ليس هناك قيمة محفوظة و لهذا سيتم اعطائنا قيمة و سيتم حفظها و كما تلاحظ في السطر 8 ان اسم المفاتح هو Score و يخزن فيه قيمة المتغير Score و سيتم اعطائنا اعلى نتيجة للنقاط فقط.

في حالة اردت اعادة قيمة النقاط الى حالتها الاصلية فهنا طريقتان لحذف المحفوظ، اما عن طريق حذف المفتاح او عن طريق حذف جميع المفاتيح في اللعبة و من خلال هذا الكلام يتضح ان هناك دالتنا.

- للدالة الاولى و هي PlayerPrefs.DeleteAll() ففي حالة قمت باستخدامها سيتم حذف جميع المحفوظات في اللعبة.
- الدالة الثانية و هي PlayerPrefs.DeleteKey() وهي تطلب متغير من نوع String او اسم المفتاح الذي تريد حذفة.

اذن نظام الفئة PlayerPrefs يتيح لنا حفظ النقاط و اشياء اخرى، فاستخدامنا لهذا الدالة ستساعدنا لاحقاً على حفظ النقاط و الانتقال بين المراحل. اخيراً سيتبقى لك مناداة الدالة في الدالة UpdateScore.

ان هذا هو الشكل الاولي لملف GameManager لهذا سنقوم باستخدامه في برمجة القوائم من اظهار قيمة و قوائم و غيرها في فقرة (قوائم لعبتنا).

اضافة و برمجة المؤثرات المرئية:

سابقاً تحدثنا بشكل سريع عن المؤثرات المرئية عندما اضفنا مؤثرات مرئية لطلقات الاعداء اما هذه الفقرة ستختص بعمل جميع المؤثرات المرئية الخاصة بالأعداء و اللاعب و سنعتمد على نظام الجزيئات Particle System كما عملنا سابقاً، هناك العديد من المؤثرات التي يجب علينا عملها و اذكر منها تصادم الطلقات و انفجار الاعداء و الانفجار الكلي للاعب و يترتب على ذلك حريق كبير، ففي النهاية سنقوم بالتعديل على انفجار واحد و اضافة العديد من الانفجارات من خلال استنساخه.

المؤثرات التي سنعملها هي:

- مؤثر انفجار اللعبة.
- مؤثر اشتعال اللاعب.
- مؤثر انفجار الاعداء.
- مؤثر اطلاق النار من اللاعب.
- مؤثر اطلاق النار من الاعداء (العدو الاول فقط).
- مؤثر انفجار الاعداء(وهذا يشمل زعيم الاعداء) و القنابل.
- مؤثر تصادم الطلقات.

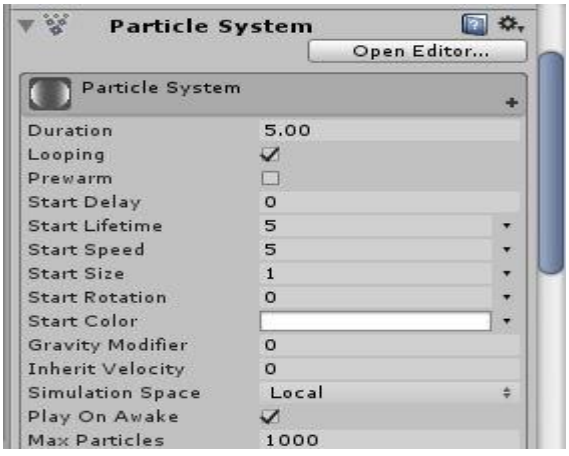
اضافة المؤثرات المرئية:

اذن البداية ستكون من عمل تأثير لتصادم الطلقات في المشهد ثم نقوم بالتعديل على بعض القيم لكي ننشئ الانفجارات الكبيرة، اتبع هذه الخطوات:

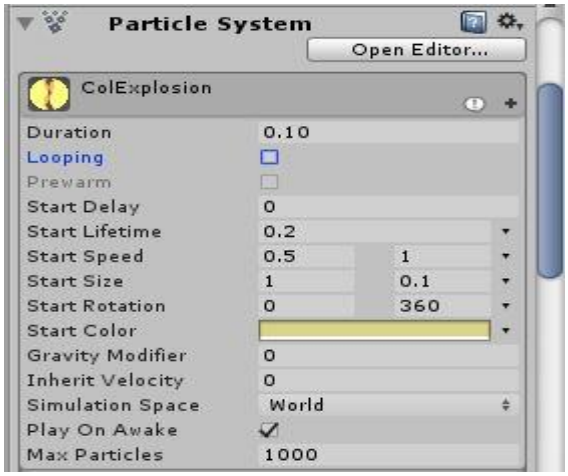
- قم بإنشاء ملف باسم Effects و هذا الملف سيضم جميع التأثيرات الموجودة في اللعبة.
- اضف Particle System الى المشهد لكي نعدل علية.

حسناً، قم بتسمية هذا الـ Particle System باسم ColExplosion اما بالنسبة للصور فستجد في ملف Smoke العديد من الصور و تستطيع استخدام ايً منها و لكن تأكد من ان نوع الصورة هو Texture بحيث ان الاسود يعني الشفافية.

حسناً قم بأنشاء material وتأكد من ان الـ Shader هو Particles/Additive لكي وهو يحدد ان نوع الماتريل هو للبارتكل سيستم.



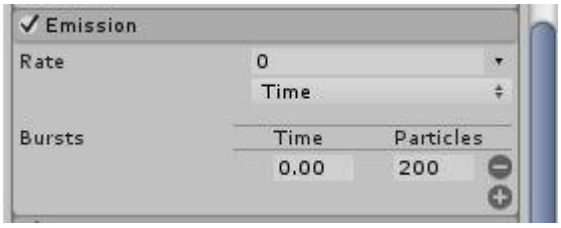
بفتح البارتكل تلاحظ اننا عدنا الى القائمة المعتادة و تجد القيم الافتراضية، الشيء الذي يجب عملة لهذا التأثير هو عدم اطالة عمره في المشهد و ايضاً الغاء تكرار التأثير و يجب ان يظهر مرة واحدة ثم يختفي و هذا يفسر سبب ان هذا التأثير فقط للتصادم. اذن قم بالتعديل على القيم التالية اعتماداً على الصورة الثانية:



لاحظ اننا قمنا بإلغاء تكرار هذا التأثير ايضاً عمرة في المشهد Start Lifetime قصير للغاية و بينما السرعة سيتم اعطائها قيمة عشوائية كل مرة و ستلاحظ اختلافات بسيطة في طول التأثير و هذا جميل، بالنسبة للحجم نفس الشيء سوف يعطينا رؤية جميلة للتأثير، و الدوران هو شيء اختياري، بالنسبة للون فأن هذا اللون سيمتد على طول التأثير و لاحظ انني استخدمت صورة تتناسب مع اللون، في الاخير تركت جميع القيم كما هي.

ان هذا التعديل السريع سوف يعطيك شكل تأثير التصادم كبداية فيجب علينا التعديل على السرعة و تغير الحجم و غيرة.

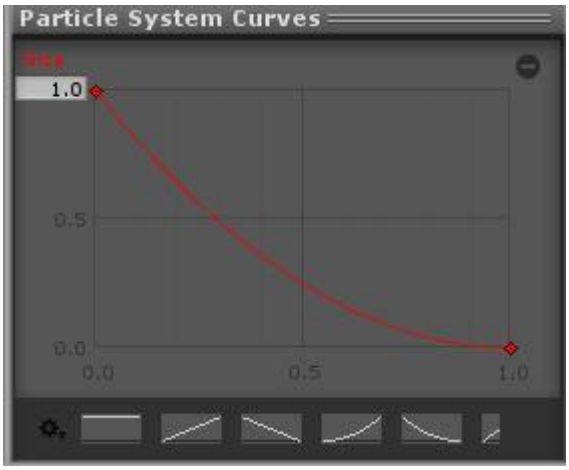
في القائمة Emission سنقوم باضافة موجات لهذا التأثير فهي تظهر بشكل سريع، اذن قم باضافة هذه القيم:



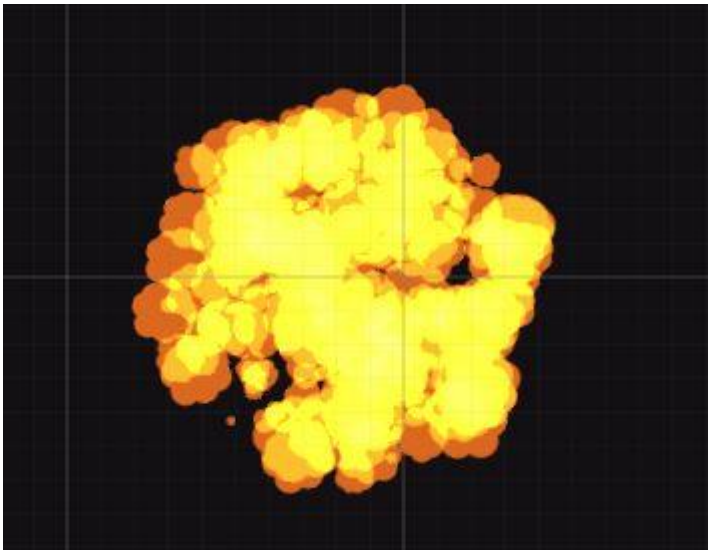
لاحظ ان في اول تشغيل او بالأصح في اول ثانية سيتم وضع 200 بارتكل في المشهد، في حالة رأيت انه غير مناسب قم بتغيير عدد البارتكلز.

بالنسبة للقائمة Shape سيتوجب علينا من خلالها وضع شكل لظهور البارتكلز ففي حالة فتحت القائمة ستجد العديد من الخيارات مثل القائمة Shape و التي تعطيك العديد من الاشكال، ايضاً Radius و هي تحدد قطر هذا الشكل وهو يعتمد على الشكل الذي قمت بتحديده فانا قمت باختيار Sphere و اجدها مناسبة.

سننتقل الى القائمة Size Over Lifetime وهي تحدد حجم البارتكلز طوال فترة بقائهم في المشهد و عند تفعيل هذه القائمة ستجد مخطط بياني و فيه العديد من الخطوط البيانية الافتراضية و يمكنك تجربتها فانا قمت باختيار الشكل التصاعدي لحجم التأثير:

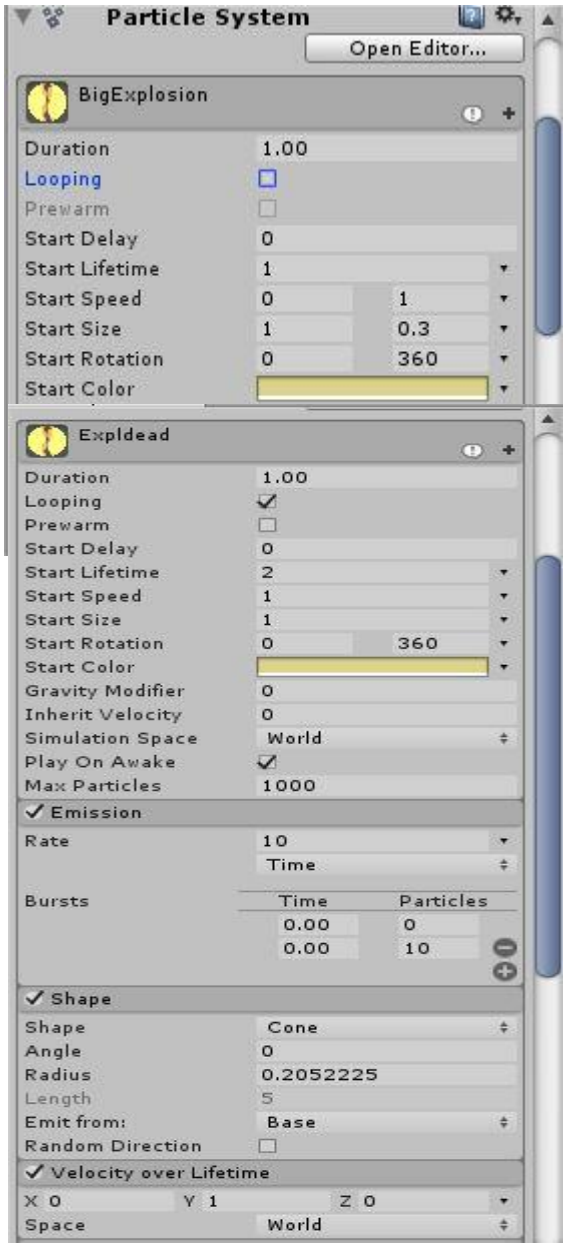


اخيراً في القائمة Renderer قم بوضع صورة التأثير الذي تريده و بهذا الشكل نكون قد جهزنا تأثير التصادم و هذا هو شكلة النهائي:



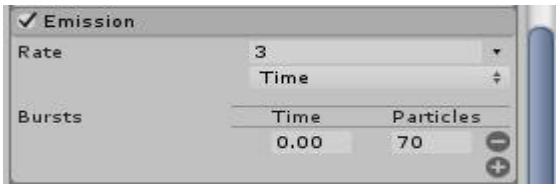
اخيراً قم بسحبة الى الملف Effects بحيث نستطيع استنساخه لاحقاً.

بالنسبة للانفجار الكبير وهو تأثير موت الاعداء و اللاعب ايضاً و انفجار القنابل و سنقوم بعمل انفجار واحد كبير ثم ننسخه و ننشئ انفجار كبير لزعيم الاعداء و اخيراً نعمل تأثير النيران التي تظهر عند موت اللاعب. اذن بالتعديل على نفس التأثير السابق قم بتغيير القيم كما هو موضح:



لاحظ القيم التي وضعتها فهي تتيح للانفجار الانتشار بمساحة اكبر من سابقة، وكما هو موضح ان مدى طول البارتكلز في المشهد هو 1 و هذه القيمة كافية، بينما السرعة سيتم اعطائها بشكل عشوائي و اخيراً الحجم، بالنسبة للون و باقي الاشياء فهي موضوعة على حالتها العادية.

القائمة Emission سنجعلها تعطينا موجة ضعيفة للانفجار اعتماداً على القيم التالية:



اخيراً باقي القيم ستضل كما هي اعتماداً على التأثير السابق، و ستلاحظ ان طول و عرض الكائن هو 1,1,1 فهو تأثير انفجار الاعداء و اللاعب و القنابل، قم بتسميته BigExplosion الان قم بنسخ واحد آخر و اجعل طولة و عرضة 2,2 و قم بتسميته BigExplosionBoss و هذا خاص بانفجار الزعيم، سيتبقى لنا عمل تأثير للنار عند موت اللاعب، اذن قم بنسخ واحد اخير و سمه FirePlayer او اي اسم دو معنى اجعلة ابن للكائن Cannon و اضبط موضعه في Position.

حسناً قم بتغيير القيم كما هو موضح في الصورة:

اذن هذه الصورة ستساعدك على ضبط اعدادات النار.

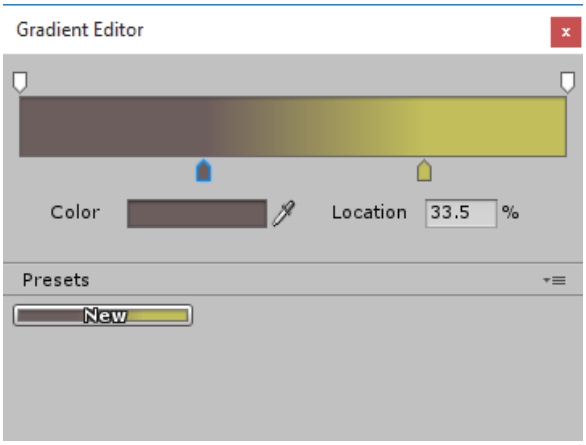
بالنسبة لامتداد البارتكلز فهو كبير 2 و تجد السرعة والحجم موحدين و هذه القيم الاولية بينما باقي القيم كما هي.

في القائمة Emission لاحظ ان الموجات ضعيفة جداً و يتم اعطائنا 10 بارتكلز نكمل دورتها كما هو موضح في القائمة Rate.

في القائمة Shape و هنا اخترت الشكل Cone و يمكنك التلاعب بحجمة بحيث يصبح بحجم اللاعب.

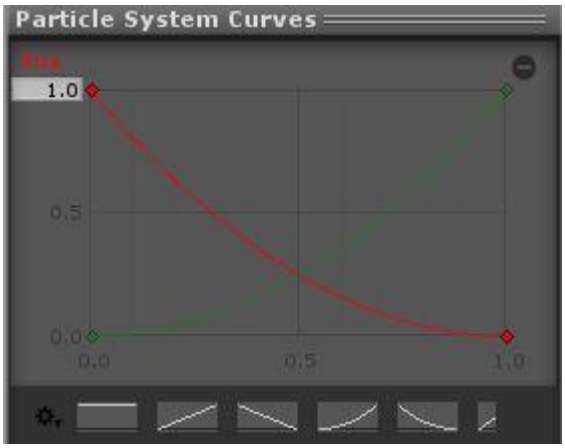
اسفلة تلاحظ استخدامي للقائمة Velocity over Lifetime و هذه القائمة خاصة بعمل سرعة لهذا البارتكلز على محاور مغين و تلاحظ المحاور الثلاثة و القيمة 1 تعتمد على نوع المساحة فمثلاً لو وضعته على World سيتم عمل سرعة للبارتكلز الى الاعلى، بينما Local سيجعل التأثير يعمل سرعة الى الاسفل.

الان سنذهب الى القائمة Color over Lifetime و علينا عمل تأثير الوان للبارتكلز و حال تفعيل القائمة ستجد ان اللون ابيض و علينا عمل تدرج لون بحيث يتغير اللون من البداية الى النهاية، لاحظ الصورة:

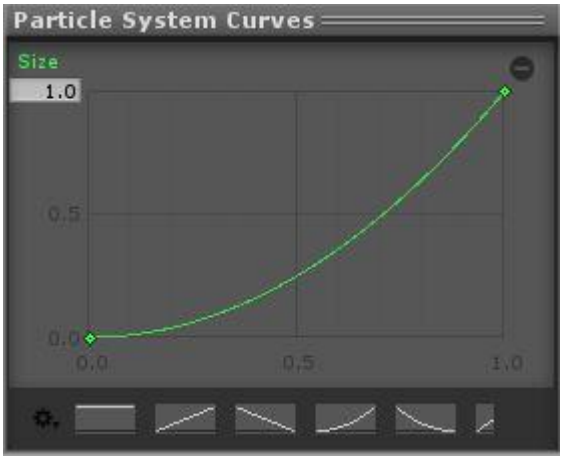


لاحظ كيف يبتدأ لون البارتكلز و كيف ينتهي، ان هذا التدرج يعتمد على مدى رؤيتك لهذا التأثير و حالياً اجد ان هذه القيم مناسبة مع التأثير.

اخيراً علينا التلاعب بمخطط القائمتين Size by Speed و Size over Lifetime، بالنسبة للقائمة الاولى اجعل مخططها كما هو موضح:



اما القائمة الثانية فأجعل مخططها بهذا الشكل:



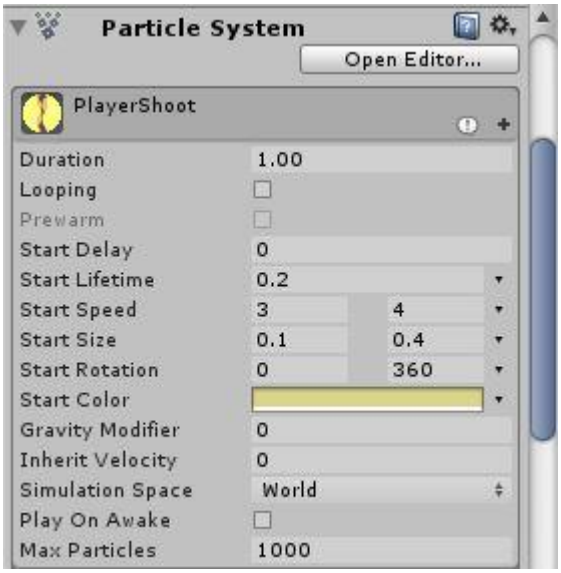
من خلال هذا الرسم سيقوم بجعل البارتكلز تنتهي و تتقلص حجمها و سرعتها تقل في النهاية و هذا هو مفهوم المخطط، يمكنك تجريب القيم الافتراضية و قد تناسبك، اخيراً هذا هو شكل تأثير النار الخاص باللاعب.

و اضن ان من خلال القيم التي وضعتها سيتم اعطائك نفس الشكل و ستلاحظ شكل التأثير و طريقة انتهائه و هذا يطفى لمسة جميلة حال انفجار اللاعب.

تبقى لنا اخيراً ربط هذه المؤثرات في الكائنات و كل ماسنقوم بفعلة هو عمل متغير يمثل هذه الكائنات و حالة انفجار الكائن يقوم بوضع هذا المؤثر.

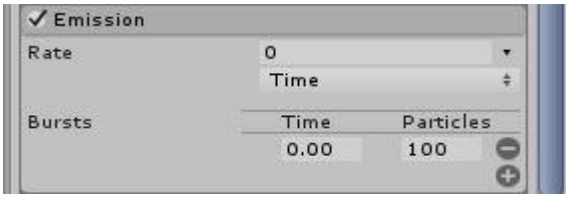
شيء ثاني علينا عمل مؤثر اطلاق النار من اللاعب وهذه العملية سهله جداً فهو نفسة التأثير ColExplosion، اذن قم بسحبة الى المشهد و غير اسمة الى PlayerShoot و اجعلة ابن للكائن Point Shoot في Cannon و تأكد من ضبط موضعه بحيث تستطيع ان ترى تأثير الاطلاق بشكل جيد.

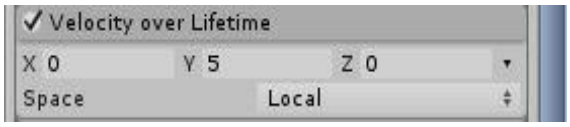
اذن علينا ضبط اعدادات التأثير لهذا قم بتغيير جميع الاعدادات الاولية كما هو موضح في الصورة:

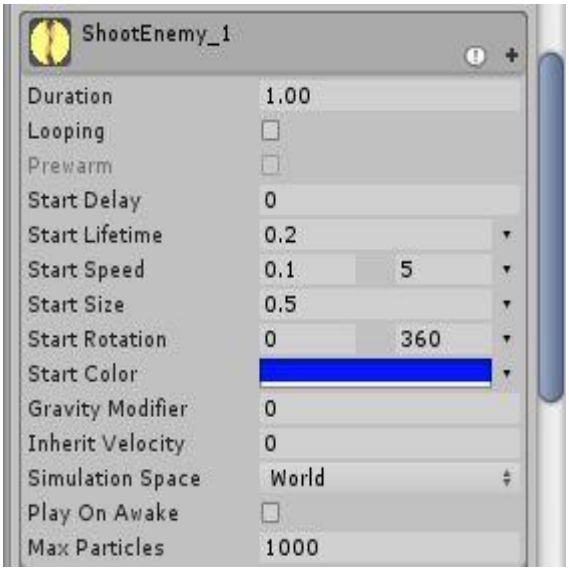


بالنسبة للعدو الاول فأننا نعرف انه يطلق الليزر لهذا يجب علينا عمل مؤثر للإطلاق الليزر، اذن العملية ستكون على المؤثر ColExplosion فبسحبة الى المشهد قم بتغييرة اسمة الى EnemyShoot و اجعلة ابن للكائن PointShoot1 في سلاح العدو الاول.

حسناً قم بتغيير اعدادات القوائم الموضحة في الصورة:







اذن ستلاحظ تكون الليزر بسيط و يمكنك تغيير القيم الى ان تتناسب معك او اعتماد القيم الحالية.

و الى هنا نكون قد انتهينا من عملية تصميم المؤثرات المرئية وسنذهب الى برمجتها ☺ .

برمجة المؤثرات المرئية:

بالحديث عن برمجة المؤثرات المرئية فهنا يتوجب علينا وضع بعض الاشياء المهم الاساسية مثل اختفاء التأثير و يجب ان يكون عن طريق ملف برمجي يعطي عمر لهذا التأثر و يختفي حال انتهائه، الشيء الثاني هو عملية وضع التأثير حال انفجار الكائنات و بشكل واضح سنستخدم الدالة Instantiate و سيتم تطبيق هذا الامر على جميع الاعداء و اللاعب و الطلقات ايضاً سنستخدم الفئة ParticleSystem لكي نستطيع تشغيل مؤثر اطلاق النار حال الاطلاق. اذن لنقوم باضافة التأثر الاولي للاعب. اضف هذه المتغيرات الى ملف PlayerController:

```
1.  [Header("Effects")]
2.  public GameObject explDead;
3.  public GameObject fireExpl;
```

لاحظ وجود متغير يمثل الانفجار و آخر يمثل النار. حسناً في الدالة ApplyHealth قم بالتعديل في الشرط كما هو موضح:

```
1.  void ApplyHealth()
2.  {
3.          if(Health <= 0)
4.          {
5.                  Sprite_r.sprite = spriteDeath;
6.                  ApplyMove = false;
7.                  WheelController.MoveWheel = false;
8.                  PlayerShooting.ApplyShooting = false;
9.                  rb2D.isKinematic = true;
10.                 fireExpl.SetActive(true);
11.
12.                 Instantiate(explDead, transform.position, Quaternion.identity);
13.
14.                 Health = 0;
15.         }else{
```

```
16.                 fireExpl.SetActive(false);
17.
18.             }
19. }
```

لاحظ ان كل ماقمت بفعلة هو وضع التأثير اعتماداً على الدالة Instantiate و قمت تفعيل النار، ففي حالة لم يتحقق الشرط سيتم الغاء تفعيل النار.

بالنسبة للأعداء قم بتعريف هذا المتغير في الملف EnemiesInfo بحيث ان هذا المتغير يمثل الانفجار:

```
1.  [SerializeField] protected GameObject explosionDead;
```

اما في الدالة ApplyHealth اضف الكود التالي:

```
1.  Instantiate (explosionDead, this.transform.position, Quaternion.identity);
```

اذن بهذا الشكل سيتم وضع الانفجار حال موت الاعداء.

بالنسبة للطلقات ستقوم الطلقة باخذ تأثير التصادم و يجب علينا تعريف متغير يمثل التأثر و اخيراً وضعة حال التصادم، اضف هذا المتغير في جميع ملفات الطلقات:

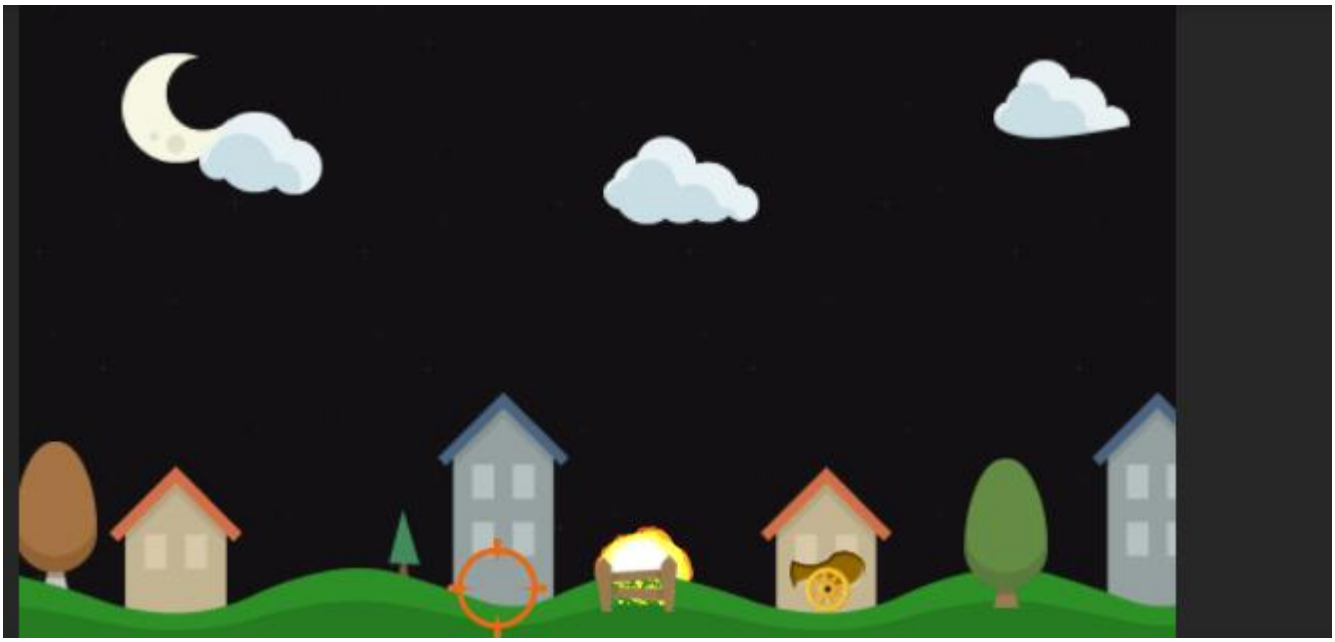
```
1.  public GameObject ExplosionCol;
```

اخيراً قم بوضع هذا السطر في حال التصادم:

```
1.  Instantiate(ExplosionCol, transform.position, Quaternion.identity);
```

و بهذا الشكل نكون قد برمجنا تأثير تصادم الطلقات، بالنسبة للقنبلة فبكل بساطة قم وضع الانفجار الكبير في المتغير الذي يمثله و ايضاً زعيم الاعداء نفس شيء.

اخيراً قم بملء المتغيرات التأثيرات و سيتم ستلاحظ ظهور التأثيرات حال التصادم او موت الاعداء.



عند ظهور التأثيرات ستلاحظ انها مازالت موجودة في المشهد حتى في حال انتهاء تشغيلها و هذه تعتبر مشكلة لو تركنا الطلقات بهذا الشكل سيتم تخزينا على شكل كائنات و حتى في حال عدم عملها، لهذا الشيء علينا عمل مدة زمنية لبقاء التأثير في المشهد و من ثم تدميرة. اذن قم بأنشاء ملف برمجي باسم EffectsInfo او اي اسم دو معنى و اضف الكود التالي:

```
1.  public float timeleft;
2.
3.  void Start()
4.  {
5.          Destroy(gameObject, timeleft);
6.  }
```

لاحظ ان لذي متغير يمثل الوقت المنقضِ و اخيراً استخدمت الدالة Destroy في Start و سيتم تدمير التأثير حال انتهاء الوقت. اذن بهذا الشكل نكون قد عملنا حد لبقاء التأثيرات في المشهد، الان قم بسحب هذا الملف الى جميع التأثيرات عدا النار فلا نريد للنار ان تختفي لأننا قد وضعنا لها شرط في ملف اللاعب.

الان علينا الذهاب الى الملف PlayerShooting واتاحة ظهور تأثير اطلاق النار. اذن قم بتعريف هذا المتغير:

```
1.  public ParticleSystem EffectShoot;
```

اذن لاحظ هذا المتغير فهو من نوع ParticleSystem بحيث سيطلب لاحقاً سحب الكائن ParticleSystem اليه.

في الشروط التي تمثل اطلاق النار قم باضافة الكود التالي:

```
1.  EffectShoot.Play();
```

حسناً ان المتغير EffectShoot من نوع ParticleSystem وهو يوفر العديد من الدوال و منها الدالة Play و هي تختص بتشغيل المؤثر فكما تعرف سابقاً ان مؤثر اطلاق النار ليس Loop ففي هذه الحالة سيتم تشغيل المؤثر مرة واحد فقط ثم يتوقف و حل تطبيق الشرط سيتم تشغيل المؤثر و بهذا الشكل نستطيع ان نرى اطلاق النار بالاضافة الى التأثير الخاص به.

بالعودة الى المشروع قم بسحب مؤثر اطلاق النار الموجود في الكائن Cannon الى المتغير الخاص به.

بالنسبة للأعداء فان علينا وضع مؤثر واحد فقط للعدو الاول فكما تعلم لذينا مؤثر اطلاق الليزر للعدو الاول لهذا يجب اضافة برمجياً.

في الملف EnemiesInfo اضف المتغير التالي:

```
1.  [SerializeField] protected ParticleSystem FireShoot;
```

اذن هذا المتغير سيمثل مؤثر اطلاق النار، الان في الدالة EnemyShoot اضف الكود التالي:

```
1.  if(gameObject.CompareTag("EnemyOne"))
2.          FireShoot.Play();
```

ان اضافة هذا الشرط سببه اننا فقط نريد من العدو الاول اظهار تأثير اطلاق النار فيجب التحقق من التاج هذا العدو ففي حالة تحقق الشرط سيتم وضع التأثير، في حالة كتبنا الكود دون الشرط سيشكل هذا مشكلة حال اطلاق باقي الاعداء وقد يطالبك بسحب مكون نظام الجزيئات ParticleSystem لهذا ضع هذا الشرط لتجنب المشكلة.

في العدو الاول قم بسحب المؤثر الى المتغير الذي يمثل و من ثم جرب اللعبة و لاحظ تأثير اطلاق النار ☺ .

إضافة المؤثرات الصوتية:

ان المؤثرات الصوتية عامل مهم فهي التي تضيف جو الحماس و المرح الى اللعبة وتساعد اللاعب على التفاعل مع اللعبة بشكل كبير، هناك عدة طرق لعمل المؤثرات المرئية اما بتصميمها، بتنزيلها او شرائها من النت فبشكل مباشر سنعتمد على الاصوات الموجودة في مواد المشروع.

المؤثرات الصوتية التي سنستخدمها في المستوى هي:

- موسيقى الخلفية.
- صوتين اطلاق نار للاعب.
- ثلاثة اصوات اطلاق نار للأعداء.
- صوت لتصادم الطلقات.
- صوت لانفجار الطلقات و تدمير الاعداء.

بالنسبة للقائمة الرئيسية و القائمة الصغرة فسنستخدم:

- صوت للمرور على الازرار.
- صوت لضغط الازرار.
- صوت للتبديل بين تفعيل الاصوات و الغائها.

دعنا نناقش آلية عمل الاصوات في اليونتي:

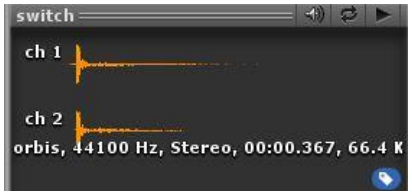
ان اليونتي يوفر لنا عدة عناصر لتشغيل الاصوات و هي:

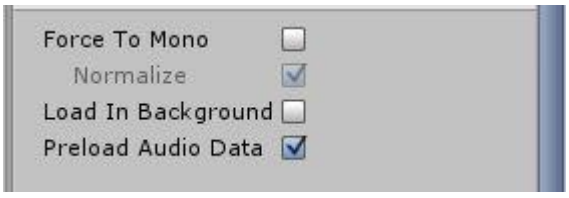
- Audio Listener: و هذا المكون يمثل الاذن التي تسمع الاصوات و افتراضياً هو موجود في الكاميرا ففي حال قمت بحذفة لن تسمع الاصوات.
- Audio Source: ويمثل هذا المكون الصوت في المشهد و يمكن وضعة على شكل 3D او 2D ويوفر عدة خيارات للتحكم بالصوت مثل Volume (وضوح الصوت)، Pitch(حدة الصوت)، Spatial Bland وهذا الخيار يتيح لك وضع الصوت في حالتين 3D او 2D.
- Audio Clip: هذا المكون يمثل مقطعاً صوتياً بحيث يتيح للاعب سماع الصوت عن بعد او قرب بحيث يكون 3D او 2D.

حسناً لنبدأ بعمل المؤثرات الصوتية في المشهد:

حال استيرادك لمواد المشروع ستجد ملف باسم Sounds وهو يحتوي على العديد من الاصوات و منها موسيقى الخلفية Background قم بالتحديد عليها وستجد قائمة بالإعدادات، منها اجعل الخيار **Load Type:** Compressed In Memory هذا الخيار سيقوم باضافة المقطع الصوتي الى الذاكرة و من ثم فك ضغطة حال تشغيله و غالباً يستعمل للمقاطع الطويلة، هناك نوعين آخرين و هما: Decompress On Load بحيث سيتم فك ضغط المقطع حال تحميلة اما الخيار الاخير Streaming يقوم بتحميل المقطع الصوتي من الجهاز مباشرة.

هناك عدة خيارات تجدها في الاعلى و منها هذه الخيارات:

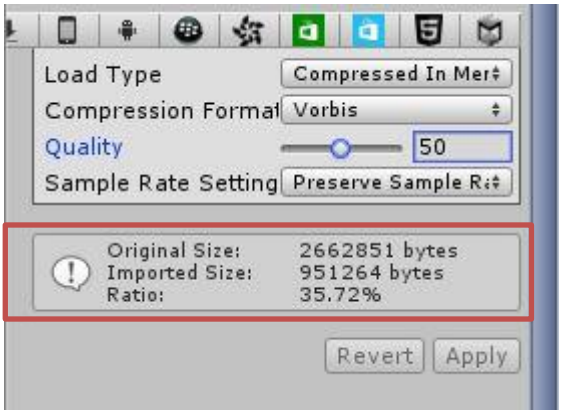




الخيار Force To Mono يقوم بجعل المقاطع ذات اثنين Channel او اكثر الى Channel موحد.

حال تحديدك على المقطع Background قم بعمل Force To Mono لأنه ذو اثنين Channel و سيتم تحويلة الى قناة وحدة او مقطع موحد.

بالنسبة لصيغ الضغط Compression Format هناك 3 انواع متاحة لضغط هذه الاصوات، فالصيغة PCM تقوم بضغط المقطع الصوتي حسب حجمها فالمقاطع الصغيرة مناسبة لهذا الخيار، اما Vorbis فهو يضعف جودة المقطع مع الحفاظ على حجم صغير وهو يوفر شريط يمكنك من خلاله تحديد قوة الضغط وفي اغلب الاحيان يستخدم لموسيقى الخلفية.



لاحظ الاشعار في المربع الاحمر فهو يمثل الخيارات التي اتخذتها مع مزيد من المعلومات فمثلاً سيتم ضغط المقطع الصوتي الى 90 كيلو بايت بينما حجمة الاصلي 2.6 ميجا.

الخيار الاخير ADPCM فهو يوفر ضغط اقل بكثير بحيث هو مناسب للأصوات التي تستخدم كثيراً في اليونتي مثل اطلاق النار و الانفجارات و صوات الازرار و غيرها من الاصوات التي تستخدم كثيراً.

اذن قم بضبط اعدادات موسيقى الخلفية كما هو موضح في الصورة، بينما باقي الاصوات قم بتغيير اعداداتها الى:

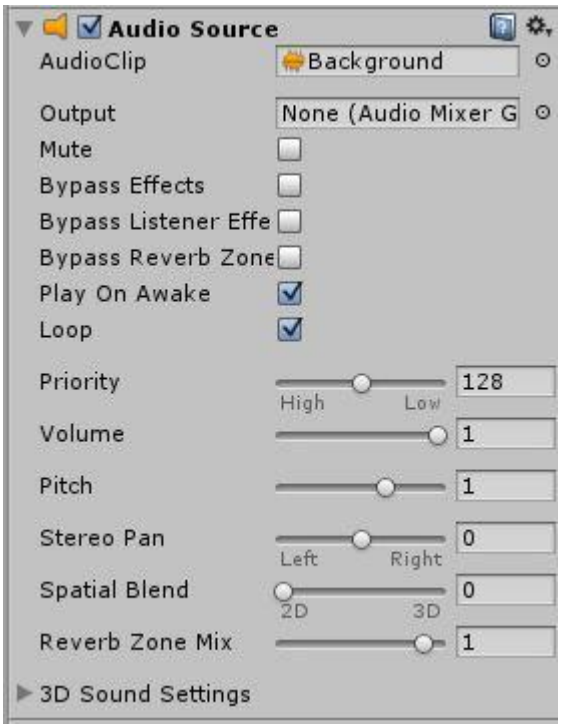
- الاصوات (Click Down, ExplCol, Shoot, switch) ستجد انهم مقطع دو قناتين لهذا قم بتفعيل الخيار Force To Mono.
- جميع الاصوات ماعدا موسيقى الخلفية وExplCol اجعل نوع ضغطهم **Load Type:** Compressed In Memory، ايضاً حول الخيار **Compression Format:** ADPCM
- اخيراً المقطع ExplCol اضبط نفس اعدادات باقي الاصوات ماعدا الخيار **Compression Format:** PCM.

اذن بهذا الشكل نكون قد جهزنا الاصوات لعملية الضغط و ثم فك الضغط حال تحميل المرحلة، تبقى لنا وضع الاصوات في المشهد، ففي هذه الحالة سنقوم بأنشاء GameObject فارغ بحيث يمثل جميع الاصوات و سنجعل له ابناء بحيث ان كل ابن يمثل صوت معين، هذه العملية ستترتب عليها عمل ملف AudioManager و سيحدد لنا هل الاصوات ستضل شغالة في المشهد ام لا و هذا الامر يعتمد على المتغير IsOn في الـ Toggle الموجود في القائمة المصغرة (في الفقرة القادمة)، اذن لنبدأ باضافة المؤثرات الصوتية الى المشهد اولاً.

- انشئ كائن فارغ و اجعلة ابن للكائن Game و سمه Audio.
- بما ان لدينا 9 اصوات فسنحتاج اليها كلها في المشهد، اذن قم بأنشاء 9 كائنات فارغ و اضف لكل كائن مقطع صوتي.

الان هناك اعدادات اولية يجب عملها مثل جعل صوت الخلفية يتكرر ايضاً التعديل على حدة الصوت pitch و التي من خلالها نستطيع عمل تنوع للصوت و سنستخدمه في صوت اطلاق النار الخاص باللاعب للتمييز بين الطلقة الاولى و الثانية.

حسناً حال اضافتك لمقاطع الصوت في الكائنات ستجد ان كل كائن قد اضاف المكون Audio Source، دعنا نذهب الى صوت الخلفية و نعدل علية، لاحظ الصورة التالي من صوت الخلفية:



هذه الخيارات هي الخيارات الافتراضية، ليس فعلياً و لكن ستجد ان الخيار Loop غير مفعل لهذا قم بتفعيلة وهذا يتيح للصوت تشغيل نفسة مراراً و تكراراً.

Play On Awake سيقوم بتشغيل الصوت حال بدأ تشغيل المشهد و هذا مهم لتشغيل صوت الخلفية لهذا قم بتفعيلة.

بالنسبة للخيار Spatial Blend تلاحظ انه 2D ففي حال قمت بتشغيل المشهد ستجد ان الصوت يعمل في جميع الاماكن اما في حال حولته الى 3D سيعطيك مساحة معينة و يمكنك تحديد مدى الصوت و هذا يستخدم في الالعاب الـ 3D لتحديد اقصى بعد للصوت.

بالنسبة لباقي الخيارات فضعها كما هي فلا حاجة لتغييرها.

بالنسبة للأصوات المتبقية فأن هناك شيء مشترك بينهم مثل الغاء تفعيل الخيار Play On Awake فلا حاجة لتشغيل صوت في بداية اللعب.

لكي لا نتوه في العمل دعني اعطيك الاسماء التي سميت بها الاصوات، لاحظ الصورة:

اذن كل صوت يمثل شيء معين و قمت بتسميتهم لكي اعرف لمن يعود هذا الصوت، حال تسميت اصواتك بنفس الاسماء الموجودة في الصوت قم بتحديد الصوت Player Shoot 1_2 وهذا الصوت يمثل الطلقة الاولى و الثانية للاعب، الان قم بتشغيل المشهد و غير الـ pitch الى 1 ثم اطلق الطلقة الاولى ستلاحظ ظهور الصوت، الان غير الطلقة الثانية و غير الـ pitch الى 2 ستلاحظ اختلاف في الصوت! اذن هاتان القيمتان هما المطلوبتان وسيتم التبديل بينهم برمجياً بحيث ان الطلقة الاولى ستأخذ القيمة 1 بينما الطلقة الثاني ستأخذ القيمة 2.

الاصوات في الاعداء:

بالنسبة للأعداء فلن نحتاج الى ان نضيف كود وضع صوت اطلاق النار و الانفجار في جميع الاعداء فبشكل مباشر في ملف EnemiesInfo سنعرف متغيرين يمثلان اطلاق النار و الانفجار، اذن قم باضافة المتغيرات التالية الى الملف:

```
1.  protected AudioSource SoundFire;
2.  protected AudioSource SoundExpl;
3.  [SerializeField] protected string mySoundFire;
```

قد تساءل لماذا يوجد String واحد لتحديد الاصوات و للعلم انه سيقوم بالبحث عن صوت واحد فقط فماذا بشأن الاخر؟ ان الامر بسيط جداً فهناك صوت يتشاركه الاعداء وهو صوت الانفجار لهذا بشكل مباشر سنقوم بالبحث عن الكائن Explosion اما الطلقات فستختلف في الاصوات، الان في Start اضف الأكواد التالية:

```
SoundExpl = GameObject.Find("Explosion").GetComponent<AudioSource>();

if(!gameObject.CompareTag("Boss"))
            SoundFire = GameObject.Find(mySoundFire).GetComponent<AudioSource>();
```

هنا قد تجد شيء غريب وهو سبب عدم وضعي لصوت اطلاق النار للزعيم، صراحةً الزعيم لن يقوم بشيء عدا توليد الاعداء لهذا لا حاجة لوضع صوت لذلك.

في الدالة EnemyShoot يجب ان نقوم بتشغيل الصوت حال اطلاق النار لهذا قم باضافة الكود التالي:

```
if(!gameObject.CompareTag("Boss"))
        SoundFire.Play();
```

هذا الكود ايضاً لن يقوم بتشغيل الصوت في الزعيم، لاحظ ايضاً استخدامي للدالة Play فهي تقوم بتشغيل الصوت و حال عمل الدالة سيتم تشغيل الصوت.

اما في الدالة ApplyHealth بالتحديد في شرط موت الاعداء قم باضافة الأكواد التالية:

```
SoundExpl.Play();
```

لاحظ هنا ان حل موت الاعداء سيتم تطبيق الامر عليهم جميعاً ولن يستثني هذا عدو معين كالزعيم فهو ايضاً سيكون له صوت تأثير الانفجار حال موته.

الاصوات في اللاعب:

بالنسبة للاعب فهنا سنحتاج الى صوتين فقط وهو اطلاق النار وهناك صوتين للإطلاق النار سنعرف لكل منهم متغير و علينا التعامل مع الـ pitch الخاص بصوت اطلاق النار، اذن قم بتعريف المتغيرات التالية في ملف PlayerShooting:

```
private AudioSource FasterShoot;
private AudioSource BombShoot;
public string FindSoundShoot;
public string FindBombShoot;
```

لاحظ انه يوجد لدينا متغيران واحد يمثل صوت القنبلة و الثاني لصوت اطلاق النار و String قوم بالبحث عن الاصوات.

في Start يجب ان نبحث عن الاصوات لهذا اضف الأكواد التالية:

```
FasterShoot = GameObject.Find(FindSoundShoot).GetComponent<AudioSource>();
BombShoot = GameObject.Find(FindBombShoot).GetComponent<AudioSource>();
```

الان يجب علينا وضع صوت اطلاق النار في كل شرط يمثل طلقة معينة، مثلاً الطلقة الاولى قم بوضع الشرط التالي مع مراعاه قيمة المتغير pitch:

```
FasterShoot.pitch = 2;
```

بما انه في حالة تبديل الطلقة الاولى سيتم وضع الـ pitch على قيمتة الطبيعية، اما في حالة قمنا بتبديل الطلقة الثانية يجب ان تتغير قيمته بحيث يتغير الصوت، اذن حال في شرط الطلقة الثانية اضف الكود التالي:

```
2.  FasterShoot.pitch = 1;
```

حسناً هنا لدينا صوتين مختلفين للطلقتين الاولى و الثانية.

اما حال اطلاق النار فقم باستخدام الدالة ()Play لتشغيل صوت الاطلاق وهذا يشمل جميع الطلقات ماعدا القنبلة قم باستخدام المتغير الذي يمثل صوت القنبلة فقط.

بالنسبة لجميع الطلقات التي تخص الاعداء و اللاعب قم بتعريف متغير يمثل الصوت و من ثم اجعلة يبحث عنه و حال الاصطدام قم بتشغيله، اما القنابل نفس الشيء و لكن تأكد من ان الصوت هو صوت الانفجار.

اذن العملية بسيطة وغير معقدة فكل مفهومها هو الحصول على الاصوات و من ثم تشغيلها فقط و بهذا الشكل نكون قد هيئنا الاصوات للعمليات القادمة.

القوائم UI Element:

ان قوائم اللعبة عامل اساسي في اي لعبة ومن خلالها تستطيع الوصول الى الاوامر البرمجية دون الحاجة للعودة الى الملف البرمجي، بالنسبة للنسخة القديمة من يونتي (قبل النسخة Unity4.6) كان اليونتي يستخدم عنصر GUI يوفر العديد من العناصر و غالباً يتم وضعة عبر الدالة التي تمثله OnGUI، و لكن من النسخة الجديدة تطور العنصر و اصبح UI و هذا العنصر يوفر لنا العديد من العناصر الفرعية وهو سهل التعامل، في حالة لم تكن عندك اي خبرة فيها فسنناقشها هنا لكي نستطيع عمل قوائم اللعبة بشكل مفهوم.

الـ Canvas

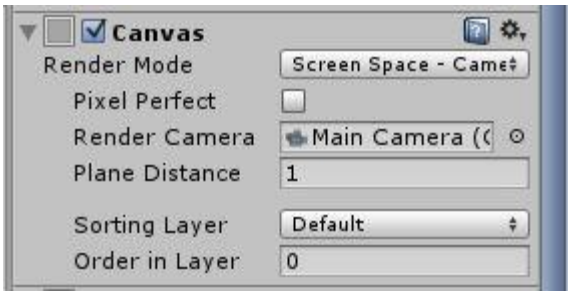
دعنا نناقش عنصر القوائم او مايسمى UI Element ومن خلاله تستطيع عمل اي قائمة تريدها دون استثناء فهو يوفر العديد من العناصر التي تساعدك على التعامل مع محتوى اللعبة و ربطها فكل خيار في اي عنصر يعبر عنة بمتغير يمثله برمجياً.

الـ Canvas وهو القماشة التي من خلالها تضع جميع العناصر ففي حالة اخرجت العناصر من القماشة لن تظهر و الـ Canvas او القماشة توفر لك العديد من الخيارات التي تساعدك على ضبط حجمها و موقعها، لكي تراه بشكل اوضح قم بأنشاء Canvas من UI> Create ستجد العديد من العناصر و منها اختر القماشة، في حالة قمت بأنشائها ستجد ان حجمها غير منطقي في الـ Scene و لكن في القائمة Game تجد انها قد غطت الكاميرا و هذا الشكل الافتراضي لها و تستطيع ضبط موضعها و لكن حالياً ستبقى كما هي.

من الـ Canvas ستجد العديد من المكونات لكن سنناقش الاهم، لاحظ الخيار Render Modes وهو يساعدك على ضبط مساحة القماشة مع الكاميرا و ستجد العديد من الخيارات منها:

- Screen Space – Overlay: يضبط حجم العناصر في حالة تغيرت الشاشة.
- Screen Space – Camera: يضبط حجم العناصر مع حجم الكاميرا.
- World Space: هذا الخيار يتيح للقماشة ان تتصرف كأي كائن آخر وتعتمد على محاورها لكي تتحرك و تغير حجمها و ينصح استخدامه في المنظور الـ3D.

هنا العديد من الخيارات التي تساعدك على التعامل مع الـ Canvas بدقة و عناية، في حال اخترت الخيار Screen Space - Camera ستجد ان موقع القماشة اصبح في الخلف او قد اختفت القماشة ففي هذه الحالة نريد تقديم القماشة او سحبها الى الامام، في الخيار Plane Distance لاحظ ان القيمة الافتراضية له هي 100 و هو بعد القماشة عن الكاميرا فبكل بساطة قم بجعل قيمة 1 و ستجد القماشة امامك.

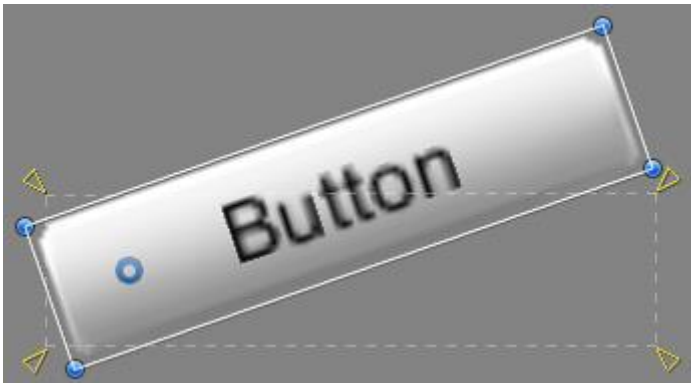


عند تغيير حجم الكائنات دائماً نتطرق الى هذه القائمة  القائمة Rect Tool تتيح لك ايضاً التعامل مع عناصر الـ UI فلاحظ الخيار الاخير على اليسار المحدد وهو يمثل جميع العناصر لغرض التخطيط و تغيير حجم الزوايا فهو لايقتصر على العناصر 2D فقط بل على جميع العناصر و هذا يشمل الـ 3D،

المكون Rect Transform لا ينفك ان تجده في اي كائن، فهو المكون الرئيسي الذي يمثل محاور الكائنات و لكن هذا المكون بالذات موجود في عناصر الـ UI و يختلف عن الـ Transform المعروف في عدة اشياء.

لاحظ القائمة Anchors فهي تساعدك على نقل العناصر الى عدة مواقع داخل الـ Canvas، لاحظ الخيارين Width, Height خيارين جديدين ووظيفتهما التعامل مع طول و عرض العناصر و القيم الحالية تعبر على ان هذا الكائن هو مستطيل، اما في الاخير تجد الخيار Rotation و Scale و تستطيع ان تتحكم بحجمهم، دائماً قيم الطول و العرض لا تعيد لنا الحجم الحقيقي في حالة تغير الشاشة و لهذا يجب ضبط موقع القماشة و جعلها تتناسب مع الشاشة بشكل جيد حتى في حالات تغير الحجم.

نقطة الارتكاز، ان نقطة الارتكاز مهمة جداً ففي حالها لم تعرفها فلاحظ هذه الدائرة الزرقاء في الصورة التالية وهي تمثل نقطة الارتكاز:



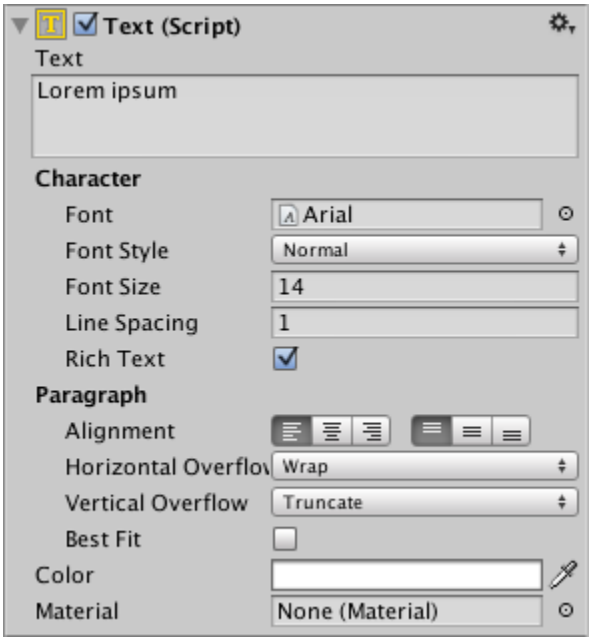
ان ميلان الزر على الجانب سببه هو نقطة الارتكاز، انظر الى الدائرة الزرقاء التي تكلمت عنها، و لكن لن تستطيع تغيير موقعها الا في حالة قمت بإتاحة تغيير موقع نقطة الارتكاز من القائمة التالية:

القيم الافتراضية التي ستجدها Center, Global فبشكل مباشر قم بتغييرها الى هذه القيم، لاحظ ان نقطة الارتكاز متاحة للتلاعب بها عن طريق تحديد الخيار Pivot كما هو موضح و من خلالها تكون نقطة الارتكاز متاحة للتلاعب بها.

المكونات البصرية Visual Components:

مع اضافة العنصر UI تم اضافة العديد من المكونات البصرية مثلا الازرار و عناصر الكتاب و السلايدر و اشياء كثيرة اخرى يمكنك ان تجدها في القائمة UI.

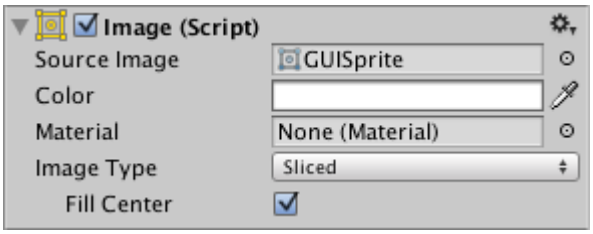
:Text



العنصر Text و الذي يتيح لك كتابة النصوص وهو يوفر العديد من الخصائص لكي تضيفها الى النص مثل تغيير الخط و حجمة ايضاً موقع الخط و محاذاته افقي و عمودي في حدوده، تغيير اللون و عمل ماتريل في حالة اردت و ستجد العديد من الخيارات و تستطيع تجربتها لتوسيع معرفتك فيه فهو بسيط جداً.

:Image

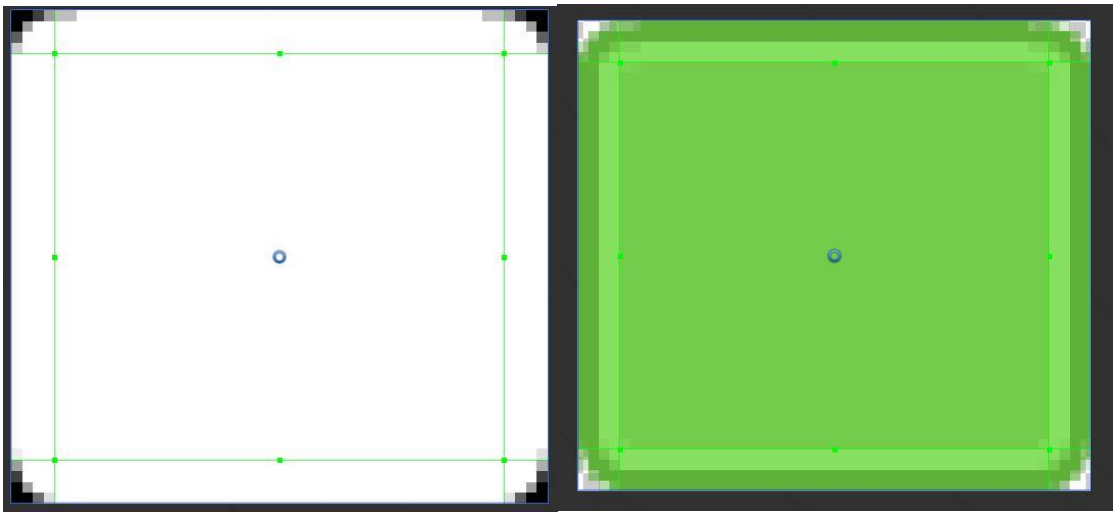
هذا العنصر خاص بوضع الصور في المشهد وهو كثير الاستخدام ففي حالة اضفته الى المشهد ستجد المكون التالي الذي يمثله:



ان هذا المكون هو المكون الرئيسي في العنصر و تجد فيه العديد من الخصائص و منها تغيير الصور، اللون ايضاً، اضافة ماتريل و تحديد نوع الصورة و هذا الخيار مهم جداً و تتفرع منة 4 خيارات و هي:

- Simple: يضع الصور على شكلها الطبيعي.
- Sliced: يتيح لك تغيير الحجم دون تشوه زوايا الصورة.
- Tiled: هذا الخيار يقوم بتكرار الصورة بدل من تمديدها، ففي حالة التمديد يجب تقسيم الصورة و سنتطرق اليه لاحقاً.
- Filled: ليس له وصف معين لكن في حالة قمت باختياره ستجد انه يعطيك بعض الخصائص مثل التعبئة و التناقص على العديد من الاشكل و منها عن طريق 360 درجة و اترك لك تجربة الخيارات.

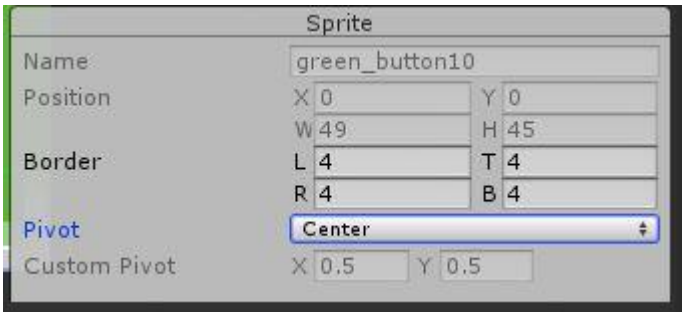
بالنسبة لتقسيم الصورة وهو امر مهم جداً لكي لا تتشوه زوايا الصورة في حالة قمت بتغيير حجمها يكون عبر عمل Sprite Editor و من خلاله تظهر لك قائمة التشريح او التقسيم، لاحظ هذه الصورة:



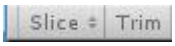
ان الخطوط الخضراء تمثل الاماكن التي لن تتغير حجمها فبشكل مباشر تقوم بسحب الخطوط لتقسيم الصورة بينما الجزء الغير مقسم يسمى Fill و تستطيع تفريغه من الخيار السابق Fill Center، هناك العديد من الخيارات التي توفها لك القائمة و اذكر منها خيار التباين و تجده في الاعلى، لاحظ هذه الصورة التي تشمل خيارات التباين و الحجم:

الصورة الملونة هي خيار التباين ففي حالة قمت بالضغط عليها ستتحول الصورة باللون الابيض و الاسود و هذا قد يساعدك على قسيم الصورة في حالة عملت على صورة فيها العديد من الأوان، تجد ايضاً ازرار تطبيق التغيرات و اعادتها، التحجيم و اخيراً الوضوح.

تغيير نقطة الارتكاز و تقسيم الشكل اما ان يتم يدوياً او عن طريق القائمة Sprite التي تجدها في هذه الصورة:



فمنها تستطيع تقسيم الشكل و تغيير موقع نقطة الارتكاز من الخيار Pivot و تجد فيها العديد من الخيارات، و هنا العديد من الخيارات التي تتحكم بتقسيم الصورة على عدة طرق، لكن حالياً لن تستطيع التعامل مع الخيار الموجود في الاعلى على اليمين:



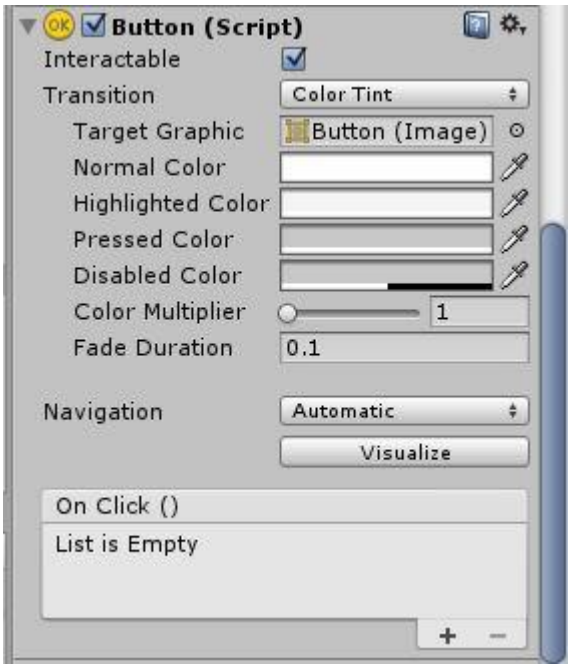
فأولاً يجب ان تكون نوع الـ Sprite هو Multiple و حل العودة الى التقسيم ستجد القائمة متاحة و لك امر مراجعتها لأنها غير مهمة حالياً.

مكونات التفاعل Interaction Components:

ان مكونات التفاعل تختلف عن المكونات المرئية في عملها فهي تتيح لك الاتصال بها عبر اللمس او الضغط و غيرها من الاشياء و ايضاً ربطها و تستطيع اعادة تشغيل الاوامر البرمجية من خلالها و هي تحتوي على العديد من الكائنات التي تتفرع منها و سنمر عليها الان.

:Button

هذا المكون لديه وظيفة وهو تشغيل العديد من الاشياء حال الضغط عليها اي يتم الحدث عن طريق الضغط علية، حسناً دعنا نراجع هذا المكون بشكل سريع:



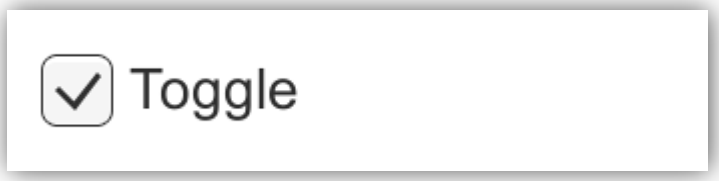
لاحظ العديد من الخيارات في هذا المكون مثل:

- Interactable: حال الغاء تفعيل هذا المكون فلن يقبل الزر الادخال.
- Transition: وهذا الخيار يحدد استجابة الزر حال الضغط و افتراضياً عن طريق اللون سيتغير حال الضغط و هناك العديد من الخيارات مثل وضع اسبريت معين او انميشن او الخيار none سيجعل الزر عبارة عن صورة دون الاستجابة لأي تأثير خارجي (تأثير اللمس او الضغط).
- Navigation: هذا الخيار مهم جداً فهو يحدد نوع التصور وربط الوظائف و هو يوفر العديد من الخيارات لطريقة توصيل التصوير وهم:
  - none: لا يتيح الانتقال عبر الكيبورد.
  - Horizontal: الانتقال الافقي.
  - Vertical: الانتقال العمودي.
  - Automatic: الانتقال التلقائي (جميع الاتجاهات).
  - Explicit : يحدد اماكن الانتقال.
  - Visualize: التمثيل المرئي و الذي يسمح لك برؤية الانتقالات.

في حالة لم يتضح لك الامر فيمكنك مراجعته من مستندات اليونتي: http://docs.unity3d.com/Manual/script-SelectableNavigation.html

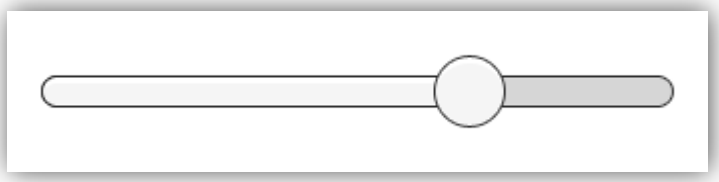
اخيراً قائمة الاحداث و التي من خلالها نستطيع عمل حدث معين اثناء الضغط على الزر، حالياً لن نناقش هذه القائمة و سنتركها الى ان نقوم بتصميم القوائم.

:Toggle



هذا العنصر يساعدك على التبديل بين القيمتين true, false وهو يعتمد على الزر IsOn و يوجد متغير يمثل هذا الزر، شكلياً هو لا يختلف عن باقي العناصر في الصفات عدا انه قوم بالتبديل بين القيمتين فلو راجعت مكونة لرأيت انه سهل التعامل، لاحقاً سنستخدمه لتفعيل و ايقاف الاصوات في اللعبة.

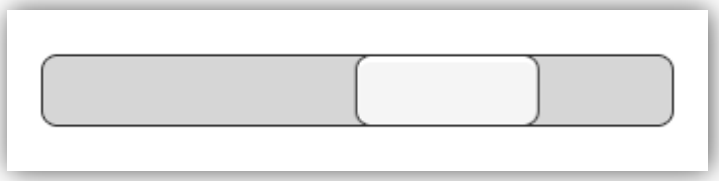
:Slider



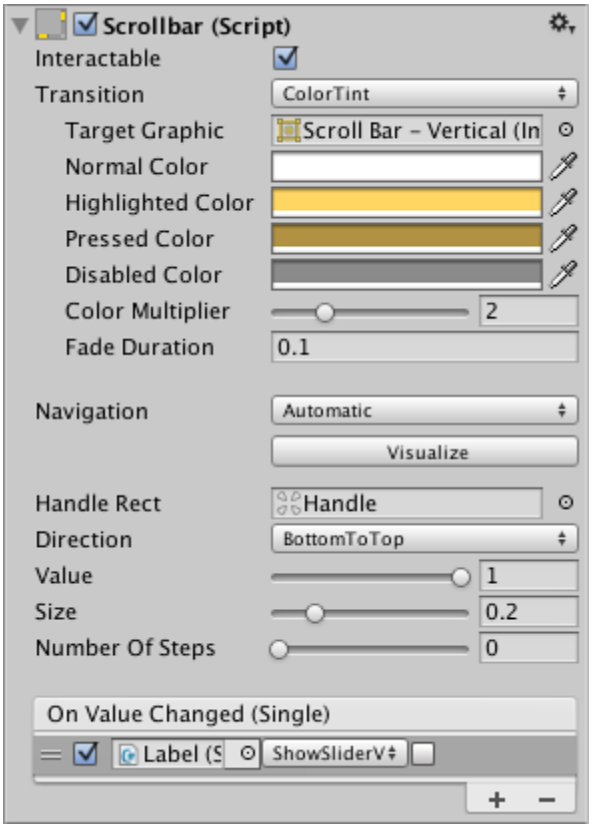
بالنسبة للسلايدر قد تحدثنا عنة سابقاً لكن سأعيد الحديث عنه بشرح اشياء جديدة، السلايدر هو عبارة عن شريط انزلاق وهو يعتمد على الخيار Valus الذي يتم تحديد قيمة من المتغيرين Min Value و Max Value.

افتراضياً اتجاه السلايدر هو Left to Right و من الخيار Direction تستطيع تغيير اتجاهه و هناك 4 خيارات يمكنك تجربتها. اما باقي الخيارات فهي متشابهة مع سابقتها بخلال القليل منها و الذي لن نحتاجه في اللعبة.

:Scrollbar



بالنسبة لشريط التمرير فهذا غالباً يستخدم في الـ Text اي مثلاً لقراءة اتفاقية او مراجعة ويمكن تمريرة بين القيمتين 0 ,1 فلو عدنا الى المكون ستجد العديد من الخصائص و لكن اهمها اي القيم في الاخير و التي تتعامل مع تمرير هذا الشريط وعدد الخطوات وحجم مقبض التمرير، لاحظ الصورة:



لاحظ ان الخيارات الاخيرة هي فقط الشيء الجديد في هذا المكون، الخيار Value و التي تتحكم بتمرير الشريط، اما الخيار Size يقوم بتحديد حجم مقبض التمرير بينما الخيار الاخير يتحكم بعدد الخطوات من و الى النهاية و افتراضياً هي 0 اي ليس هناك اي خطوة و سيتحرك بشكل سلس.

:Input Field

زر الادخال مفهومه سهل جداً ووظيفته هي الكتابة فقط و لكن تجد فيه العديد من المكونات الجديدة مثل الكتابة، عدد الاحرف، نوع النص المدخل (اسم، ايميل، كلمة مرور، pin ...الخ) لون خلفية النص عند التحديد علية اول مرة و خيار تحديد هل سيظهر على الهواتف ام لا، و هناك خيارين للأحداث و يمكنك مراجعة العنصر من هذا الرابط: http://docs.unity3d.com/Manual/script-InputField.html.

ي لعبتنا لن نستخدم هذا العنصر فلو اردت مراجعته فبكل بساطة راجعة من الموقع و ستجد انه سهل جداً.

قوائم لعبتنا:

في البداية سنناقش فكرة انشاء القائمة التي تضم الاسكورات، الصحة و الطلقات ان هذه القائمة مهم جداً فمن خلالها نستطيع التعرف على الاشياء الغير ظاهره او بمعنى آخر الموجودة برمجياً فقط، ان هذه القائمة بسيطة جداً ويجب علينا عمل قائمة خرى تختص بالتحكم بجميع الاصوات في اللعبة، اعادة اللعبة و العودة الى القائمة الرئيسية اي الخروج من المستوى و ليس من اللعبة بشكل كامل، اذن سنقسم هذه الفقرة الى عدة اقسام فرعية لكي نناقش كل شيء بالتفصيل.

قائمة العرض:

ان قائمة العرض او شريط العرض هو الشيء الاولي الذي يجب عملة و دعني اعطيك صورة توضح ماهي هذه القائمة:

كما تلاحظ ان القائمة بسيطة جداً فكل ماتتضمنة هو النقاط، الذخيرة و الصحة وهي تعتمد على Slider، اذن لنقم بأنشائها، اتبع الخطوات التالية:

- انشئ كائن فارغ وهو سيمثل القوائم لهذا قم بتسميته UI.
- انشئ Canvas و تأكد من ان **Render Mode:** Screen Space – Camera اما الخيار Plane Distance اجعلة 1، في المكون Canvas Scaler اجعل الخيار **UI Scale Mode:** Scale With Screen Size اخيراً ضع الشريط **Match** = 0.5 وسمه Header.
- منه اختر العنصر Image و سيتوجب علينا اضافة صورة سندعها الى القادم اما الان تأكد من ان و تأكد من ان موقع من القائمة Anchors كما هو موضح في المربع الاحمر (اضغط على Alt ثم الخيار المحدد):

- منة قم بأنشاء Text 2 وضع الاول على اليمين و الاخر في الوسط، و سمي الاول Score و الاخر Ammo .
- اخيراً قم بأنشاء Slider و ضعة على الطرق الايسر ومنه قم بأنشاء Text و سمة 100% وهو يمثل قيمة الحياة و لكن تأكد من ان الخيار Value اكبر قيمة له هي 100 في Slider.

من هذه النقاط سيتكون لنا الشريط في الاعلى و لكن ستتبقى لنا الصورة لهذا قم بالبحث عن الصورة green_button10 و قم بتقسيم الاطراف كما هو موضح:

شيء اخير وهو عمل Shadow للـ Text لكل في كل Text قم بالبحث عن المكون Shadow و ستجده، اضع المحور y =2 و المحور x = 0 و اخيراً للون اجعلة ابيض و هذا سيجعل الخط يظهر بشكل جميل.

بالنسبة للـ Slider قم بحذف الـ Handle Slide Area فقط اما الكائن Fill لونة الى الاحمر، بالنسبة طول و عرض السلايدر فقم بتغييرة من القائمة Scale افضل.

الان بهذا الشكل نكون قد انهينا تصميم قائمة العرض ووظيفتها عرض الصحة، النقاط و الذخيرة و سيتبقى لنا برمجتها و هذا الامر سيعتمد على ربط العناصر في الملفات البرمجية التي تمثلها، لاحظ ان الصحة في الملف PlayerController بينما الذخيرة في PlayerShooting و اخيراً النقاط في ملف GameManager. اذن لنقم ببرمجتهم.

- في الملف PlayerController قم بتعريف هذه المتغير و لكن تأكد من استيراد نطاق الاسم using UnityEngine.UI; :

```
1. public Slider healthSlider;
2. public Text healthValuse;
```

اذن متغير يمثل السلايدر (شريط الصحة) و متغير يمثل الرقم 100% و هذا الشيء اختياري.

- في الدالة Update اضف السطرين:

```
1. healthSlider.value = Health;
2. healthValuse.text = Health + "%";
```

لاحظ ان قيمة السلايدر مربوطة بقيمة الصحة، اخيراً المتغير سيتم وضع قيمة الصحة في المتغير Text بكل بساطة.

بالنسبة للذخيرة هنا سنعود الى الملف PlayerShooting يجب علينا وضع قيمة الذخيرة التي سنستخدمها في متغير من نوع Text.

- عرف المتغيرات التالية في الملف:

```
1. public Text textAmmo;
```

حسناً هذا المتغير سيهتم بإظهار عدد الطلقات المتبقية.

- الان في كل شرط يمثل ذخيرة معينة قم باضافة السطر التالي مع تغيير المتغير Ammo الى مايتناسب مع نوع الطلقة:

```
1. textAmmo.text = "Ammo: " + Ammo_1;
```

الان في حالة قمت بتغيير الطلقة سيتم عرض الذخيرة المتبقية لها.

اخيراً في الملف GameManager علينا تعريف متغير من نوع Text يمثل النقاط.

- قم بتعريف المتغير التالي:

```
1. public Text textScore;
```

في الدالة UpdateScore اضف السرط التالي:

```
1. textScore.text = "Score: " + Score + "/" + HightScore;
```

لاحظ ان السرط بسيط جداً فهو سيعطينا قيمة النقاط الحالية بالاضافة الى اعلى نقاط، في الانسباكتور قم بملء المتغيرات و حال تشغيل اللعبة ستجد القائمة ظهرت بهذا الشكل:

اذن هذه هي القائمة الاولية و ستجد ان القيم تظهر بدون اي مشكلة و لكن لن نترك القائمة بهذه البساطة بل سنضع لها قائمة مصغرة Min Menu و هي تختص بإيقاف الصوت و تشغيله ايضاً اعادة اللعبة او العودة الى القائمة الرئيسية، اخيراً استخدم الخط kenvector_future.

تصميم القائمة المصغرة Min Menu:

بالنسبة للقائمة المصغرة فاضن انك قد اخدت فكرة عنها لهذا سنبدأ بتصميمها و حال الانتهاء علينا عمل انميشن لها بحيث هذه القائمة تظهر من الاعلى الى الاسفل و حال الاختفاء تتراجع الى الاعلى و هذا يضيف جمالية لظهور القائمة، اذن اتبع الخطوات التالية لعمل القائمة:

- انشئ Canvas و تأكد من ان نوع العرض هو Screen Space – Camera و الـ Plane Distance = 2، و **UI Scale Mod:** Scale With Screen – Size اخيراً سمة Min Menu.
- داخلة اضف العنصر Image ثم ابحث عن الصورة green_button10 و قم بتقسيم اطرافها و من ثم اسحبها الى العنصر وسمة Background.
- داخل الـ Image اضف العنصر Toggle وسمة Sounds و عنصرين من نوع Button سم الاول Restart و الثاني Exit، تأكد من توزيعهم بهذه الطريقة:

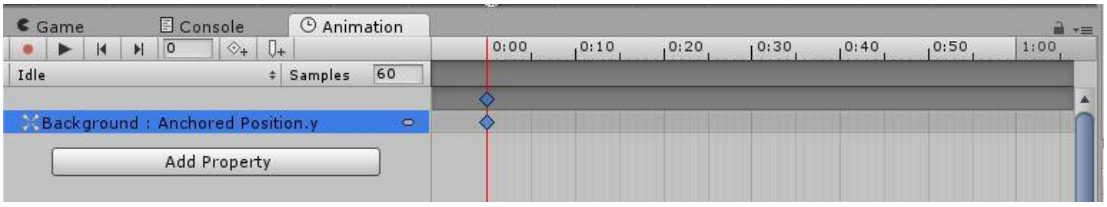
بالنسبة للصور ستجد في ملف UI العديد من الصور و لكن في حال اردت ان تضع نفس هذه الصور فقم بالبحث عن الصورة flatDark32 وهي تمثل زر الخروج، بينما الصورة flatDark20 تمثل زر اعادة تشغيل اللعبة.

بالنسبة للـ Toggle فأننا سنبحث عن صورتين تمثلان تشغيل الاصوات و ايقافها، بالنسبة لصورة تشغيل الوقت سنضعها في الكائن Checkmark وهو ابن للكائن Background في Toggle اذن ابحث عن الصورة flatDark16 اما بالنسبة للصورة التي تمثل ايقاف الاصوات سنضعها في الكائن Background وهي الصورة flatDark18 و بهذا الشكل نكون قد قمنا بترتيب القائمة ففي حال قمت بتشغيل اللعبة و ضغطت على الازرار ستلاحظ انها تعمل بشكل جيد و هذا يشمل الـ Toggle ايضاً.

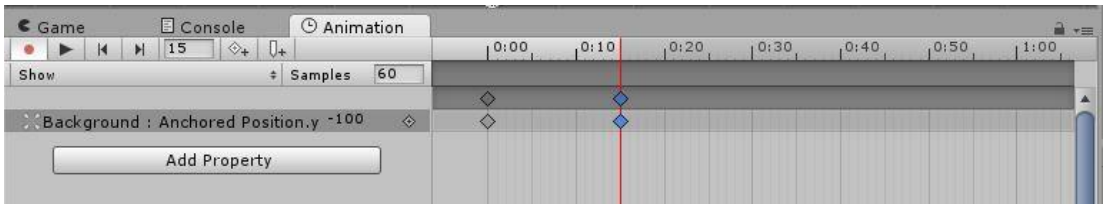
هذا هو ترتيب القائمة من الهايروكي، تبقى لنا عمل الانميشن وهم 3 انواع: اظهار القائمة، اخفاء القائمة و انميشن وسيط Idle.

حسناً في ملف Animations انشئ مجلد و سمة Min Menu، و اتبع الخطوات التالية لكي لا تتوه في العمل:

- اضغط على الكائن Background ثم من النافذة Animation اضغط على Create و سمة idle، ثم آخر وسمة Show و اخير سمة Hide سيتبقى لنا اعطاء القيم و انا سأعتمد القيم الموجودة معي.
- في الاول انميشن Idle اجعل موقع القائمة في Pos Y = 80 وهذا يخفي القائمة في الاعلى، هذا شكل الانميشن:



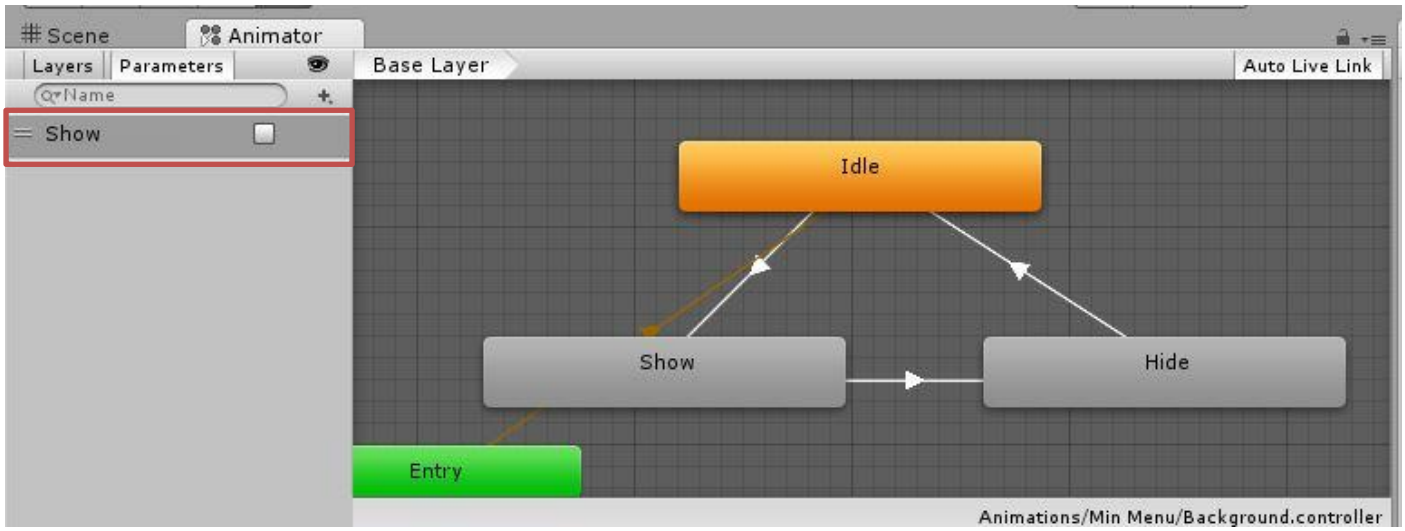
- ثاني انميشن Show يجب ان يظهر القائمة لهذا في الانميشن من الخطوة 0.00 الى 0.10 اجعل موقعة في Pos Y = -100 و ستلاحظ ان القائمة ظهرت في المشهد.



- ثالث انميشن Hide يجب ان يخفي القائمة مجدداً لهذا من الخطوة 0.00 يجب ان يكون موقعة في Pos Y = -100 ثم الى الخطوة 0.10 غير موقعة الى Pos Y = 80 و بهذا الشكل ستختفي القائمة او بالأصح ستعود الى موقعها الاصلي في الاعلى.

حسناً بهذا الشكل سيتبقى لنا ربط الانميشن من القائمة Animator وهي تختص بربط الانميشن وتوفر العديد من الخصائص مثل انشاء Layer جديد لكي تعمل علية، ايضاً بارامترات لكي تستطيع تمريرها او التعامل معها برمجياً بحيث تعطي لكل خطوة قيمة معينة و سيتم تشغيلها.

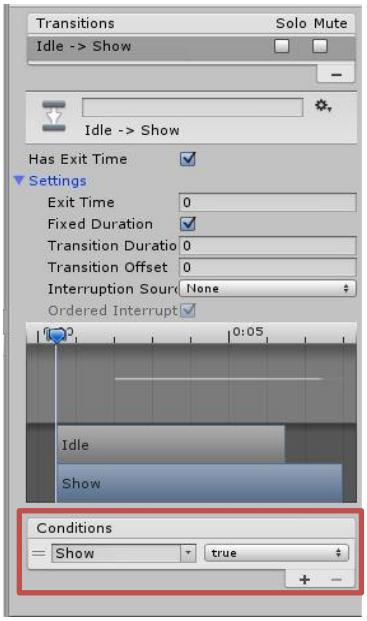
حال فتح النافذة Animator ستجد جميع الانميشن الذين عملناهم للقائمة بينما الانميشن Idle باللون البرتقالي يمثل الانميشن الذي سيبدأ اولاً وهو الوسيط بيت اظهار القائمة و اخفائها، في حال لم يكن اللاير Idle فبالضغط علية بالزر الايمن للماوس ستجد الخيار Set as Layer Default State وسيظهر باللون البرتقالي، لاحظ هذه الصورة بعد ربط الانميشن مع بعضها:



في حال لم تفهم طريقة الربط فلا تقلق سأقوم بشرحها لك، ان الاسهم الموجودة في الصورة و التي وظيفتها هي ربط الانميشن تسمى Make Transition ويمكنك ان تجد هذا الخيار بالضغط على الزر الايمن للماوس في Idle و قم بتوصيلة الى Show ثم من Show الى Hide منة الى Idle كما هو موضح.

الاحظ البارامتر Show فهو يمثل تشغيل نوعين من الانميشن اعتماداً على القيمتين true, false بحيث ان القيمة true ستشغل الانميشن Show بينما false ستشغل Hide و من ثم تلقائياً سيعود الى Idle، لإنشاء هذا البارامتر اضغط على اشارة + في الاعلى و ستجد عدة انواع قم باختيار Bool و سمة Show.

الان بالضغط على السهم الموصل من Idle -> Show ستجد القائمة التالية:



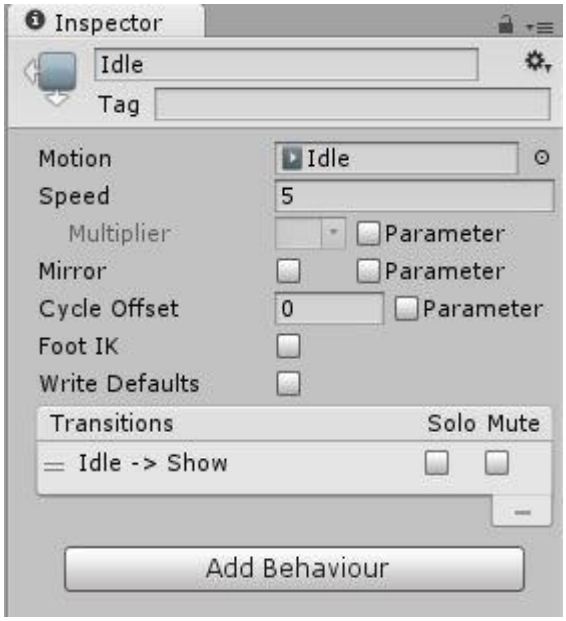
ان هذه القائمة توفر لك العديد من الخصائص كربط الانميشن و تحديد سرعة ايضاً وقت التشغيل و الانسيابية و اشياء اخرى، في القائمة الاخيرة Condition الشروط تلاحظ انني وضعت البارامتر Show ففي حال كانت قيمتة true سيتم تشغيل هذا الانميشن.

ركز على خيار الاعدادات Settings وهو يطلب منك تحديد وقت الخروج من الانميشن اجعلة صفر اما باقي الخيارات فاضبطها كما هو موضح.

اخيراً تأكد من ان جميع الانميشن لا تعمل Loop Time او قم بإلغاء تفعيل الخيار Loop Time.

الان بالضغط على السهم الموصل من Show -> Hide ستجد نفس القيم او القيم الافتراضي فكل ماستقوم بفعلة هو تغيير قيمة Show الى false اما باقي الخيارات فأضبطها كما هو موضح.

ان هذه العملية ستساعدنا على الانتقال بين الانميشن بسرعة دون الحاجة لانتظار وقت تشغيل اي سيتم تشغيله بشكل مباشر، حال ضغطت على الانميشن Idle ستظهر لك هذه القائمة:

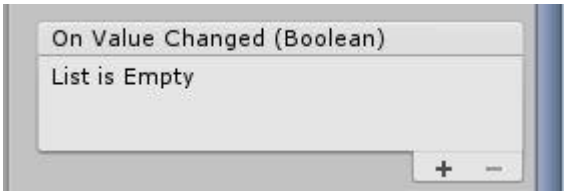


لاحظ الخيار Speed افتراضياً هو 1 و السرعة الطبيعية بطيئة لهذا قم بجعلة 5 او مايتناسب معك و عند تشغيله في اللعبة لاحقاً ستجد انه يظهر بشكل سريع.

ان العملية البرمجية لهذا الانميشن ستكون في ملف GameManager وهو الملف الذي سيتضمن مكونات المشهد و من ضمنها القائمة المصغرة فيجب علينا اظهرا القائمة بالضغط على المفتاح Esc و يجب علينا وضع الشروط و التأكد من قيمة المتغير timescale السبب اننا سنعتمد على تشغيل الانميشن في حالة توقف الوقت فقط و حال هذا المتغير حساسة فأننا استخدمناه سابقاً اكثر من مرة لهذا يجب علينا أحاطته بعد شروط لكي لا نخرج عن نطاق عملنا او لا يتشعب الامر اكثر.

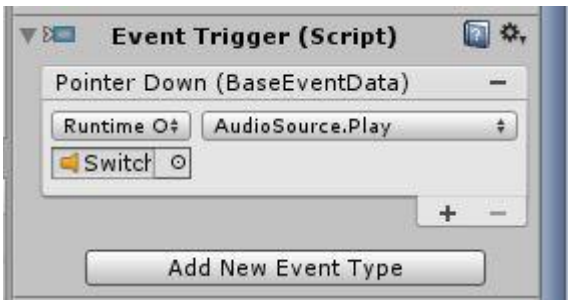
بالعودة الى الكائن Background الذي يمثل القائمة ستجد المكون Animator ومن المؤكد انك رايك الخيار Update Mode ان هذا الخيار يتيح لك تشغيل الانميشن في عدة حالات و الخيار الافتراضي هو normal يعمل مع الوقت لهذا قم باختيار Unscaled Time وسيتيح لنا تشغيله عند توقف الوقت.

تبقى لنا وضع الاصوات في الازرار وهو عن طريق اضافة الأحداث، قم بفتح الـ Toggle وستجد هذا الخيار:



هذه القائمة تتيح لك عمل احداث معينة تحصل حال الضغط على هذا الزر، حالياً نعمل على الـ Toggle ففي حالة قمنا باضافة الصوت في هذه القائمة سنواجه مشكلة سماع الصوت لاحقاً حال تغير قيمة IsOn لهذا يجب علينا استخدام المكون Event فهو يوفر العديد من الاحداث من المرور غلى الزر، الضغط الخروج منه و اشياء اخرى يمكنك مراجعتها، اذن قم بالبحث عن المكون Event وحال أضافته ستظهر هذه القائم:

ان المفتاح Add New Event Type يتيح لك اضافة نوع الحدث الذي تريده ففي حال ضغطت علية ستجد العديد من الاحداث ومنها المذكورة، منها اختار النوع Pointer Down هذا الحدث يتيح لك تطبيق شيء معين حال الضغط على الزر، ستظهر لك هذه القائمة:



فعالياً لن تظهر كما هو موضح لهذا حال الضغط على الزر + ستظهر لك خانة فاضية لها قم بسحب الصوت Switch اليها و ستجد ان القائمة تطلب منك تحديد حدث معين لهذا قم بالضغط عليها و ستظهر لك قائمة وفيها المكونات الرئيسية مثل Transform, GameObject و اخيراً المكون Audio Source حال الضغط علية ستجد قائمة طويلة بالمتغيرات فقم بالبحث عن الدالة Play() و اخترها، ففي حال الضغط على ازر سيتم تشغيل الصوت.

الان قم بنسخ المكون بعمل Copy Component و اللصقة في جميع الازرار في القائمة و لكن قم بتغيير الصوت الى Click Down، و بهذا الشكل نكون قد هيئنا الازرار لعملية البرمجة.

برمجة الـ Min Menu:

في الملف GameManager علينا تعريف متغيرات اولية تمثل المكون Animator و نوع العرض و متغير Bool ضروري جداً لكي نتأكد من ان يعمل لكي تعمل القائمة فقط، اذن عرف هذه المتغيرات:

```
1.  public enum TypeMenu{
2.      Show,
3.      Hide,
4.  }
5.
6.  [Header("Min Menu")]
7.  public TypeMenu typeMenu;
8.  private bool IsMinMenu;
9.  public Animator anim;
```

لاحظ الـ enum يمثل حالتين للكائن و سيتم التبديل بينهم و التحقق منهم، في القائمة Min Menu تجد المتغير bool و هو الذي سيجدد هل القائمة ستعمل ام لا، اخيراً متغير يمثل الانميتور لكي نستطيع تشغيله و إيقافه.

اذن قم بإنشاء دالة جديدة وسمها MinMenu و ضف الشرط التالي:

```
1.  if(typeMenu == TypeMenu.Show || typeMenu == TypeMenu.Hide)
2.  {
3.      IsMinMenu = true;
4.  }else IsMinMenu = false;
```

بدايةً علينا التحقق من حالة الكائن و في كلا الحالتين سيتم تشغيل المتغير IsMinMenu غير هذا سيتم توقيفه و بالنسبة لتوقيفة هو عند العمل على قائمة النجوم سنناقشها لاحقاً.
حالياً علينا توقف الوقت و لكن ليس بهذا الشكل، اضف الشرط التالي لكي نناقشه:

```
1.  if(Input.GetKeyDown(KeyCode.Escape) && Time.timeScale == 1 && IsMinMenu)
2.  {
3.      Time.timeScale = 0;
4.      typeMenu = TypeMenu.Show;
5.      return; }
6.  if(Input.GetKeyDown(KeyCode.Escape) && Time.timeScale == 0 && IsMinMenu)
7.  {
8.      Time.timeScale = 1;
9.      typeMenu = TypeMenu.Hide;
10.     return;
11. }
```

الان قد لاحظت ان الشرط منسوخ مرتين بحيث ان الاول سيتحقق من الضغط على المفتاح Escape و من حالة الوقت و من المتغير IsMinMenu و في حالة تحقق الشرط سيتم ايقاف الوقت لان الشرط لن يعمل الا في حالة كان الوقت يعمل و سيتم تشغيل نوع العرض من Hide الى Show اخيراً سيتم اخراجنا من الشرط.

الشرط الثاني نفس الشرط الاول و لكن في حالة كان الوقت لا يعمل سيتم تشغيل الوقت و سيتغير نوع العرض الى Hide و اخيراً نخرج من الشرط.

الان بما ان قيمة المتغير timescale يتغير في الشرطين لهذا يجب علينا احاطتها بشرط اخير وهو التأكد من انه تم التحقق من نوع العرض و سيتم تطبيق العرض الخاص به، اضف الشرطين الاخيرين:

```
if(Time.timeScale == 0)// Show Menu
{
        if(typeMenu == TypeMenu.Show)
        {
                anim.SetBool("Show", true);
                PlayerShooting.ApplyShooting = false;
                PlayerController.ApplyMove = false;
                WheelController.MoveWheel = false;
        }
}

if(Time.timeScale == 1)// Hide Menu
{
        if(typeMenu == TypeMenu.Hide)
        {
                anim.SetBool("Show", false);
                PlayerShooting.ApplyShooting = true;
                PlayerController.ApplyMove = true;
                WheelController.MoveWheel = true;
        }
}
```

الان هذه العملية مهم جداً، ان الشروط الاول يفسر في تعليق وهو تشغيل القائمة و سيتم التأكد من قيمة الوقت و نوع العرض، لاحظ ان النوع الاول Show سيقوم بتشغيل البارامتر Show و لاحظ استخدامي للمتغير anim و كما ربطناه سابقاً في بارامتر من نوع bool نقوم باستخدام الدالة SetBool ومنها نضع القيمة التي ستعمل و المربوط بها الانميشن بحيث ان true تعبر عن تشغيل انميشن Show حال تحقق الشرط سيتم تطبيق الامر و اخيراً ايقاف اطلاق النار، الحركة و الدوران.

الشرط الثاني هو نفسة الاول و لكن سيتم تشغيل الانميشن المربوط بالقيمة false وهو Hide بحيث تختفي القائمة و تعود الحركة، اطلاق النار و الدوران الى حالتهم الاصلية.

ان العملية بسيطة جداً فمن بداية كتابة الشروط وهي عبارة عن استنساخات و تغيير قيم فقط، اخيراً قم بمناداة الدالة في Update و من القائمة انسباكتور عبئ المتغيرات.

تصميم قائمة النجوم Menu Stars:

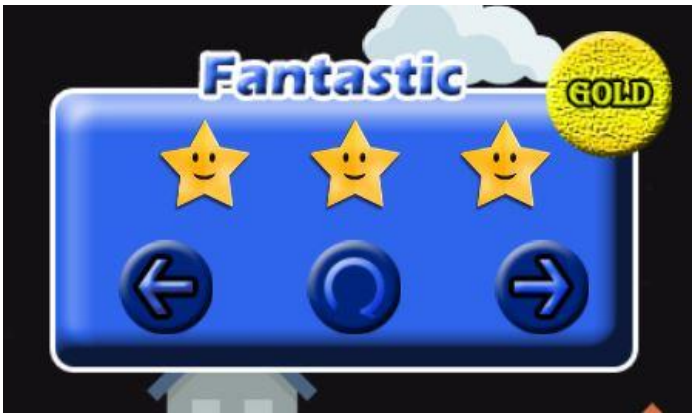
ان قائمة النجوم مهمة جداً فهي التي ستحدد لك نسبة لعبك للمرحلة و فوزك او خسارتك قائمة النجوم تختلف في عملها في كل لعبة، ففي لعبتنا ستعتمد القائمة على فوزك بالمرحلة بالاضافة الى نسبة صحتك لكي تظهر لك قائمة النجوم المناسبة، اما في حالة خسرت ستظهر لك قائمة الخسارة المعروفة.

تصميم القائمة لن يختلف كثيراً عن اي قائمة لهذا سنقوم بتصميمها بشكل عادي و عملي انميشن لظهورها من الاعلى الى الاسفل، اذن لنقم بتصميم اول قائمة و من ثم نستنسخها ونعدل على بعض الاشياء فيها لكي ننشئ باقي القوائم.

اذن العملية ستكون كالاتي:

- انشئ كائن فارغ باسم MenuStars و اجعلة ابن للكائن UI و منة انشئ Canvas و جعلة ابن للكائن MenuStars، تأكد من ان اعدادات الـ Canvas تتناسب مع الكاميرا.
- انشئ Image و اجعل خلفيته هي الصورة Background-Gold بحيث انها تمثل القائمة الذهبية.
- داخلة اضف اثنين GameObject فواحد سيحوي الازرار وواحد سيحوي النجوم بحيث ان اسم الاول Buttons و الثاني Stars.
- في الكائن Buttons اضف العنصر Button و اجعل صورته هي Back Blue وهذا هو زر الخروج من المرحلة و ستجد ان للصورة شبيه باللون الاحمر، سنستخدمها لاحقاً.
- الان قم بنسخ الزر 3 مرات بحيث ان كل زر يكون في محل معين اي (يمين، يسار ووسط) بحيث ان الزر في اليسار هو الخروج من المرحلة، بينما الوسط هو زر اعداده اللعب وصوته هي Restart-Blue و الزر على اليمين زر الانتقال الى المرحلة التالية و صورته هي Next Blue و بهذا الشكل نكون قد جهزنا الازرار.
- بالنسبة لنجوم فبشكل مباشر قم بعمل Image و اجعلة ابن للكائن Stars و اضف الية الصورة Star و انسخها ثلاث مرات و هي لقائمة الثلاث نجوم.

اذن شكل القائمة النهائي سيكون:



تبقى لنا عمل انميشن للقائمة وسأترك الامر لك فتحريك القائمة من الاعلى الى الاسفل تحدثنا عنة سابقاً ولكن سأعطيك شريح مبسط عن طريقة انزال القائمة.

ان العملية بسيطة جداً ففي البداية ستكون القائمة مخفية في الاعلى ثم تنزل الى الاسفل فقط فلا حاجة لعمل انميشن يعيدها الى الاعلى.

بالنسبة للأزرار فأما ان تضيف اليها انميشن او ان تختار الخيار Sprite Swap ففي كلا الحالتين تستطيع اضافة الزر باللون الاحمر الذي يعبر عن كل زر.

بالنسبة للنجوم فأمرها سهل، ان كل ماستقوم بفعلة هو عمل انميشن بسيط يظهر الزر اي ان الزر سيكون مخفي اي ان القائمة Scale ستكون جميع قيمها صفر، اما في الانميشن ستتحول الى 1 او القيمة المناسبة كما في الصورة.

- بالنسبة للقائمة الثانية فهي نفسها القائمة الاولى و لكن قم بحذف النجمة في الوسط و اخيراً اجعل خلفية القائمة هي Background-Normal.
- قائمة الخسارة بسيطة جداً فكل ماستقوم فعلة هو حذف زر الانتقال الى المرحلة القادمة ايضاً حذف النجوم و كتابة اي شيء مثل You Loser.

هذه هي القوائم الثلاث:

حسناً قبل الولوج الى البرمجة علينا اضافة الاصوات الى الازرار لهذا قم بنسخ نفس المكون الاولي الذي اضفنا الية صوت الازرار و اضفه الى جميع الازرار بالإضافة الى اختيار الحدث Pointer Enter بحيث في حالة لم الماوس الزر سيشغل صوت المرور و هو الصوت Click Trigger يمكنك سحبة و تجربة الامر.

اخيراً تأكد من ان الانميشن يعمل عند توقف الوقت Unscaled Time.

برمجة الـ Menu Stars:

في هذه النقطة سنناقش شيء مهم و هو التعرف على طريقة عمل تسلسل لكلاس نقوم من خلاله باستنساخ نفس الاوامر بشكل سريع دون الحاجة لعمل مصفوفة كبيرة، بشكل واضح سنقوم بعمل Serializable لكلاس علينا انشائه في الملف GameManager و سيتكون اسمة MenuStars و هذا الامر سيوفر علينا عمل ملف جديد ووضعة تحت اوامر GameManager، اذن في الملف GameManager علينا تعريف كلاس و يجب ان يكون public وسيتوجب علينا فيه تعريف متغيرات تمثل (قائمة النجوم، الكائن الذي يحوي الازرار، الكائن الذي يحوي النجوم و وقت الظهور) بحيث اننا نعطي وقت لظهور القائمة وتشغيل الانميشن فلن نعتمد على اخفاء القائمة فقط فقد تواجهنا مشكلة ظهورها اثناء اللعب او اثناء توقيف اللعبة و لهذا علينا ايقاف عمل الانميشن و من ثم تشغيله و اظهار القائمة.

- اولاً قم باضافة هذا الكود وهو يمثل كلاس جديد باسم MenuStars و لاحظ انه يعمل Serializable:

```
1.  [System.Serializable]
2.  public class MenuStars{
3.          public GameObject menuStar;
4.          public GameObject Stars;
5.          public GameObject Buttons;
6.          public float timeShow;
7.  }
```

يجب ان يكون هذا الكلاس خارج الكلاس GameManager و سنقوم بالوصول الى متغيراته عبر الحلقة foreach.

- الان هل تذكر الكلاس TypeMenu هو من نوع enum وفية الحالتين Hide, Show قم باضافة الحالات الاتية:

```
1.          WinGold,
2.          WinNormal,
3.          Loser,
```

لاحظ ان كل حالة تمثل قائمة معينة فسنستخدمها لاحقاً.

- الان يجب علينا تعريف متغيرات مثل الانميشن في القوائم، قائمة تمثل الفئة MenuStars، مصفوفة تمثل القوائم التي ستظهر و التي ستختفي و اخيراً متغير يقوم بتوقف الوقت لكي يعمل الانميشن:

```
1.          [Header("Menu Stars")]
2.          public List<MenuStars> menuStars;
3.          public Animator[] animMenu;
4.          public GameObject[] SelectMenu;
5.          public float timsStop;
```

اذن هذه المتغيرات الضرورية التي ستختص بتشغيل القوائم.

- هناك شيء يجب ان تعلمة ان القائمة لن تظهر الا في حال وصلت النقاط الى اعلى قيمة لها، اذا قم بأنشاء دالة جديدة باسم ShowMenuStars وهي تختص بإظهار قائمة النجوم، اذن اضف الكود التالي:

```
1.  		if(maxScore >= HightScore)
2.  		{
3.  			IsMinMenu = false;
4.  			PlayerShooting.ApplyShooting = false;
5.  			PlayerController.ApplyMove = false;
6.  			WheelController.MoveWheel = false;
7.  
8.  			if(PlayerController.Health >= 70)
9.  				typeMenu = TypeMenu.WinGold;
10. 
11. 			if(PlayerController.Health < 70 && PlayerController.Health > 0)
12. 				typeMenu = TypeMenu.WinNormal;
13. 		}
14. 
15. 		if(PlayerController.Health <= 0)
16. 		{
17. 			IsMinMenu = false;
18. 			typeMenu = TypeMenu.Loser;
19. 		}
20. 
```

حال التمعن في الكود ستجد ان ظهور القوائم يعتمد على صحة اللاعب، حال وصول النقاط الى اقصى قيمة سيتم توقيف كل شيء ماعدا الوقت و هذا الامر سنتركه للقوائم و هي تحدد متى يتوقف الوقت لكي تشغل نفسها، انواع القوائم سيتغير حسب صحة اللاعب و كل نوع سيظهر القائمة التي تخصه، لاحظ النوع الاخير Loser سيعمل حال موت اللاعب بينما سيتم ايقاف القائمة المصغرة في كلا الحالتين (موت اللاعب او فوزة) يمكنك ملاحظ المتغير IsMinMenu في السطرين 3, 17.

- الان يجب علينا الوصول الى الفئة MenuStars عبر الحلقة foreach و يجب عليها التحقق من حالة الوقت لكي لا يتم تشغيل القائمة مع اللعب، اضف الكود التالي:

```
1.  	foreach(MenuStars m_stars in menuStars)
2.  	{
3.  		if(Time.timeScale == 1)
4.  		{
5.  			m_stars.menuStar.SetActive(false);
6.  
7.  		}else m_stars.menuStar.SetActive(true);
8.  	}
```

صراحةً ان هذا الكود بسيط جداً فجل ماسيقوم بعمل هو التحقق من حالة الوقت و سيوصل الى متغيرات الفئة MenuStars على المتغير m_stars الذي يمثلها و سيتم تطبيق الامر على جميع القوائم، ففي حالة لم يتحقق الشرط ستعمل القائمة.

- هنا سنقوم بعمل خطوة جميلة وهي عمل بارامتر يحدد لنا نوع القائمة و رقم الانميشن لكي نستطيع تشغيل القائمة التي نريدها دون الحاجة لاستنساخ الأكواد فهذا الامر مكلف و استنساخ نفس الكود مراراً و تكرار ليس امر جيد لهذا يجب علينا استخدام طريقة من خلالها نقلل من حدة نسخ الاكود، اذن اضف الدالة التالية و هي تختص بتشغيل جميع القوائم:

```
1.  	void Menu(TypeMenu t_menu, int n_anim)
2.  	{
3.  		if(typeMenu == t_menu && !IsMinMenu)
4.  		{
5.  			foreach(MenuStars m_stars in menuStars)
6.  			{
7.  				m_stars.timeShow -= Time.deltaTime;
8.  				if(m_stars.timeShow <= 0)
```

```
9.                                  {
10.                                         m_stars.timeShow = 0;
11.                                         timsStop -= Time.deltaTime;
12.                                         if(timsStop <= 0)
13.                                                 Time.timeScale = 0;
14.
15.                                         if(Time.timeScale == 0)
16.                                         {
17.                                                 animMenu[n_anim].enabled = true;
18.                                                 m_stars.menuStar.SetActive(true);
19.                                                 m_stars.Stars.SetActive(true);
20.                                                 m_stars.Buttons.SetActive(true);
21.                                         }
22.                                 }
23.                         }
24.                 }
25.         }
```

لاحظ المراحل التي تمر بها الدالة و منها انها تعيد لنا بارامترين الاول يحدد نوع القائمة و الثاني رقم الانميشن ففي حال قمنا باستدعاء الدالة سنقوم بوضع المتغيرات التي تطلبها الدالة.

لاحظ الشرط الاول وهو يتحقق من نوع القائمة و توقف القائمة المصغرة و داخلها سيتم الوصول الى الفئة MenuStars ومنها الوصول الى المتغير timeShow و الذي سيحدد لنا متى ستعمل القائمة، ففي حاله تحقق الشرط فلن تعمل القائمة الا اذا توقف الوقت لهذا سيتم اناقص قيمة المتغير timsStop و سيتم التحقق من حالته فأن اصبحت صفر سيتوقف الوقت و حال توقف الوقت ستعمل القائمة، لاحظ ان العملية بسيطة جداً فكل المطلوب منها هو توقف الوقت في فترة زمنية يحددها المتغير timeleft.

بالنسبة للبارامترات تلاحظ انني استخدمتها في اول شرط و في الانميشن في السطر 17.

- حالياً لن نترك القائمة بهذه البساطة فعلينا ربط كل قائمة بنوعها، لهذا قم باضافة الكود التالي:

```
1.                  if(typeMenu == TypeMenu.WinGold)
2.                  {
3.                          SelectMenu[0].SetActive(true);
4.                          SelectMenu[1].SetActive(false);
5.                          SelectMenu[2].SetActive(false);
6.                          Menu(TypeMenu.WinGold, 0);
7.                  }
8.                  if(typeMenu == TypeMenu.WinNormal)
9.                  {
10.                         SelectMenu[0].SetActive(false);
11.                         SelectMenu[1].SetActive(true);
12.                         SelectMenu[2].SetActive(false);
13.                         Menu(TypeMenu.WinGold, 1);
14.                 }
15.
16.                 if(typeMenu == TypeMenu.Loser)
17.                 {
18.                         SelectMenu[0].SetActive(false);
19.                         SelectMenu[1].SetActive(false);
20.                         SelectMenu[2].SetActive(true);
21.                         Menu(TypeMenu.WinGold, 2);
22.                 }
```

حسناً لا تقلق ان الكود بسيط جداً جداً، لاحظ ان لذينا 3 شروط وكل شرط يتحقق من نوع معين، فلاحظ ان النوع الاول هو للقائمة الذهبية لهذا سيتم تشغيل القائمة الذهبية بينما باقي القوائم ستتوقف، حالياً لا توجد معنا الدوال المذكورة و لكن سنقوم بتعريفها بعد هذا الشرح، اذن لاحظ الشرط الثاني يختص بتشغيل القائمة الثانية ونفس الكود المستنسخ و لكن تم تشغيل القائمة الثانية و اقفال باقي القوائم و نفس الشيء على القائمة الخسارة، لاحظ استخدامي للدالة Menu ففي كل شرط اعطيت للدالة نوع معين و رقم انميشن معين لكي تعمل القائمة التي تدل على ذلك. ان هذه الشروط ضرورية جداً لكي نقوم بتوقيف قائمة معينة و تشغيل اخرى، فلن نسمح بتشغيل جميع القوائم مرة واحدة و هذا سيسبب مشكلة كبيرة و لكن بهذا الشكل نكون قد تخطينا المشكلة.

تدمير الكائنات في المشهد:

ان عملية تدمير الكائنات في المشهد ضرورية جداً ففي حالة خسارتك او فوزك علينا تدمير الاعداء فلن نتركهم موجودين في المشهد و هذا قد يسبب مشكلة فلن تنتهي اللعبة، تدمير الكائنات يشمل الاعداء و طلقاتهم فهناك طلقات تتبع اللاعب و علينا تدميرها كلها، العملية سهله جداً فكل ماسنقوم به هو البحث عن جميع الكائنات ووضعهم في مصفوفة ثم تدميرهم، اذن اضف هذه الدالة:

```
1.  void DeleteAllObjects()
2.  {
3.      //Find And Destroy All EnemyOne
4.      GameObject[] EnemyOne;
5.      EnemyOne = GameObject.FindGameObjectsWithTag("EnemyOne");
6.      for(int i = 0; i < EnemyOne.Length; i++)
7.          Destroy(EnemyOne[i]);
8.
9.      //Find And Destroy All EnemyTwo
10.     GameObject[] EnemyTwo;
11.     EnemyTwo = GameObject.FindGameObjectsWithTag("EnemyTwo");
12.     for(int i = 0; i < EnemyTwo.Length; i++)
13.         Destroy(EnemyTwo[i]);
14.
15.     //Find And Destroy All EnemyThree
16.     GameObject[] EnemyThree;
17.     EnemyThree = GameObject.FindGameObjectsWithTag("EnemyThree");
18.     for(int i = 0; i < EnemyThree.Length; i++)
19.         Destroy(EnemyThree[i]);
20.
21.     //Find And Destroy All BE_1
22.     GameObject[] BE_1;
23.     BE_1 = GameObject.FindGameObjectsWithTag("b_Enemy_1");
24.     for(int i = 0; i < BE_1.Length; i++)
25.         Destroy(BE_1[i]);
26.
27.     //Find And Destroy All BE_2
28.     GameObject[] BE_2;
29.     BE_2 = GameObject.FindGameObjectsWithTag("b_Enemy_2");
30.     for(int i = 0; i < BE_2.Length; i++)
31.         Destroy(BE_2[i]);
32.
33.     //Find And Destroy All BE_3
34.     GameObject[] BE_3;
35.     BE_3 = GameObject.FindGameObjectsWithTag("b_Enemy_3");
36.     for(int i = 0; i < BE_3.Length; i++)
37.         Destroy(BE_3[i]);
38. }
```

حسناً دعنا نناقش هذا الكود الطويل، لاحظ ان الأكواد هي عبارة عن استنساخات لأول كود، لاحظ التعليقات اعلى كل كود وهي تدل على وظيفة هذا الكود، الكود الاول سيقوم بالبحث عن العدو الاول و من ثم تخزينه في المصفوفة EnemyOne ثم عمل حلقة لتحديد على جميع الاعداء المخزنين و من ثم تدميرهم، ان الامر ينطبق على جميع الاكود لكن بالبحث عن اشياء اخرى مثل العدو الثاني و الثالث و الطلقات كما هو موضح و سيتم تدمير الجميع.

الان قم بمناداة هذه الدالة في الشرط ظهور القوائم وهذا يشمل موت اللاعب.

القائمة الرئيسية Main Menu:

القائمة الرئيسية هي القائمة الاهم في اللعبة و ربما العمل الاكبر سيكون فيها و دائماً عند تشغيل أي لعبة تلاحظ انتقالك الاولي الى قائمة اللعبة و هذا يتيح لك التعديل على جودة اللعبة، الاصوات، تحديد مستوى و العديد من الاشياء و قائمتنا ستحتوي على بعض الاشياء المهمة مثل الانتقال الى مستوى و تعديل الاصوات وجودة اللعبة و باقي القوائم الاساسية مثل الخروج، تشغيل اللعبة و معلومات المطور، اخيراً علينا عمل قائمة تقوم بتحميل المستوى التالي و هذا امر مهم.

حسناً دعنا نبدأ بصميم القائمة الرئيسية، اتبع الخطوات التالية:

- انشئ Canvas و تأكد من ان اعداداته مل القائمة السابقة.
- منة انشئ Image و هو يمثل الخلفية، ابحث عن الصورة colored_castle اجعلها الخلفية.
- داخلة انشئ Image و اجعل صورته هي metalPanel_red واضف الية text بحيث يعرف القائمة و تأكد من ضبط موضعه كما هو موضح في الاعلى| وقم بتسميته Main Menu.

- بالنسبة للأزرار فداخل القائمة قم بأنشاء 5 ازرار بحيث ترتبهم كما هو موضح في الصورة، اما بالنسبة لصورهم فقم بأخذ الصورة red_button12 تأكد من تقطيعها و اخيراً قم بتسمية كل زر.

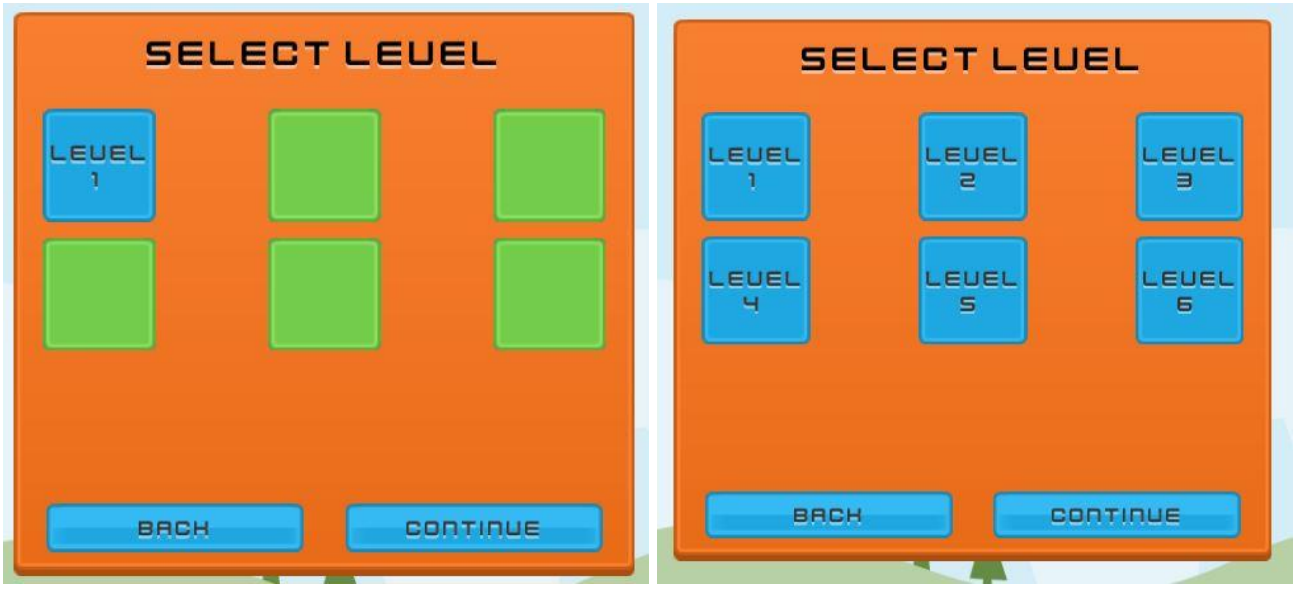
اذن هذا هو شكل القائمة الرئيسية و التي ستتفرع منها عدة قوائم تمثل كل زر في القائمة.

الان بشكل مباشر قم بنسخ القائمة و احذف الازرار منها و هذه القائمة ستكون Options و التي تتحكم بإعدادات الصوت و دقة الشاشة و هي القائمة التي ستمثل الزر Options، اذن بنسخ هذه القائمة تأكد من ان حجمها كما في الصورة السابقة و قم بتسميتها Settings.

بالنسبة لصورة الخلفية فهي red_button08 وهي ستكون الصورة الموحدة لجميع القوائم الفرعية.

اذن بأنشاء Slider و انسخه 3 مرات و تأكد من تباعدهم و اضف لكل واحد text بحيث يميز ماهي وظيفة هذا الشريط، بالنسبة لزر التحكم Handle اجعل صورته هي blue_circle اما الشريط Fill فصورته ستكون blue_button10 و اخيراً الخلفية Background ستكون صورتها grey_button11 و هذا يشمل جميع الـ Slider في القائمة، اخيراً اضف زر وجعل صورته blue_button10 و سمة Back وهو الذي سيتيح لنا العودة الى القائمة الرئيسية، ان الصورة في الاعلى فقط توضيحية اما عند تشغيل اللعبة ستجد القائمة الرئيسية فقط و حين الضغط على اي زر سيتم فتح القائمة التي تخصه ثم تختفي القائمة الرئيسية بينما الزر Back سيعيد فتح القائمة الرئيسية و يخفي القائمة الحالية و السبب هو تجنباً لفتح اكثر من قائمة في وقت واحد.

قائمة تحديد المستويات Select Level وهي تختص بفتح المستويات لكي تكمل المرحلة التي وصلت لها، ايضاً تستطيع عمل Continue للمرحلة التي وصلت لها، اذن لاحظ هذه الصورة التي تمثل قائمة تحديد المستويات:



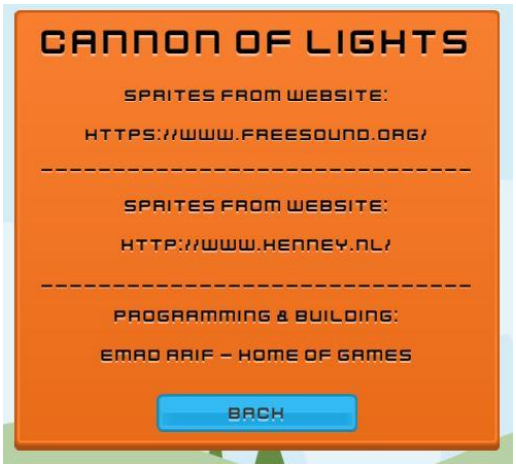
كما تعلم اننا سنعمل على 6 مراحل لهذا لذينا 6 ازرار و بحيث ان كل زر يمثل مستوى معين، بالاضافة الى الزر Continue الذي يستكمل آخر مرحلة.

حالياً لن نترك الازرار بهذا الشكل فهنا سيستطيع اي شخص الوصول الى اي مرحلة دون مواصلة اللعبة، و لهذا يجب علينا تأمين الأزرار والعملية بسيطة جداً وهي كالاتي:

قم بنسخ الازرار او يفضل وضعهم داخل قائمة باسم Buttons ثم نسخ القائمة و تسميتها LockButtons وفي كل زر قم بحذف الـ Text الخاص به قم غير لون الازرار الى الصورة green_button10 و اخيراً قم بحذف الزر الاول او تأمين الزر الاول كما هو موضح في الصورة.

بهذا الشكل نكون قد انهينا تصميم قائمة تحديد المراحل و سيتبقى لنا تصميم القائمة Develop و هذه القائمة وظيفتها عرض معلومات المطور فنحن قمنا ببرمجة اللعبة و تعديل القوائم بينما الصور و الاصوات اخذناها من مواقع لهذا يجب علينا ذكر الاعتمادات.

بالنسبة لشكل القائمة فهي بسيطة جداً، لاحظ الصورة التالية:



تلاحظ ان هذه القائمة تختص بعرض معلومات المطور فالبرمجة و البناء قمنا بعملة بينما الاصوات اخذناها من مواقع و تم ذكرها كم هو موضح، ان هذه القائمة تعتمد على المكون Text الذي من خلاله قمت بكتابة المعلومات كما هو موضح، و بهذا الشكل نكون قد انهينا العمل على القائمة.

اخيراً تبقى لنا عمل قائمة اخيرة و هي الخروج من اللعبة و هذه القائمة غالباً تجدها في عدة مواقع و تأكد الخروج من اللعبة، لاحظ الصورة:

ان تصميم هذه القائمة يختلف قليلاً و لكن خلفيتها هي نفس خلفية القائمة الرئيسية بالاضافة الى الازرار لكن في حالة ظهرت هذه القائمة ستجد انك تستطيع الوصول الى ازرار القائمة الرئيسية و لهذا يجب علينا تأمين الخلفية! فهنا سنستخدم العنصر Panel اذن اتبع الخطوات التالية لتأمين الخلفية:

- قم بأنشاء Panel و تأكد من انه يغطي الشاشة بشكل كامل و قم بتغيير لون الخلفية بحيث يرى بوضوح.
- اخيراً جعل صورة القائمة ابن لهذا الـ Panel و ستلاحظ ان القائمة الخلفية مؤمنة او لن تستطيع الوصول الى ازرارها الا في حال تم الخروج من القائمة الحالية،

حسناً قم بإخفاء جميع القوائم لكي نستطيع العمل عليها اي قم بجعل القائمة الرئيسية هي الموجودة في المشهد فقط، الان علينا انشاء ملف MenuManager و الذي سيختص بتشغيل جميع الازرار في القوائم

علينا عمل مشهد اخير من خلاله نستطيع تحميل المرحلة القادمة تصميم المشهد سهل جداً فكل محتواه هو خلفية و Slider لعرض نسبة تحميل المستوى القادم.

- انشئ مشهد جديد و اضف الية خلفية.
- فيه انشئ Slider و هذا سيكون شريط التحميل لهذا قم بجعل قيمتة 100.

اذن بهذا الشكل نكون قد جهزنا المشهد، يمكنك اضافة اشياء اخرى مثل نصوص او صور او اي شيء تراه مناسب، بالمناسبة هذا شكل مشهدي ☺.

انشاء مدير القوائم MenuManager:-

الملف MenuManager يعتبر من اهم الملفات في اللعبة فهو الذي يتحكم بنقلك الى المستويات وتحميلها و اشياء عديدة سنتعرف عليها في هذه الفقرة، من اهم الاشياء التي ستتعرف عليها في هذه الفقرة هي الفئة Application و دوال الـ Coroutines و الوصول اعدادات الجرافيك عبر الفئة QualitySettings، اشياء اخرى مثل التحكم بالموسيقى و الاصوات وغيرها، اذن لنقم بأنشاء ملف برمجي باسم MenuManager لكي نعمل علية.

برمجة المستويات و الانتقال بين المراحل:

سنقوم بتقسيم العملية الى 3 نقاط من خلالها نكمل العمل على ملف مدير القوائم، اذن لنبدأ.

- حال انشاء ملف MenuManager اول خطوة سنعملها هي برمجة الانتقال بين المراحل وهذا يشمل العودة الى المرحلة السابقة ،القادمة، اعادة المرحلة، الخروج من اللعبة، تحميل المستوى ومواصلة المرحلة الاخيرة او اكمال المرحلة التي وصلت لها، اذن اضف الدوال التالية لكي نناقشها:

```
1.  public enum TypeMenu{
2.      main,
3.      loading,
4.      none
5.  }
6.
7.  [Header("Menu & Levels")]
8.  public TypeMenu typeMenu;
9.  private int ContinueLevel;
10.
11. public void NextLevel(string loading)
12. {
13.     Application.LoadLevel(loading);
14. }
15.
16. public void Quit()
17. {
18.     Application.Quit();
19. }
20.
21. public void Continue()
22. {
23.     Application.LoadLevel(ContinueLevel);
24. }
25.
26. public void RestartLevel()
27. {
28.     Application.LoadLevel(Application.loadedLevel + 0);
29. }
30.
31. public void StartGame()
32. {
33.     Application.LoadLevel(Application.loadedLevel + 1);
34. }
35.
36. public void BackToMainMenu(string nameLevel)
37. {
38.     Application.LoadLevel(nameLevel);
39. }
40.
41. public void Level(int i)
```

```
42.     {
43.             Application.LoadLevel(i);
44.     }
```

اذن لاحظ في البداية عرفت enum يمثل انواع القوائم و تجد ان لذينا قائمة رئيسية، مصغرة و اخيرة لا شيء و سيستخدم في المستويات فلا حاجة لتشغيل الأكواد ايضاً ستظهر مشكلة ان لم نضعه، هناك عدة طرق و لكن افضل هذه الطريقة.

لاحظ المتغيرات، بحيث ان هذه المتغيرات تعيد لنا نوع القائمة و استمرار اللعب، بالنسبة للدوال تجد ان لدينا حالياً خمس دوال، لاحظ معي الدالة الاولى و هذه الدالة تحدد لنا المستوى التالي ولاحظ انها تعيد بارامتر من نوع String بحيث نحدد المرحلة التي سننتقل اليها بالاسم، ففي حالة سالتنا لماذا لم اجعلة متغير من نوع int لكي يوفر علي عملية كتابة اسماء المستويات، فسأجيبك اننا لن ننتقل الى المستوى التالي مباشرة بل علينا الانتقال الى قائمة التحميل لكي يتم تحميل المستوى التالي.

حال لاحظت على جميع الدوال ترى استخدامي للفئة Application ففي حال لم تعلمها فأليك معلومة عنها، الفئة Application توفر لك دوال تساعدك على تطبيق مستويات و منصات و اشياء اخرى و يمكنك ان ترى في جميع الدوال استخدامي لدوال تمثل الاوامر التي نريدها، مثلاً الدالة Quit بسيطة جداً فبشكل مباشر من الفئة استخدمت الدالة Quit وسيتم تطبيق الخروج من اللعبة بشكل كامل، الدالة LoadLevel تطلب متغير من نوع String او int وهذا يفسر سبب تعريفي للبارامتر loading في الدالة NextLevel، في الدالة Continue تلاحظ استخدامي للمتغير Continue لكي نستطيع تطبيق رقم المرحلة الاخيرة التي وصلت لها ولكن لن نترك الامر بهذه البساطة بل علينا الانتقال الى آخر مرحلة اعتماداً على النقاط.

لاحظ الدالتان RestartLevel و StartGame وهما تستخدمان نفس الكود و لكن مع اختلاف في رقم المستوى، الدالة RestartLevel ستقوم بتطبيق المستوى نفسة و كما تلاحظ انه سيتم تطبيق المستوى 0 ليس صفر من الترتيب بل صفر من المستوى الحالي، لاحظ الدالة StartGame بشكل مباشر ستقوم بتطبيق المستوى القادم اي المستوى الذي يلي المستوى الحالي.

هذه هي الدوال التي عرفناها:

- NextLevel: تطبيق المستوى القادم.
- RestartGame: اعادة تشغيل المستوى.
- Quit: الخروج من اللعبة.
- Continue: تطبيق آخر مستوى وصلت له.
- BackToMainMenu: العودة الى القائمة الرئيسية (حال اللعب).
- StartGame: بدأ اللعب.
- Level: تطبيق المستوى الذي وصلت اليه (هذا الامر اختياري).

- بالنسبة لتحميل المستوى فهنا سنتعرف على دالة من نوع Coroutine و هذا النوع من الدوال يتيح لك تشغيل نفسة ثم تتوقف لفترة و هي تعيد yield return و تختلف نوع الاعادة مثلاً null سيتم توقيف الدالة لمدة فريم (إطار) واحد، بينما توفر لك فئة باسم WaitForSeconds و هي تتيح لك توقيف الدالة لثواني و من ثم تعيد تشغيلها. لاحظ هذا الكود يوضح طريقة استخدام الفئة WaitForSeconds :

```
1.  void Start()
2.  {
3.          StartCoroutine("Test");
4.  }
5.
6.  public IEnumerator Test()
7.  {
8.          print("wait");
9.
```

```
10.             yield return new WaitForSeconds(3);
11.             print("ready");
12.
13.             yield return new WaitForSeconds(3);
14.             print("GOoOoO");
15.         }
```

جميعنا نعرف ان الدالة Start تقوم بتشغيل نفسها في اول فريم فقط، هنا لاحظ في الدالة Test انا من نوع IEnumerator و هي تحقق لنا 3 حالات للانطلاق(انتظر، استعد، انطلق) حال تطبيق اول سطر سيتم توقيف الدالة لمدة 3 ثواني كما هو موضح و من ثم سيعاد تشغيلها و طبعه الكود التالي و من ثم تتوقف لمدة 3 ثواني وبعدها تطبع الكود الاخير ففي حال قمت بتجربة هذه الدالة ستلاحظ طريقة عملها.

بالنسبة لتشغيل دوال Coroutine يجب علينا تشغيلها في الدالة التي تمثلها، لاحظ في الدالة Start الدالة StartCoroutine تقوم بتشغيل الدالة وهناك عدة انواع من هذه الدوال ومنها StopCoroutine و StopAllCoroutines وتقوم بتوقيف جميع الدوال من نوعها. ويمكنك كتابة الدالة على شكل String او ان تناديها بشكلها العادي.

حسناً يجب علينا انشاء دالة تقوم بتحميل المستوى القادم، اولاً قم بتعريف متغير يمثل شريط التحميل و من ثم اضف الكود التالي:

```
1.      public IEnumerator LoadingNextLevel()
2.      {
3.              AsyncOperation loading = Application.LoadLevelAsync(Application.loadedLevel + 1);
4.              while(!loading.isDone)
5.              {
6.                      SliderLoading.value = loading.progress * 100;
7.                      yield return null;
8.
9.              }
10.     }
```

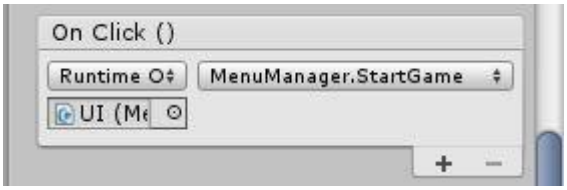
لاحظ معي، قمت باستخدام فئة جديدة وهي AsyncOperation هذه الفئة تتيح لك تزامن العمليات التي تقوم بها مع المتغيرات التي توفرها، لاحظ اول سطر بحيث قمت بتعريف متغير يمثل هذه الفئة و منها سيتم مزامنة تحميل المستوى القادم، في الشرط While وتجد استخدامي المتغير isDone و هو من نوع Bool وهو يساعدك على معرفة ما اذا كان هذا المستوى لم يكتمل تحميلة ام اكتمل فلاحظ في الشرط انه في حالة لم يكتمل التحميل سيتم وضع الاشياء المحملة في الـ Slider ولكن يجب علينا تمريرها عبر متغير من نوع float و هذه الفئة توفر لنا متغير progress و قيمتة من (0 الى 1) وهذا يفسر سبب ضرب الرقم 100 في المتغير فلو عدنا لحجم الـSlider ستجد ان اعلى قيمتة له هي 100 و بهذا الشكل سيتم وضع الاشياء المحملة على شكل ارقام في الشريط، اخيراً الدالة تعيد لنا null و سيتم تطبيق الشرط.

يجب علينا تشغيل الدالة لهذا اضف السطر التالي في Start:

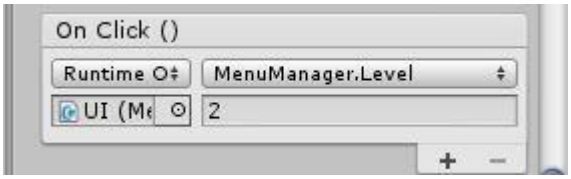
```
1.      if(typeMenu == TypeMenu.loading)
2.      {
3.              StopAllCoroutines(LoadingNextLevel());
4.      }
```

سيتم التحقق من نوع القائمة، فنحن لن نترك الامر يتطبق على جميع القوائم لهذا قمت بتحديد قائمة و هي loading.

اذن العملية بسيطة الان بالعودة الى القائمة الرئيسية قم بوضع الدوال في الازرار بالتحديد في قائمة الاحداث، لاحظ الصورة:



في كل زر قم بوضع الدالة التي تمثله، هناك دوال تعيد بارامترات من نوع int مثل الدالة Level قم بوضعها في القائمة Select Level تحديداً في الازرار بحيث ان كل زر سوف ينقلك الى المستوى الذي يمثله، لاحظ الصورة:



لاحظ ان الزر الاول سيقوم بتطبيق المستوى الاول و رقمة 2 (ليس الزاماً ان يكون نفس الرقم، فهي حسب ترتيبك للمستويات).

- حسناً دعنا نعود الى المشهد الذي يمثل تحميل المستويات فهنا يجب علينا تشغيل الدالة LoadingNextLevel لكي يتم تحميل المستوى التالي، اذن حال فتحك لمشهد التحميل قم بحسب الملف البرمجي الى اي كائن و اضف العنصر Slider الى المتغير الذي يمثله اخيراً قم بجعل نوع القائمة هو loading:

في حال قمت بتجربة المشروع و لم يعمل فلا مشكلة لأننا لم نقم بترتيب المشاهد لهذا سأترك الامر لنهاية الفقرة.

- علينا استكمال المستوى الاخير لهذا سنعود الى قيم الاسم Score و ربط رقم المستوى فيها، اذن قم باضافة هذا الكود في الدالة Start :

```
1.  if(typeMenu == TypeMenu.main)
2.  {
3.          if(PlayerPrefs.GetInt("Score") < 3000)  //level_1
4.                  ContinueLevel = 2;
5.
6.          if(PlayerPrefs.GetInt("Score") >= 3000)  //level_2
7.                  ContinueLevel = 3;
8.
9.          if(PlayerPrefs.GetInt("Score") >= 6000) //level_3
10.                 ContinueLevel = 4;
11.
12.         if(PlayerPrefs.GetInt("Score") >= 9000)  //level_4
13.                 ContinueLevel = 5;
14.
15.         if(PlayerPrefs.GetInt("Score") >= 12000) //level_5
16.                 ContinueLevel = 6;
17.
18.         if(PlayerPrefs.GetInt("Score") >= 15000) //level_6
19.                 ContinueLevel = 7;
20. }
```

اذن كل شرط يعبر عنه بتعليق فكل تعليق يدل على حالة الشرط وهي عباره عن ارقام المستويات و هي تعتمد على المتغير ContinueLevel الذي استخدمنه في الدالة Continue، لاحظ معي الشرط الاول يجب ان تكون حالة الاسكورات قل من 3000 اي لم تقوم بلعب اللعبة او لم تفز في اول مرحلة لهذا ستضل المرحلة الاولى مفتوحة و حال فوزك فيها ستفتح الثانية ويجب عليك التأكد من ان الكود يعمل على القائمة الرئيسية main في الدالة Start وحال الخروج من المرحلة و العودة الى المستويات ستجد انه قد فتح المرحلة التي تريدها (ليس على الازرار المستويات و لكن على الزر Continue) و حال الضغط علية سيتم تطبيق المستوى الاخير.

بالحيث عن فتح المستويات بحيث ان ازرار المستويات تفتح مع المستوى الذي وصلت له فهذا الشيء يعتمد على ملف خارجي آخر و يفضل ان يكون ملف خارجي باسم LevelsManager وهذا الملف سيوضع في القائمة Select Levels فلا حاجة لوضعة في ملف آخر. اذن قم بأنشاء هذا الملف ووضعة في الكائن Select Levels و اضف المتغيرات التالية:

```
public GameObject[] Buttons;
public GameObject[] LockButtons;
public int OpenLevel;
```

اذن لدينا متغير يمثل رقم الزر الذي سنفتحه و مصفوفتان، واحدة تحوي ازرار الانتقال الى المستوى المطلوب، اما الثانية فهي لتحديد اقفال الازرار لكي نقوم بحذفها حال فتح مستوى جديد.

هنا سيتوجب علينا انشاء دالة تحوي حلقة تقوم باخفاء الاقفال حال فتح الزر الذي تخفية، قم باضافة الدالة التالية:

```
void HideLock()
{
        for(int i = OpenLevel; i < LockButtons.Length; i++)
        {
                LockButtons[i].SetActive(false);
                print(i);
        }
}
```

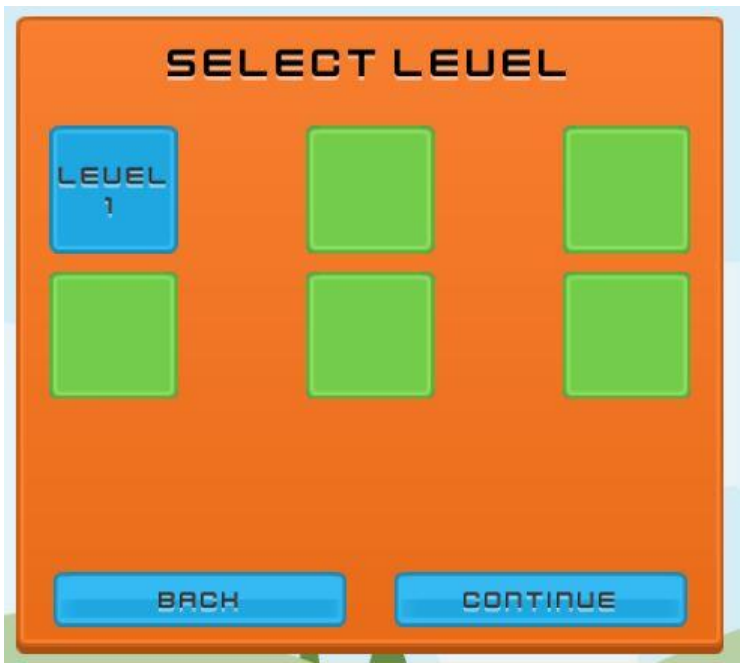
ان الحلقة تقوم بجعل قيمة المتغير i تساوي عدد ازرار المستويات و تلاحظ ان المتغير i ايضاً محصور بين عدد الاقفال اي يساوي عدد الاقفال لكي يتم تحديدها كلها، الان عدد الاقفال سيكون بقيمة المتغير i و لكن لن نترك الامر بهذه البساطة بل علينا اعطاء قيمة للمتغير OpenLevel لان قيم المتغير i مربوطة بقيم المتغير OpenLevel، اضف معي الكود التالي و يجب ان يكون في Update:

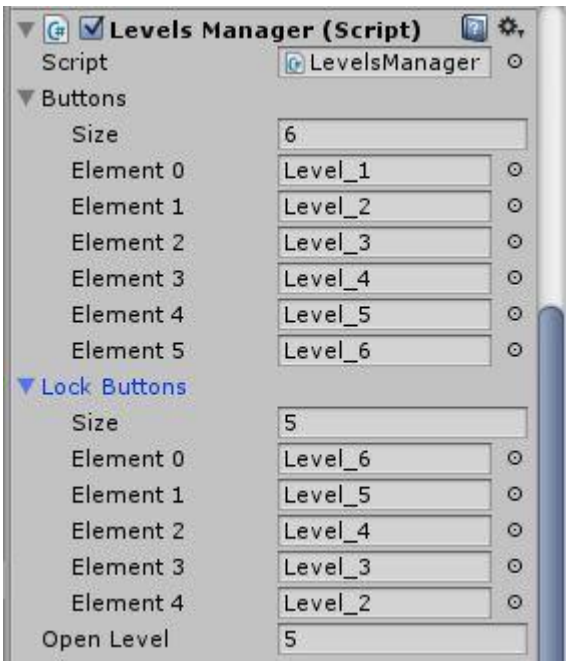
```
        void Update()
        {
                PlayerPrefs.Save();
                HideLock();

                if(PlayerPrefs.GetInt("Score") < 3000)
                        OpenLevel = 5;

                if(PlayerPrefs.GetInt("Score") >= 3000)
                        OpenLevel = 4;

                if(PlayerPrefs.GetInt("Score") >= 6000)
                        OpenLevel = 3;

                if(PlayerPrefs.GetInt("Score") >= 9000)
                        OpenLevel = 2;

                if(PlayerPrefs.GetInt("Score") >= 12000)
                        OpenLevel = 1;

                if(PlayerPrefs.GetInt("Score") >= 15000)
                        OpenLevel = 0;
        }
```

العملية بسيطة وهي تعتمد على اعطاء قيمة للمتغير OpenLevel مع زيادة الـ Score و بزيادته يترتب عليه فتح المراحل فالمرحلة الاولى ستعطينا القيمة 5 وهي تعني ان جميع الاقفال الخمسة ستكون موجودة بينما حال فتح المستوى التالي سينقصف قفل و لكن خارجياً علينا

ترتيب الاقفال في المصفوفة تنازلياً، اذن هذه العملية الضرورية لفتح الاقفال، الان بالعودة الى المشروع ستجد المصفوفات و المتغير OpenLevel وليس من الضروري اعطائه قيمة فهي ستحدد تلقائياً.

حال فتح المصفوفات علينا جعل المصفوفة Buttons حجمها 6 بينما المصفوفة LockButtons عددها 5 كما هي محاطة في الشرط، بالنسبة لترتيب الاقفال اجعل القفل الاول في آخر رقم (رقم 5) و الثاني فوقة الى ان تصل الى بداية المصفوفة، لاحظ معي هذه الصورة:





يجب ان يكون حجم المصفوفات وقيمها بهذا الشكل، ايضاً شكل القائمة Select Level لكي ترى العمل بشكل جيد.

التحكم بإعدادات الموسيقى و الاصوات و الجودة:

اولاً هذه ليست فقرة متفرعة فعلياً و لكن اردت تمييزها لأهميتها و ضرورتها فهي تعتمد على انشاء الملف AudioManager و الذي سنناقشه في فقرة خاصة.

دعنا نعد الى قائمة الاعدادات Options و نرى ما نستطيع عملة:

اذن لنناقش هذه القائمة، لاحظ ان لدينا الشريط Quality و الذي يمثل جودة اللعبة وله 3 قيم لهذا اجعل قيمة في Value هي 2 بحيث ان الصفر يعبر عن قيمة.

الشريطان Sounds, Music اجعل قيمهم هي 1، ايضاً تجد الـ Toggle في كلا الشريطان بحيث ان كل زر يعبر عن تشغيل الاصوات و ايقافها و سيترتب على إيقافه اشياء مثل توقيف شريط الصوت الذي يمثله، لاحظ هذه الصورة:

حال الغاء تفعيل الخيار تجد ان الشريط قد تغير لونة ولكن ليس تغيير لون فقط بل ايقاف تحريكه و سببه هو الوصول الى الخيار Interactable برمجياً فهو يعتمد على حالة الـ Toggle الذي يمثل القائمة و هذا الامر ينطبق على شريط الموسيقى.

اذن المفهوم سهل و برمجياً سنعتمد على الفئة PlayerPrefs لحفظ قيم الاعدادات ووضعها في المقدمة، سنستخدم اسمين لحفظ اعدادات الموسيقى و هما (Music و MusicIs) اما IsMusic فهو يمثل حالة الـ Toggle و سيترتب عليها توقيف الموسيقى حال الغاء تفعيل الـ Toggle، الاسم Music سيعتبر عن قوة الصوت بالتعامل مع قيمة الـ Volume في الاصوات وهو يعيد قيمة الـ Slider الحالية و يحفظها.

نفس الشيء ينطبق على الاصوات Sounds وهو حفظ الاعدادات عن طريق اسمان (Sounds و SoundsIs).

حسناً لنبدأ ببرمجة الاصوات، الموسيقى وجودة اللعبة و عطائها اعدادات لكي نقوم بإعادة استخدامها خارجياً.

اضف المتغيرات التالية وهي تمثل الـ Sliders و الـ Toggles الخاصة بالأصوات و الموسيقى:

```
1.  [Header("Options Menu")]
2.  public Slider MusicSlider;
3.  public Slider SoundsSlider;
4.  public Toggle SoundsToggle;
5.  public Toggle MusicToggle;
```

لاحظ معنا Slider 2 واحد يمثل الاصوات و الثاني للموسيقى و نفس الشيء ينطبق على الـ Toggles المعرفة.

اضف الدالتان التاليتان وهما تحفظان قيمة المتغير Value حال تغيرها في الشريطان:

```
1.  public void soundsSlider()
2.  {
3.          PlayerPrefs.SetFloat("SliderSounds", SoundsSlider.value);
4.  }
5.
6.  public void musicSlider()
7.  {
8.          PlayerPrefs.SetFloat("SliderMusic", MusicSlider.value);
9.  }
```

الدالة الاولى تقوم حفظ قيمة شريط الاصوات بينما الدالة الثاني تحفظ قيمة شريط الموسيقى و سنستخدمها في المتغير Volume في الاصوات.

هنا يجب علينا اربط حالة interactable الخاص بالاصوات بحال الـ Toggle، قم باضافة الكود التالي في Update:

```
1.          void Update()
2.          {
```

```
3.                  PlayerPrefs.Save();
4.
5.                  if(typeMenu == TypeMenu.main)
6.                  {
7.                          //this Script to save sounds volume----------------------
8.                          if(SoundsToggle.isOn == true)
9.                                  PlayerPrefs.SetString("SoundsIs","true");
10.                         if(SoundsToggle.isOn == false)
11.                                 PlayerPrefs.SetString("SoundsIs","false");
12.
13.                         if(PlayerPrefs.GetString("SoundsIs") == "true")
14.                                 SoundsSlider.interactable = true;
15.
16.                         if(PlayerPrefs.GetString("SoundsIs") == "false")
17.                                 SoundsSlider.interactable = false;
18.                         //this Script to save sounds volume----------------------
19.
20.                         //this Script to save music volume----------------------
21.                         if(MusicToggle.isOn == true)
22.                                 PlayerPrefs.SetString("MusicIs","true");
23.                         if(MusicToggle.isOn == false)
24.                                 PlayerPrefs.SetString("MusicIs","false");
25.
26.                         if(PlayerPrefs.GetString("MusicIs") == "true")
27.                                 MusicSlider.interactable = true;
28.                         if(PlayerPrefs.GetString("MusicIs") == "false")
29.                                 MusicSlider.interactable = false;
30.                         //this Script to save music volume----------------------
31.                 }
32.         }
```

لاحظ معي، ان العملية تعتمد على انشاء اسم يحمل المعنى true او false لإعطاء المتغير interactable احدى القيم و لكن هي String لهذا يجب ان نحيط المتغير بشرط لاحظ في السطر 8 الى السطر 11 عند تغيير قيمة المتغير IsOn سيعطينا نفس قيمتة و سيحفظها، و في السطرين 13 الى 17 سيتم استعمال الشرطين فكل حالة تعطينا قيمة للمتغير interactable وهي true, false كما هو موضح، و في النهاية هذا الكود لن يعمل الا في القائمة الرئيسية فقط.

نفس الشيء ينطبق على الموسيقى و ترى التعليقات تعبر عن وظيفة الكود ففي الموسيقى لقمت بإعطائها الاسم MusicIs و استخدمت نفس الطريقة في الاعلى و لكن على الشريط الذي يمثل الموسيقى.

حالياً لن ندع الكود بهذا الشكل بل علينا حفظ قيمة المتغير IsOn لكي لأتعود الى قيمتها الافتراضية حال تشغيل اللعبة لهذا اضف الكود التالي في Start و يجب ان يعمل في القائمة الرئيسية فقط:

```
1.          if(PlayerPrefs.GetString("SoundsIs") == "true")
2.                  SoundsToggle.isOn = true;
3.          if(PlayerPrefs.GetString("SoundsIs") == "false")
4.                  SoundsToggle.isOn = false;
5.
6.          if(PlayerPrefs.GetString("MusicIs") == "true")
7.                  MusicToggle.isOn = true;
8.          if(PlayerPrefs.GetString("MusicIs") == "false")
9.                  MusicToggle.isOn = false;
```

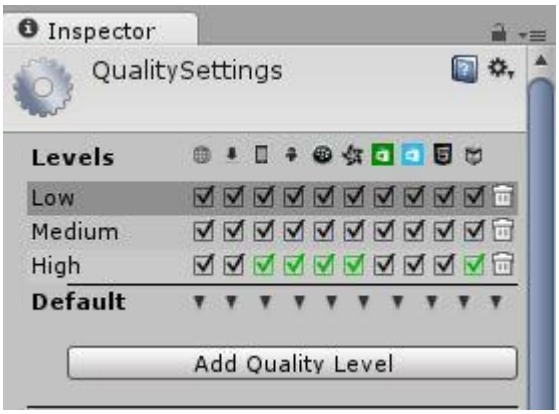
حسناً، هذه الشروط تقوم بإعطاء المتغير IsOn في الـ Toggles نفس القيم الاخيرة التي اتخذها فالشرط الاول يقوم بإعطاء المتغير نفس القيمة الاخيرة اي ان الاصوات في حالة لم تكن تعمل آخر مرة فستكون على حالها حتى في حالة الخروج من اللعبة الى ان تقوم بتشغيلها ومن ثم يحفظ القيمة الاخيرة و نفس الشيء ينطلق على الموسيقى.

بالنسبة للدالتين soundsSlider و musicSlider فأنه يجب علينا حفظ القيم الاخيرة لها لهذا علينا استخدام الدالة GetFloat لاستعادة القيمة الاخيرة التي وصل لها شريط الموسيقى و الاصوات، قم باضافة الكود التالي في Start و يجب ان يعمل في القائمة الرئيسية فقط:

```
1.      SoundsSlider.value = PlayerPrefs.GetFloat("SliderSounds");
2.      MusicSlider.value = PlayerPrefs.GetFloat("Music");
```

صراحةً اجد ان العملية بسيطة جداً فهي تعيد استخدام القيم التي حفظتها سابقاً و الغرض منها هو اعطاء القيم الاخيرة للشريط حال تشغيل اللعبة و بهذا الشكل نكون قد انهينا العمل على الاصوات (ليس فعلياً و لكن سنقوم بإنشاء ملف AudioManager بعد برمجة اعدادات الجودة و حفظها).

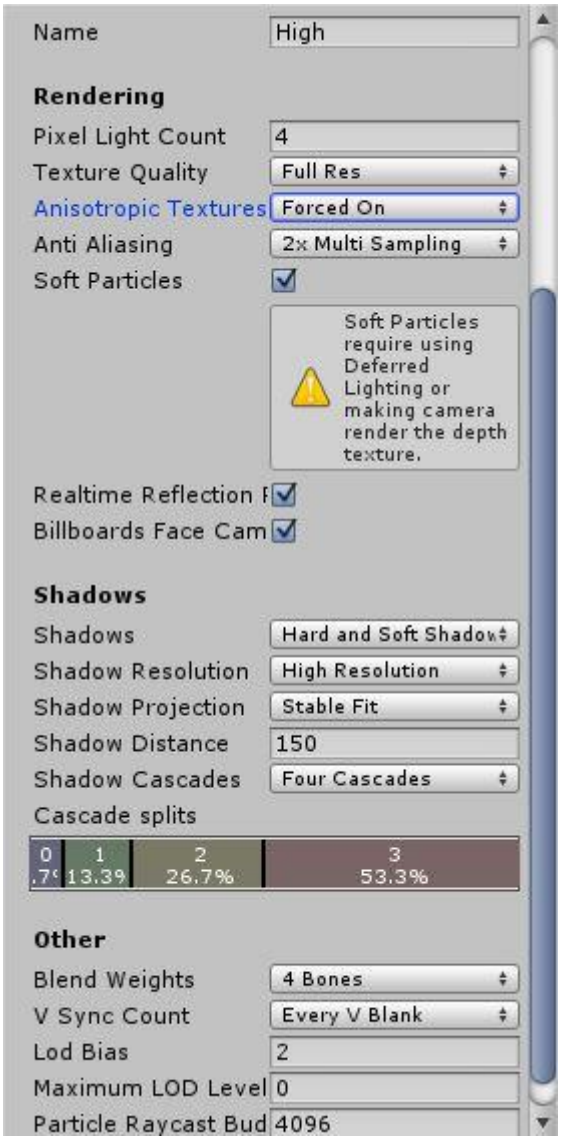
بالنسبة لإعدادات الجودة فأن علينا الوصول الى الفئة QualitySettings ويمكنك عبر البرنامج رؤية الاسماء التي تعطيها بحيث ان كل اسم يحوي اعدادات معينة لاحظ معي هذه الصورة:



يمكنك الوصول الى هذه القائمة من Edit> Project Setting> Quality و لكن لن ترى هذه القيم لأنني قمت بعملها و هي بسيطة و لكن القيم الافتراضية هي 6:

١. Fastest
٢. Fast
٣. Simple
٤. Good
٥. Beautiful
٦. Fantastic

عملية اضافة اسماء ووضع الاعدادات عليها هي امر بسيط جداً. لاحظ الزر Add Quality Level و يتيح لك اضافة مراحل للجودة فكل مرحلة تخزن جودة معينة، اما إعطائه اسم فستجد خيار اسفلة يتيح لك تحديد اسم فبكل بساطة اضف الاسم الذي تريده و من ثم اضغط على الزر و سيتم اضافة الاعدادات.



هناك العديد من الاشياء التي تساعدك على ضبط اعدادات الجودة ومنها تستطيع استخدامه في الالعاب 2D و اخرى للـ 3D فلاحظ القائمة Rendering تتيح لك التعامل مع التكستشرس و تنعيم الحواف واشياء يمكنك تجربتها.

تصنيف آخر باسم Shadows ويساعدك على التحكم بإعدادات الظلال من جودة و حجم و مسافة وغيرها.

اخيراً تصنيف باسم Other يحدد لك اشياء جانبية ففي النهاية لن نحتاج الا الى الخيار Rendering فقط اما باقي الاعدادات لن تؤثر على اللعبة.

برمجياً علينا تعريف متغير يمثل شريط الاعدادات و متغير يمثل العنصر Text لعرض اسم الجودة و في نفس الوقت تغيير الجودة، هنا علينا الوصول الى المصفوفة names و التي تظهر جميع انواع المستويات بحيث نعيد متغير من نوع float و من ثم نحولها الى int، اضف الكود التالي ويجب ان يكون خارج اي دالة:

```
1.  [Header("Quality Settings")]
2.  public Text qualityText;
3.  private float _currentQuality;
4.  public Slider mySlider;
5.  public float currentQuality
6.  {
7.      get{ return _currentQuality;}
```

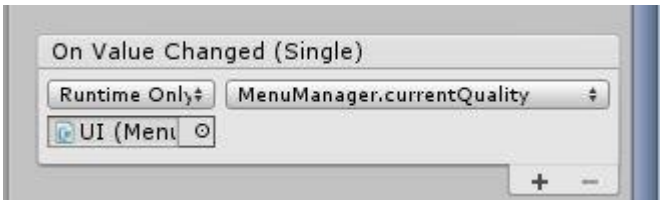
```
8.                  set
9.                  {
10.                         _currentQuality = value;
11.
12.                         QualitySettings.SetQualityLevel((int)_currentQuality);
13.                         qualityText.text = "Quality: " +
14.                                 QualitySettings.names[(int)_currentQuality];
15.
16.                         PlayerPrefs.SetInt ("Q",(int)_currentQuality);
17.                 }
18.         }
```

لاحظ قمت بتعريف متغيرين يمثلان الشريط و النص و آخر يمثل نوع الجودة وهو من نوع float، اخيراً عرفت متغير float يعيد لنا القيمة الحالة للجود و من ثم يعطينا القيمة التي اتخذناها عبر الدالة SetQualityLevel و قمت بتحويل الاسماء الى ارقام، اما النص سيقوم بطبع اسم المستوى اعتماداً على الرقم الذي اتخذناه و اخيراً قمت بحفظ اسم الجودة.

في الدالة Start علينا اعطاء القيمة الاخيرة التي وضعناها، اضف الكود التالي:

```
1.      mySlider.value = PlayerPrefs.GetInt ("Q");
2.      currentQuality = PlayerPrefs.GetInt ("Q");
```

الان بالعودة الى المشروع علينا وضع المتغير currentQuality في قائمة الاحداث الخاصة بالـ Slider الذي يمثل الجودة، لاحظ الصورة:

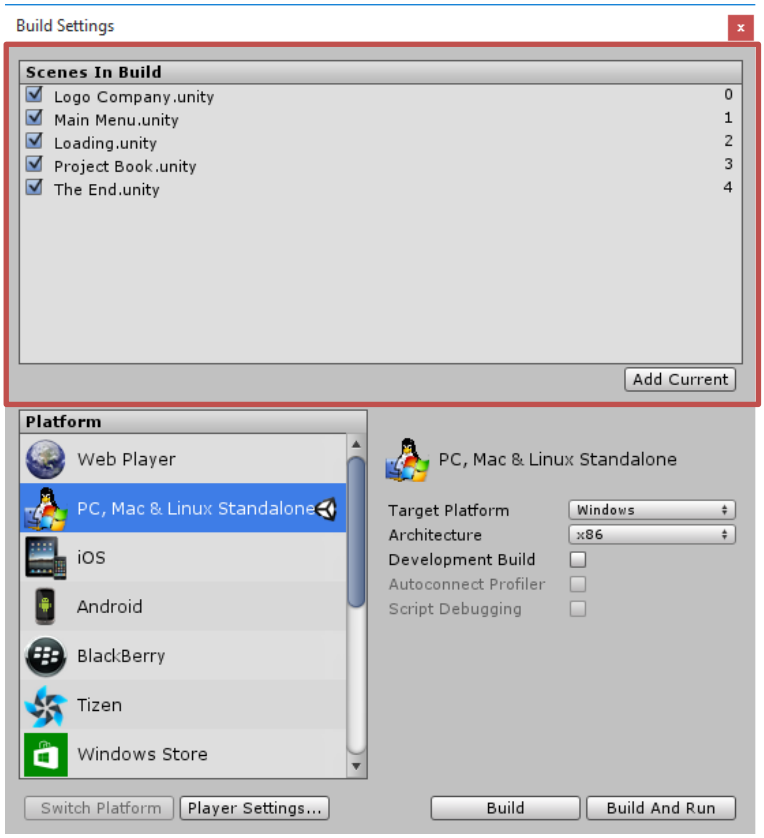


شيء ثاني يجب وضع العنصرين Slider و Text في المتغيران اللذان يمثلاهما:

و بهذا الشكل نكون قد انهينا العمل على اعدادات الجودة ففي حال قمت بتشغيل اللعبة ستجد الاعدادات و يمكنك تغييرها عبر الشريط.

ترتيب المشاهد لتطبيق المستويات:

في هذه الفقرة ماسنقوم به هو ترتيب المشاهد لرؤية عمل المستويات فقط، العملية بسيطة فهي تعتمد على سحب المستويات الى قائمة الـ Scenes In Build ويمكنك الوصول اليها من File> Build Setting وستظهر لك هذه القائمة:



لاحظ القائمة المحددة تجد انني قمت بترتيب المشاهد (في حالة لم تكن تعلم ماهو مشهد Logo Company و The End فلا تقلق سأقوم بشرحهم لاحقاً) وتجد ارقام المشاهد و حال الضغط على زر تطبيق المستوى في اللعبة علينا ادخال رقم المشهد كما هو موضح فمثلاً المستوى الاول تجده رقم 3 و اضافة المشاهد تكون اما بسحب المشهد الى القائمة او بإضافته من الزر Add Current.

في حاله قمت باضافة المشاهد عد الى القائمة Select Level و غير ارقام ازرار المستويات الى مايتناسب مع المستوى الذي تحدده، المستوى الثاني سيكون رقم 4 و المستوى الثالث الرقم الذي يليه و هاكذا.

المشهد Loading رقم 2 بحيث انه عند الضغط على Start او الفوز بالمرحلة سيعيدنا الية تلقائياً لكي يحمل المستوى القادم.

هناك بعض التعديلات التي يجب ان نجريها لكي يتم تطبيق المستويات بشكل جيد، في الملف MenuManager احط هذا الكود بهذا الشريط ويفضل ان يكون منفصل عن باقي الشروط:

```
1.  if(typeMenu == TypeMenu.loading || typeMenu == TypeMenu.main)
2.  {
3.      if(PlayerPrefs.GetInt("Score") < 3000)
4.          NamNextLevel = 3;
5.
6.      if(PlayerPrefs.GetInt("Score") >= 3000)
7.          NamNextLevel = 4;
8.
9.      if(PlayerPrefs.GetInt("Score") >= 6000)
10.         NamNextLevel = 5;
11.
12.     if(PlayerPrefs.GetInt("Score") >= 9000)
13.         NamNextLevel = 6;
14.
15.     if(PlayerPrefs.GetInt("Score") >= 12000)
16.         NamNextLevel = 7;
17.
```

```
18.                     if(PlayerPrefs.GetInt("Score") >= 15000)
19.                             NamNextLevel = 9;
20.             }
```

من خلال هذا الكود تجد اننا لم نعد نستخدم المتغير ContinueLevel و استبدلناه بمتغير NamNextLevel و سيتم تطبيق المستوى القادم مع تحميلة و تجد ان هذا الكود يعمل في قائمة التحميل و القائمة الرئيسية لكي نستطيع الوصول الية في قائمة التحميل.

الان اجعل الدالة LoadingNextLevel تحميل المستوى NamNextLevel بشكل مباشر (NamNextLevel).

بالنسبة للدالة Continue قم بوضع بارامتر من نوع String من خلاله ننتقل الى قائمة التحميل لكي نقوم بتحميل آخر مستوى وصلنا الية و شكل الدالة سيكون:

```
1.      public void Continue(string nameLevel)
2.      {
3.              Application.LoadLevel(nameLevel);
4.      }
```

اما الدالة الاولى سيكون شكلها:

```
1.      public IEnumerator LoadingNextLevel()
2.      {
3.              AsyncOperation loading = Application.LoadLevelAsync(NamNextLevel);
4.              while(!loading.isDone)
5.              {
6.                      SliderLoading.value = loading.progress * 100;
7.                      yield return null;
8.
9.              }
10.     }
```

بالسبة للدالة StartGame يجب ان تحتوي على بارامتر من نوع int ينقلها الى المستوى الاول و ليس قائمة التحميل يعني مباشرةً الى المستوى الاول.

```
1.      public void StartGame(int i)
2.      {
3.              Application.LoadLevel(i);
4.      }
```

حسناً تبقى لنا شيء اخير وهو العودة الى قائمة النجوم و يجب علينا وضع الدوال لكن سنطبق الامر على قائمة واحدة و من ثم انقلها الى باقي القوائم.

بالنسبة لزر الانتقال الى المستوى التالي في القائمة الذهبية و العادية قم باضافة الدالة NextLevel الى قائمة الاحداث و هي ستعود الى قائمة التحميل لتطبيق المستوى القادم.

اما زر اعادة اللعبة فبكل بساطة اضف الدالة RestartLevel الى قائمة الاحداث.

اخيراً زر العودة الى القائمة الرئيسية قم الدالة BackToMainMenu الى قائمة الاحداث.

انشاء مدير الصوتيات AudioManager:-

ان مدير الصوتيات مهم جداً فهو الذي سيتحكم بالاصوات في المشهد بشكل عام (حال اللعب) و سيعتمد على القيم المعطى سابقاً من ملف MenuManager لكي نربط بعظها ببعض. العملية سهله و لكن الملف الحالي سنعرف فيه فئة جديدة تمثل الاصوات في المشهد فقط بحيث نفصلها عن الاصوات في المستوى عن الموجودة في القائمة الرئيسية لأننا سنستخدم الملف في كلا المشهدين.

حسناً، لكي نعمل على الملف بشكل جيد قم بالعودة الى اللعبة اعني المرحلة الاولى و اتبع النقاط التالية:

- انشئ ملف برمجي باسم AudioManager وضعة في الكائن Audio:
- في هذا الملف يجب علينا استيراد انطقه الاسم الاتية:

```
using System.Collections.Generic;
using UnityEngine.UI;
```

بما اننا نستخدم الملف في القائمة الرئيسية و مراحل اللعبة علينا تعريف enum يمثل المستويات، اضف المتغيرات:

```
1.  public enum TypeSounds
2.  {
3.      level,
4.      menu
5.  }
6.
7.  public TypeSounds typeSounds;
```

ايضاً تعريف اثنان List<> واحدة تمثل الاصوات في المراحل و الثانية تمثل الاصوات في القوائم، اضف المتغيرات التالية:

```
1.  [Header("Audio In Level")]
2.  public List<AllAudio> allSoundsInLevel;
3.
4.  [Header("Audio In Manu")]
5.  public List<AudioSource> SoundsMenu;
```

الـ Header يوضح لنا معنى المتغيرات لكن ستجد ان الفئة AllAudio غير موجودة هي الفئة التي سنقوم بتعريفها في الملف، اذن اضف هذه الفئة و يجب ان تكون خارج اي فئة اخرى اي انها فئة منفصلة:

```
1.  [System.Serializable]
2.  public class AllAudio
3.  {
4.      public Toggle toggleSounds;
5.      public List<AudioSource> SoundsGame;
6.      public AudioSource MusicGame;
7.  }
```

لاحظ اننا عملنا Serializable للفئة وهذا سيتيح لها التسلسل في الملف الحالي و الظهور في الانسباكتور ولكن الامر لا يكون بهذه الطريقة بل عبر المتغير allSoundsInLevel فهو الذي يظهر الفئة.

فيها قمت بتعريف متغير يمثل الـ Toggle الموجود في القائمة المصغرة، قائمة تمثل الاصوات في المشهد و متغير اخير يمثل الموسيقى.

علينا الوصول الى المتغيرات عبر الحلقة foreach، اضف هذه الدالة الملف و يجب ان تكون داخل الفئة AudioManager وليس AllAudio:

```
1.  void SoundsInLevel()
2.  {
3.      foreach(AllAudio allsounds in allSoundsInLevel)
4.      {
5.          if(allsounds.toggleSounds.isOn)
6.              PlayerPrefs.SetString("Sounds","true");
7.
8.          if(!allsounds.toggleSounds.isOn)
9.              PlayerPrefs.SetString("Sounds","false");
10.
11.         foreach(AudioSource sounds in allsounds.SoundsGame)
12.         {
13.             if(PlayerPrefs.GetString("Sounds") == "true")
14.             {
15.                 sounds.enabled = true;
16.                 allsounds.toggleSounds.isOn = true;
17.             }
18.             if(PlayerPrefs.GetString("Sounds") == "false")
19.             {
20.                 sounds.enabled = false;
21.                 allsounds.toggleSounds.isOn = false;
22.             }
23.         }
24.     }
25. }
```

قبل الولوج الى تفصيل الدالة دعني واضح لك مفهومها ليتسنى لك قراءتها وفهمها، الدالة وظيفتها هي ايقاف الاصوات في مراحل اللعبة وتستخدم المتغيرات في الفئة المعرفة فهي تقوم بتوقيف الـ Toggle الخاص بالقائمة المصغرة اعتماداً على المتغير Sounds.

لاحظ في السطر 3 استخدمت الحلقة foreach للوصول الى الفئة و من خلالها الان نستطيع الوصول الى الفئة عبر المتغير allsounds و لاحظ انني اولاً وضعت قيم لتشغيل و ايقاف الـ Toggle و سيتم حفظها وهي على شكل String، اما في السطر 11 استخدمت الحلقة مجدداً و لكن الغرض منها هو تطبيق الامر على جميع الاصوات بدل ان احددها على شكل مصفوفه لهذا اي امر سيحصل للمتغير sounds سيطبق على جميع الاصوات في المشهد ماعدا الموسيقى.

لاحظ في الحلقة ان تشغيل الاصوات و ايقافها يعتمد على حالة الـ Toggle الموجود في القائمة المصغرة، اما true و تعمل الاصوات او false و تتوقف الاصوات.

ان الامر بسيط فكل ما علينا فعلة الان هو استدعاء الدالة في Update:

```
1.      PlayerPrefs.Save();
2.      if(typeSounds == TypeSounds.level)
3.          SoundsInLevel();
```

لاحظ انها محاطة بالشرط فهي لن تعمل الا في مراحل اللعبة فقط.

تبقى لنا تعريف القيم الاولى عند بدأ تشغيل المرحلة وهي تخص تشغيل الاصوات و الموسيقى وقيمة المتغير Volume اعتماداً على قيمة الـ Slider الخاص بالاصوات و الموسيقى.

- اولاً: الاصوات يجب علينا توقفها و تشغيلها اعتماداً على حالة العنصر Toggle الخاص بالقائمة المصغرة او القائمة الرئيسية و من ثم نحيط على جميع الاصوات و نقوم بجعل نسبة الـ Volume الخاص بهم مع نسبة الـ Slider الذي يخص الاصوات، اضف الكود التالي:

```
1.  void Start()
2.  {
3.          if(typeSounds == TypeSounds.level)
4.          {
5.                  foreach(AllAudio allsounds in allSoundsInLevel)
6.                  {
7.                          if(PlayerPrefs.GetString("Sounds") == "true" || PlayerPrefs.GetString("SoundsIs") == "true")
8.                                  allsounds.toggleSounds.isOn = true;
9.                          if(PlayerPrefs.GetString("Sounds") == "false" || PlayerPrefs.GetString("SoundsIs") == "false")
10.                                 allsounds.toggleSounds.isOn = false;
11.
12.                         foreach(AudioSource sounds in allsounds.SoundsGame)
13.                                 sounds.volume = PlayerPrefs.GetFloat("SliderSounds");
14.                 }
15.         }
16. }
```

حسناً دعنا نناقش الكود، بدايةً الكود سيعمل في مراحل اللعبة فقط، و تجد داخلة ان الحلقة foreach تحيط على جميع الشروط بحيث نتعامل مع متغيرات الفئة AllAudio منها يجب علينا استخدام القيم المحفوظة للاسم Sounds و الذي يحدد لنا تشغيل و ايقاف الاصوات ولكن سنستخدمه لإعطاء الـ Toggle الخاص بالقائمة المصغرة قيمة الـ Toggle الخاص بالقائمة الرئيسية، مثلاً الموجود في القائمة الرئيسية كانت قيمتة false سيجعل الموجود في القائمة المصغرة قيمتة false ايضاً و هذا سيؤدي الى توقيف جميع الاصوات في المشهد، و تلاحظ في الشرط انه يعتمد اما على القيمة الحالية او على القيمة المعطاة في القائمة الرئيسية وهي تعتمد على الاسم SoundsIs وهو من نوع String.

في السطر 12 قمت بعمل حلقة تدور على جميع الاصوات في المشهد و تجعل قيمة الـ Volume الخاص بها تتناسب مع قيمة الـ Slider الموجود في القائمة الرئيسية عبر الاسم SliderSounds الذي يخزن قيمة الشريط الخاص بالاصوات.

- ثانياً: الموسيقى ومفهومها سهل هو ان تتوقف او يكون قيمة المتغير Volume الخاص بها يتناسب مع قيمة الشريط الخاص بالاصوات.

في الحلقة السابقة في الدالة Start اضف الكود التالي:

```
1.          allsounds.MusicGame.volume = PlayerPrefs.GetFloat("Music");
2.
3.          if(PlayerPrefs.GetString("MusicIs") == "true")
4.                  allsounds.MusicGame.enabled = true;
5.          if(PlayerPrefs.GetString("MusicIs") == "false")
6.                  allsounds.MusicGame.enabled = false;
```

اذن لاحظ معي ان العملية مقسمة الى قسمان [(تشغيل و ايقاف الموسيقى) و (اناقص وزيادة صوت الموسيقى)] بحيث ان السطر الاول يقوم بجعل المتغير Volume ستناسب مع قيم الشريط الخاص بالموسيقى.

في الشرطين الاخيرين يقوم بتحديد هل الموسيقى ستعمل ام لا؟ و هذا الامر يعتمد على حالة الاسم MusicIs.

بهذا الشكل نكون قدا انهينا العمل على اصوات و موسيقى المراحل.

تبقى لنا برمجة الاصوات الخاصة بالقوائم وهي اصوات الازرار فقط لهذا اضف الدالة التالية لكي نناقشها:

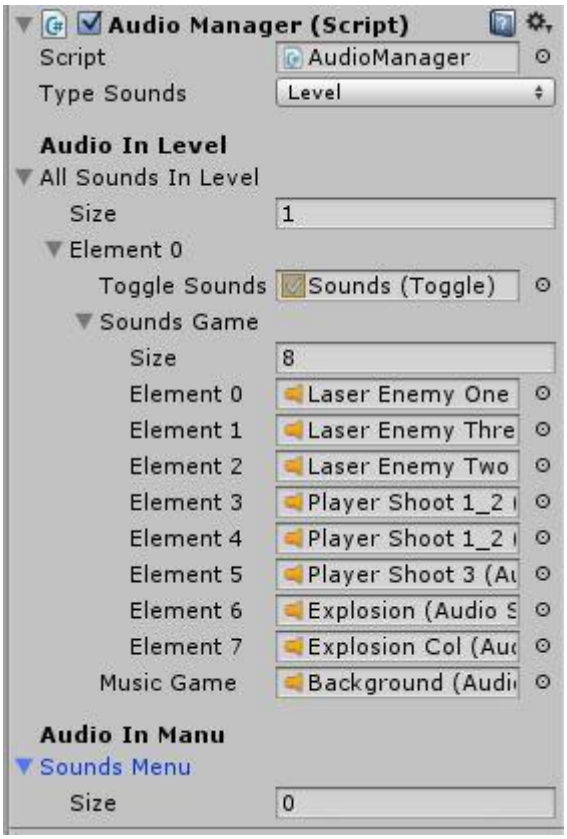
```
1.  void SoundsInMenu()
```

```
2.  {
3.          foreach(AudioSource sounds in SoundsMenu)
4.          {
5.                  if(PlayerPrefs.GetString("Sounds") == "true" || PlayerPrefs.GetString("SoundsIs") == "true")
6.                          sounds.enabled = true;
7.                  if(PlayerPrefs.GetString("Sounds") == "false" || PlayerPrefs.GetString("SoundsIs") == "false")
8.                          sounds.enabled = false;
9.
10.                 sounds.volume = PlayerPrefs.GetFloat("SliderSounds");
11.         }
12. }
```

الدالة بسيطة جداً فمفهومها هو ايقاف و تشغيل الاصوات او انقاص الـ Volume الخاص بها، لاحظ الشريطين فهما يعتمدان على حالة الاسم Sounds الخاص بالـ Toggle الموجود في القائمة المصغرة او الاسم SoundsIs الخاص بالـ Toggle الموجود في القائمة الرئيسية و سيتم اعطاء القيمة التي تدل على ذلك.

يعتمد المتغير Volume على القيمة التي يخزنا الاسم SliderSounds الشريط الخاص بالاصوات، اخيراً قم باستدعاء الدالة في Update و يجب عليها ان تعمل في القوائم فقط اي النوع menu.

بالعودة الى المشروع ستجد القائمتان، بالنسبة لقائمة Sounds Menu فاتركها فارغة. اما القائمة All Sounds In Level فاجعل حجمها 1 و ستجد ان متغيرات المصفوفة قد ظهرت فقم بملئها:



بالنسبة للقوائم قم بسحب الملف الى اي كائن و اجعل حجم القائمة Sounds Menu هو 3 بحيث يتناسب مع عدد الاصوات في القوائم و اسحب الاصوات الى الخانات، اما القائمة All Sounds In Level اتركها فارغه ولا تفتحها فلن نحتاجها هنا.

اذن بهذا الشكل نكون قد انهينا العمل على ملف AudioManager والذي اعتبره ملف مهم جداً في اللعبة.

تصميم المراحل:-

بما اننا انتهينا من برمجة الاشياء الاساسية و المهمة في اللعبة سننتقل الى تصميم المراحل و مفهوم كل مرحلة سهل جداً فهو عبارة عن حفظ لنفس المرحلة و لكن مع تغيير الاسم و الخلفية (المباني و الاشياء الجانبية) و برمجة حركة الكاميرا و يجب ان تكون في المرحلة الثالثة وما فوق.

التعديل على الملفات البرمجية:

علينا العودة على بعض الملفات البرمجية لكي نقوم بالتعديل عليها فهنا نقوم بمسح الغبار عن الملفات ونترك معماريتنا متاحة للتعديل في اي وقت و هذا الشيء سيساعدنا على وضع اكثر من مستوى و التعديل العديد من الاشياء.

- الملف EnemiesInfo و الذي يجب علينا احاطة بعض الشروط بشروط اخرى مثل وضع الاضافات و هي خاصة فقط بالأعداء ماعدا زعيم الاعداء اي الزعيم لن يقوم بوضع اي اضافة في المشهد حال موتة كما قلت سابقاً، اذن تأكد من ان الشروط في الدالة EnemiesHelpPlayer هي بهذا الشكل:

```
1. if(!gameObject.CompareTag("Boss"))
2. {
3.         // اضف جميع الشروط هنا
4. }
```

اذن المفهوم سهل فكل ما يقوم به هذا الشرط هو التحقق من ان الاعداء هم من سيضعون الاضافات حال موتهم ماعدا الزعيم وفي الشرط قم اضافة الأكواد السابقة.

شيء آخر وهو تعريف متغيرات تمثل اعطاء قيمة عشوائية للنقاط، نحن وضعناها سابقاً ولكن سنستبدلها بمتغيرات، اذن اضف المتغيرات التالية:

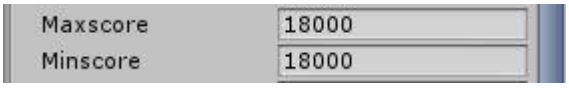
```
1. [SerializeField] protected int maxscore;
2. [SerializeField] protected int minscore;
```

لاحظ ان المتغير من نوع int فهي تتناسب مع نوع المتغير maxScore، اخيراً عدل على كود اضافة النقاط بحيث يصبح كالاتي:

```
1. GameManager.maxScore += Random.Range(minscore, maxscore);
```

في انسباكتور قم بملء القيم كما هو موضح:

اما زعيم الاعداء:



و السبب ان زعيم الاعداء سيكون في آخر مرحلة وهذه المرحلة ستكون عدد نقطها هي 18000 ففي حالة قتل الزعيم ستنتهي اللعبة فهنا لن تكون الاعداء قادرة على وضع النقاط ماعدا الزعيم حال موتة.

- في ملف EnemiesManager علينا تعريف متغير يمثل الوقت المتبقي لوضع العدو التالي لكي نستطيع زيادة سرعة اضافة الاعداء في كل مستوى، اذن قم باضافة المتغير التالي:

```
2.  public float timeAddEnemies = 2.5f;
```

الان في حالة اضاف عدو قم بجعل الوقت مساوي للوقت الحالي اما في انسباكتور ضع قيمة الوقت التي تراها مناسبه.

اضواء المباني:

بالنسبة للمباني فأننا تحدثنا عنها سابقاً وهنا سنقوم ببرمجتها، لكن قبل البرمجة يجب ان ترى ملف Lights وتجده في الملف Background وفيه صورتان لاثنان مباني ويمكنك رؤيتهما، ان علينا جعل المباني تتغير الى الصور المضيئة حال الفوز في اللعبة و هذا الامر يعتمد على انشاء ملف برمجي سنسميه LightsHouses اذن قم بأنشاء هذا الملف.

بدايةً علينا تعريف متغير من نوع Sprite يمثل الصور الحالية و كل مبنى سيحمل صورة، اذن اضف المتغيرات التالية:

```
1.  public Sprite sprite;
2.  private SpriteRenderer s_renderer;
3.  private GameManager gameManager;
```

سنستعين بالمتغير gameManager للوصول الى المتغير HightScore.

في Start يجب علينا الحصول على المكون SpriteRenderer، و البحث عن الكائن الذي يحمل الملف GameManager، لاحظ الكود:

```
1.  void Start () {
2.          s_renderer = GetComponent<SpriteRenderer>();
3.          gameManager = GameObject.Find("Game").GetComponent<GameManager>();
4.  }
```

اما في Update سنقوم بوضع الصورة عند وصول النقاط الى ذروتها:

```
1.  void Update () {
2.          if(GameManager.maxScore >= gameManager.HightScore)
3.                  s_renderer.sprite = sprite;
4.  }
```

جيد، الان قم بسحب هذا الملف البرمجي الى جميع المباني و في كل مبنى اسحب صورة المبنى المضاء الذي يمثله و حال الفوز في اللعبة ستعمل الاضواء.

حركة الكاميرا:

تحريك الكاميرا سترتب عليه توسيع المشهد الذي سيتيح للكاميرا التحرك فيه، العملية الاولية بسيطة و هي توسيع حدود التحرك للمدفع وتوسيع المشهد لهذ قم بنسخ الخلفيات و المباني مع اجزاء تعديلات بسيطة حولها بحيث تصبح كالاتي:

لاحظ مدى المساحة المتاحة لتحرك المدفع فيها و يجب عليك تمديد الحدود الى ان تراها مناسبة، بالنسبة للكاميرا ستكون في نهاية المشهد و حال تشغيله ستتحرك الى اللاعب وهي تتبع اللاعب فكلا الحالتين.

لتحريك الكاميرا علينا انشاء ملف برمجي و نشميته CameraController او اي اسم دو معنى و سحب الملف الى الكاميرا.

حسناً دعنا نتحدث عن حركة الكاميرا، ان الكاميرا تتحرك بشكل افقي (على المحور x) بينما باقي المحاور تظل على موقعها و لن تتغير، توقيف الكاميرا سيعتمد على وصول النقاط لذروتها، العملية بسيطة جداً اذن قم باضافة المتغيرات التالية الى الملف البرمجي:

```
1.  public Transform Player;
2.  private Vector2 offset;
3.
4.  public float maxPos;
5.  public float minPos;
6.  public float Speed;
7.  private GameManager gameManager;
```

لاحظ انني قمت بتعريف العديد من المتغيرات و اذكر منها: متغير يمثل اللاعب، آخر يمثل محاور الكائن، متغيرين يمثلان حدود تحرك الكاميرا ايضاً واحد للسرعة وخير للوصول الى الفئة GameManager و لكي نصل الى المتغير maxScore.

قم باضافة الدالة التالية ووظيفتها تحريك الكاميرا مع وضع الحدود:

```
1.  void movementCamera(bool move)
2.  {
3.          if(move)
4.          {
5.                  transform.position = new Vector3
6.                          (Mathf.Lerp(transform.position.x, Player.position.x, Speed* Time.deltaTime), 0, -10);
7.
8.                  if(transform.position.x >= maxPos)
9.                          transform.position = new Vector3(maxPos, 0, -10);
10.                 if(transform.position.x <= minPos)
11.                         transform.position = new Vector3(minPos, 0, -10);
12.         }
13. }
```

ان لدالة تعيد bool و القيمة true تقوم بتشغيل الكود و الامر بسيط، في الشرط نقوم بنقل الكاميرا من موقعها الحالي الى موقع اللاعب على محور x عبر الدالة Lerp فهي تطلب 3 بارامترات من نوع float تحدد الانتقال من الموقع الحالي، الى الموقع المطلوب و اخيراً تحديد سرعة للانتقال فمن خلاله ستجد ان الكاميرا تتحرك مع وجود نوع من الانزلاق الى اللاعب.

اخيراً الشرطان يقومان بعمل حدود معينه لتحرك الكاميرا فلاحظ اعتمادي على المتغيرات التي تمثل اعلى حدود و الاقل.

الان في LateUpdate يجب علينا تشغيل الدالة و لكن بعد احاطتها بشرط، سبب استخدامي لهذا النوع من دوال الـ Update هو للتأكد من تحرك اللاعب قبل الكاميرا. اذن اضف الكود التالي:

```
1.  void LateUpdate()
2.  {
3.          if(GameManager.maxScore >= gameManager.HightScore)
4.          {
5.                  movementCamera(false);
6.          }else movementCamera(true);
7.  }
```

لاحظ ان حال وصول النقاط لأعلى قيمة لها تتوقف الحركة غير ذلك ستعمل الحركة، تبقى لنا شيء اخير وهو البحث عن الكائن Game الذي يحوي على المكون GameManager، اضف الدالة Start:

```
1)	void Start()
2)	{
3)		gameManager = GameObject.Find("Game").GetComponent<GameManager>();
4)	}
```

بهذا الشكل نكون قد انهينا العمل على الكاميرا فكل ما تبقى لنا هو وضع الملف فيها و اعطائها القيم المناسبة بالاضافة الى وضع الحدود التي تراها مناسبة مع المشهد ولا تنسى سحب اللاعب الى المتغير الذي يمثله ☺ .

هنا اخيراً نكون قد انهينا برمجة اللعبة فكل ما تبقى لنا هو التعديل على مشاهد اللعبة و فحصها.

مشاهد اللعبة:

بما اننا سنعمل 6 مراحل فقط هنا يجب علينا مناقشة بعض الامور الاساسية مثل انواع الطلقات و الاعداء و الاضافات في المشاهد لهذا سأقوم بوضعها على شكل نقاط بحيث ان كل نقطة تعبر عن حالة المرحلة:

- مشهد شعار اللعبة.
- مشهد القائمة الرئيسية.
- مشهد قائمة تحميل المراحل.
- المستوى الاول: استعمال الطلقة الاولى، قتال العدو الاول، اضافات الطلقة الاولى و مساحة تحرك محدودة، عدد اقصى نقاط هي 3000.
- المستوى الثاني: استعمال الطلقة الاولى، قتال العدو الاول، اضافات الطلقة الاولى و مساحة تحرك محدودة، عدد اقصى نقاط هي 6000.
- المستوى الثالث: استعمال الطلقة الاولى و الثانية، قتال العدو الاول و الثاني، اضافات الطلقة الاولى و الثانية، مساحة التحرك مفتوحة، حركة الكاميرا، اقصى عدد نقاط هي 9000.
- المستوى الرابع: استعمال الطلقة الاولى و الثانية، قتال العدو الاول و الثاني، اضافات الطلقة الاولى و الثانية، مساحة التحرك مفتوحة، حركة الكاميرا، اقصى عدد نقاط هي 12000.
- المستوى الخامس: استعمال جميع الطلقات، قتال جميع الاعداء، اضافات جميع الطلقات، مساحة التحرك مفتوحة، حركة الكاميرا، اقصى عدد نقاط 15000.
- المستوى السادس: استعمال جميع الطلقات، قتال جميع الاعداء مع الزعيم، اضافات جميع الطلقات، مساحة التحرك مفتوحة، حركة الكاميرا، اقصى عدد نقاط 18000.
- مشهد نهاية اللعبة.

حسناً هذه العشر مراحل ستساعدنا على استخدام الاعدادات الموجودة معنا او تقسيمها بهذا الشكل اراه جيد، بالنسبة للمرحلة الاخيرة سيتوجب علينا التعديل على النقاط المكتسبة من قتل الاعداء لهذا بشكل بسيط سنقوم بتعريف متغيرين من نوع float يمثلان اقصى عدد و اقل عدد للنقاط.

اضن ان شرح تغيير الخلفيات مفهوم سابقاً فملف Background يحوي العديد من الصور التي يمكن استخدامها، لكننا هنا سنستخدم الصور في الملف Lights و التي تمثل اضواء المباني و هذا يدل على انك قد فزت في اللعبة و تعمل اضواء المباني، العملية سهله فهي تعتمد على انشاء ملف برمجي فقط.

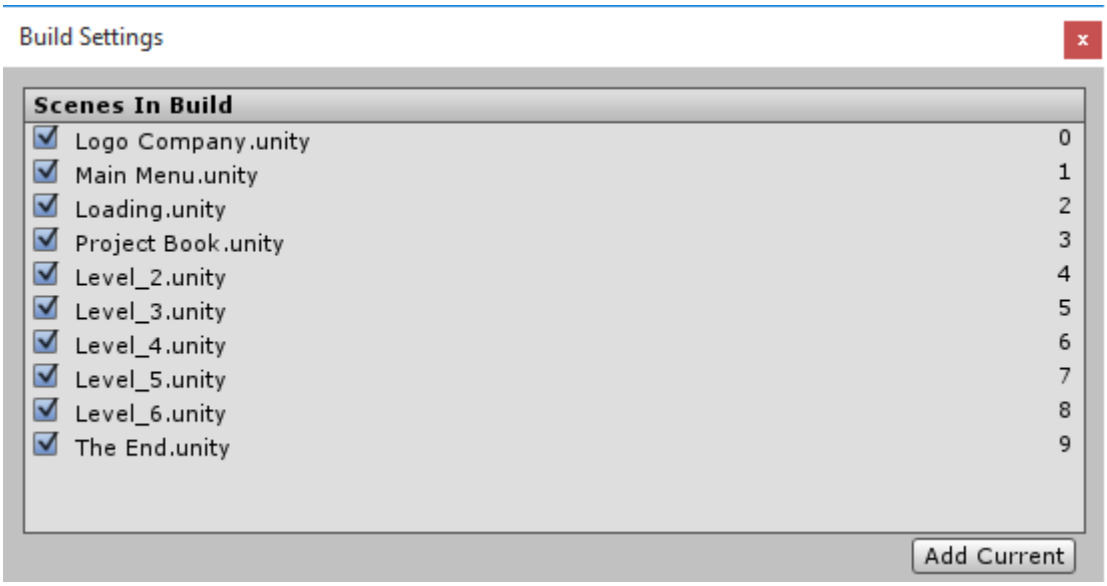
بالنسبة للأعداء في المستوى الاخير و الذي ستقاتل فيه زعيم الاعداء وحال قتلة ستصل النقاط الى ذروتها و لكن هنا عند قتل الاعداء ستزيد عدد النقاط و هذه مشكلة، ولحلها نقوم بنسخ الاعداء من prefabs و تسميتهم بأسماء اخرى و جعل قيمة maxScore تساوي الصفر حال موتهم ماعدا الزعيم و بهذا الشكل نكون قد جعلنا الزعيم هو محور القتال في اللعبة.

في مشهد نهاية اللعبة غالباً يتم عرض المطورين الذين عملوا على اللعبة لهذا ليس هناك الكثير لنضيفه لهذا لك حرية وضع اي شيء في المشهد الاخير ففي النهاية هو يذكر المطور و الوظائف التي قام بها لبناء اللعبة.

شعار اللعبة امر ضروري لمعرفة الاستوديو الذي قام بتطوير اللعبة ففي حال لم يكن لديك شعار معين فبكل بساطة ضع اسمة ادا اردت.

انشاء المراحل الست سيترتب عليها ترتيب المراحل بشكل تسلسلي بحيث ان كل مرحلة تزيد الصعوبة فيها ومن خلال النقاط السابقة التي ذكرتها و لك حرية توسيع الآمر.

اخيراً في القائمة Build Setting يجب ان ترتب المشاهد بهذا الشكل:



المشهد project Book هو المستوى الاول الذي عملنا علية طول هذه الفترة وباقي المشاهد تأتي بعدة الى المشهد الاخير.

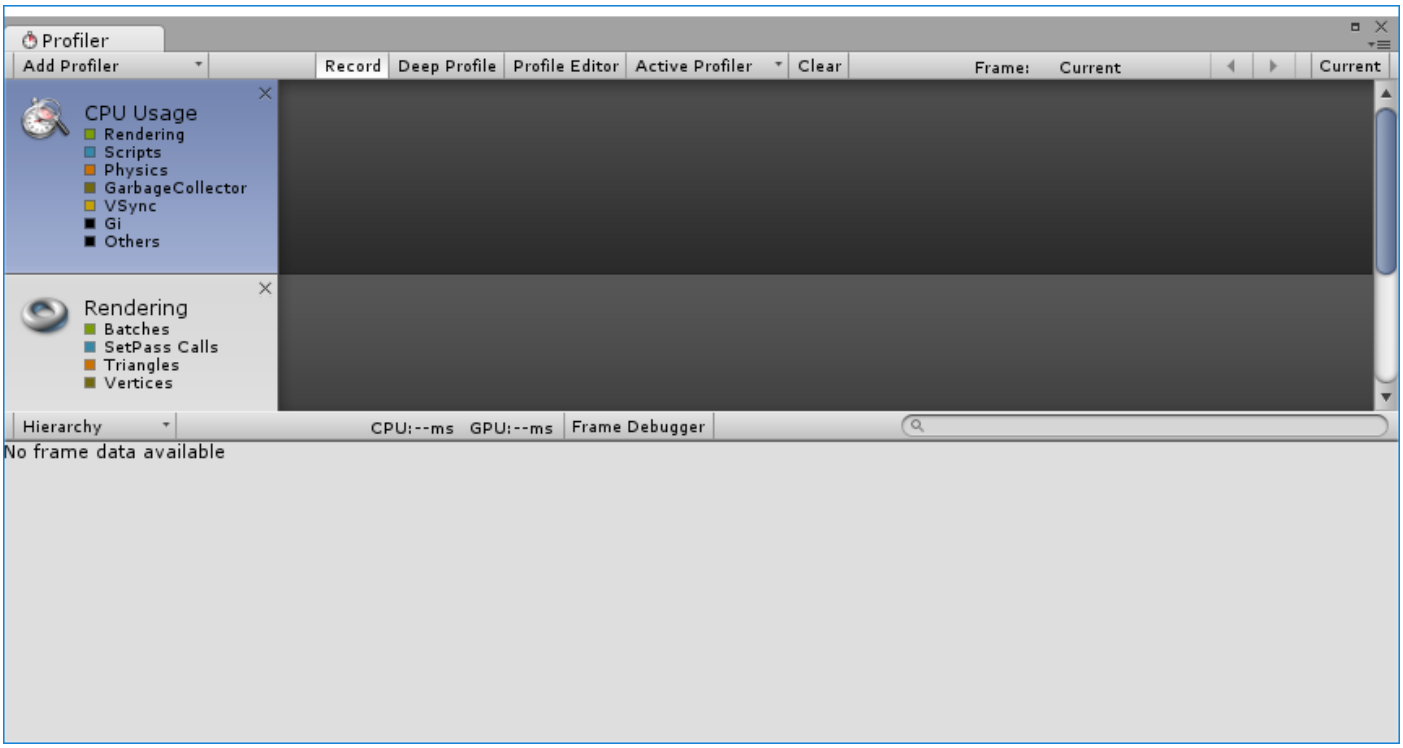
في المستوى السادس يجب وضع زعيم الاعداء في المشهد بينما اللعبة و الزعيم سيولدون الاعداء التي لاتعطي نقاط و بهذا الشكل تقاتل الاعداء لقتل زعيم الاعداء و لهذا يجب ايضاً اطلاق النار على الزعيم لقتلة.

تحسين اللعبة (Optimization):

ان عمل تحسين للعبة هو امر مهم جداً ففي المشاريع الكبيرة يتم التأكد من ان العمل يسير على افضل حال و عمل تحسين للعبة يساعدك على معرفة الموارد التي تستهلك الكثير في اللعبة و اين يضيع الوقت ومدى الاشياء التي تخزنها ذاكر اليونتي و التي تتيحها للعبة، تحسين اللعبة هو امر مهم ففي وخاصة عند تصدير اللعبة على الهواتف التي تتميز بضعف ذاكرتها و ادائها بخلاف الحاسوب، و تحسين اللعبة في فترات متفاوتة اثناء اللعبة يساعدك على معرفة التغيرات التي اجريتها وهل تضغط على اداء اللعبة اثناء عملها على انظمة الهواتف ام لا، هذا الامر يقتصر على الهواتف فقط بل في الالعاب الكبيرة يتم عمل تحسين للعبة للتأكد من عملها على اجهزة الكمبيوتر بشكل صحيح، فلا حاجة للعبة بسيطة ذات موارد سيئة تتطلب مواصفات علية و سبب هذا هو سوء استخدام هذه الموارد في اللعبة و هذا يؤدي الى عمل تحسين جذري للعبة.

الـ Profiler:

اليونتي يوفر لنا احد اقوى الادوات لرؤية اداء اللعبة من جميع النواحي و بتفاصيل دقيقة جداً وهي Profiler وتجدها في الشريط العلوي في القائمة Window ففي حال قمت بفتحه ستظهر لك هذه النافذة:



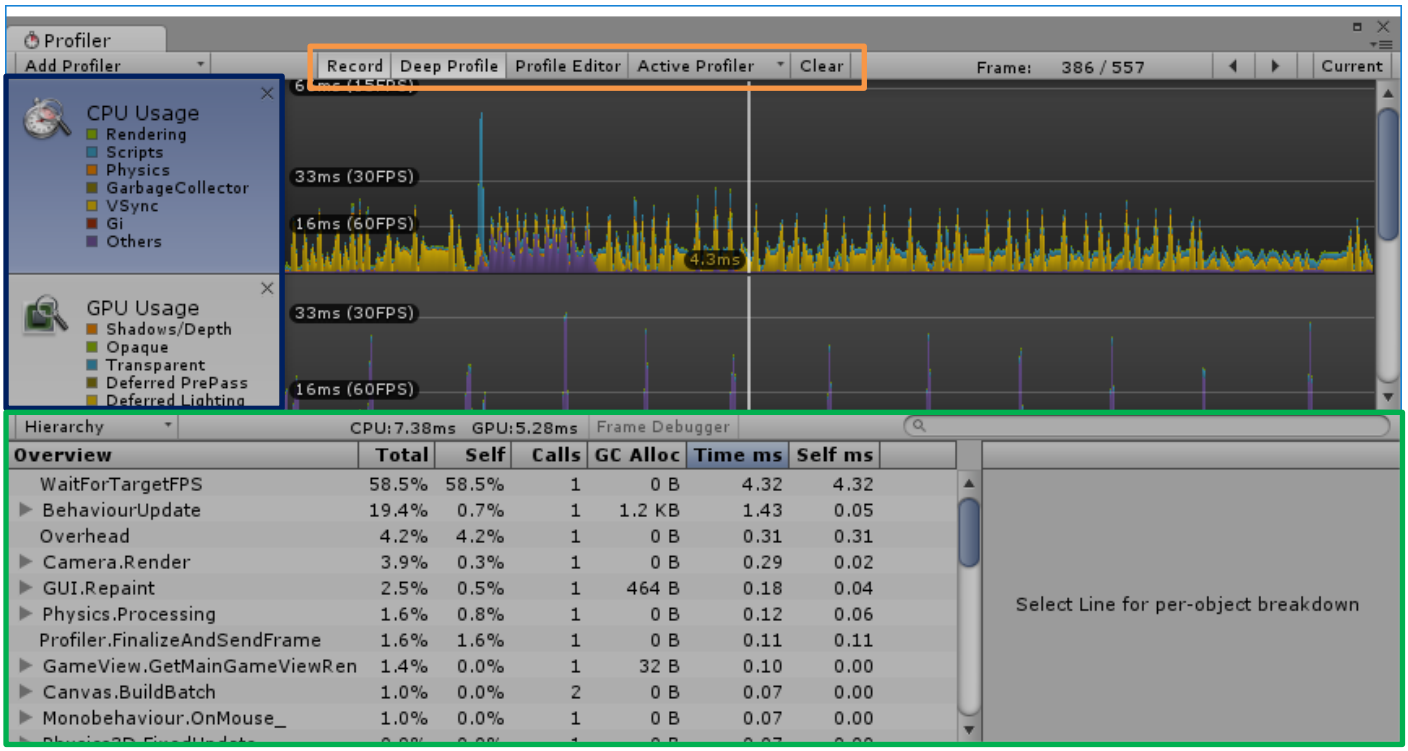
دعنا نتحدث اكثر عن الـ Profiler ان من اهم ميزات هذه الاداة انها تقوم بتسجيل حركة اللعبة وتعرض لك الاشياء التي تعمل في اللعبة من مكونات، مواد كائنات، ملفات ... الخ و تتيح لك فحص اللعبة فريم فريم مع عدة خصائص اضافية، لايقتصر امر تشغيلة في اللعبة فقط بل ايضاً يمكنك ربط في اللعبة حال بنائها وتعرض البيانات في كلا الحالتين على شكل مخطط بياني ويمكنك الانتقال الى اي فريم لرؤية الاشياء التي عملت فيه.

ايضاً يقوم بعرض الملفات البرمجية التي تستهلك الكثير من الوقت لعملها و يعطيك نسب مئوية و الوقت المستغرق لعمل هذا الملف مع خيارات اضافية و تستطيع رؤية جميع الملفات و الدوال التي تعمل اثناء اللعبة و في اي اطار تريده.

عدم عمل تحسين للعبة قد يؤدي الى اعطاء اللعبة معدل اطارات سيء فيمكنك رؤية بعض الاشياء الاساسية مثل الاطارات و غيرها بشكل سريع من الخيار Stats في النافذة Game و هذا يعطيك لمحة عن معدل الاطارات و للحرص على بقائها تعمل بشكل جيد يفضل اختبار اللعبة في اصعب حالاتها وظروفها للتأكد من عدم نزول معدل الاطارات كثيراً.

واجهه الـ Profiler:

لنناقش واجهه البروفايلر ككل، ان الواجهة السابقة تحوي العديد من الاشياء و لكن دعنا نعتمد الصورة التالية للمزيد من التفاصيل:



- الشريط العلوي في النافذة يتيح لك رؤية العديد من الخيارات او بالأصح تفعيل الخيارات التي تساعدك على رؤية التفاصيل الدقيقة مثل (رؤية المعلومات في كل فريم، فحص جميع الموارد، مسح المخطط البياني).

١) Deep Profiling: حال تفعيل هذا الخيار فهو يقوم بتسجل جميع الوظائف التي تعمل في اللعبة وهو مفيد لمعرفة اين يضيع الوقت و خاصة في الملفات البرمجية الخاصة بك. هذا الخيار يقوم بالتعمق في الفصح بشكل كبير بحيث يفحص كل صغيرة و كبيرة في اللعبة ومعدل استخدامه للذاكرة هو كبير جداً و قد تلاحظ بطئ شديد في اللعبة و السبب هو ما يقوم به هذا الخيار من عرض المعلومات الكبيرة. تعمق فحصة لايقتصر على الموارد فقط بل على الملفات البرمجية بشكل كبير فهو يعرض لك جميع الدوال و التي تستهلك الاطارات بشكل كبير واستخدامه مفيد جداً، في حالة وجدت بطئ شديد في عمل اللعبة اثناء تفعيل هذا الخيار فيفضل عدم تفعيلة و استكمال الفحص بالوضع الطبيعي. من الصورة السابقة تستطيع رؤية التفاصيل الدقيقة التي يعرضها لنا البروفايلر اثناء فعيل هذا الخيار Current يعرض لك جميع التفاصيل في هذا الفريم اما Frame تعرض لك الاطار الحالي الذي انتقلت الية من بين جميع الاطارات التي تم تسجيلها اثناء اللعب.

٢) Record: هذا الخيار يقوم بتسجيل الاطارات اثناء تشغيل اللعبة. الزر Clear يمسح المخطط البياني لكي تبدأ من جديد. الازرار في الطرف الايمن تساعدك على الانتقال بين كل فريم، ايضاً الخيار

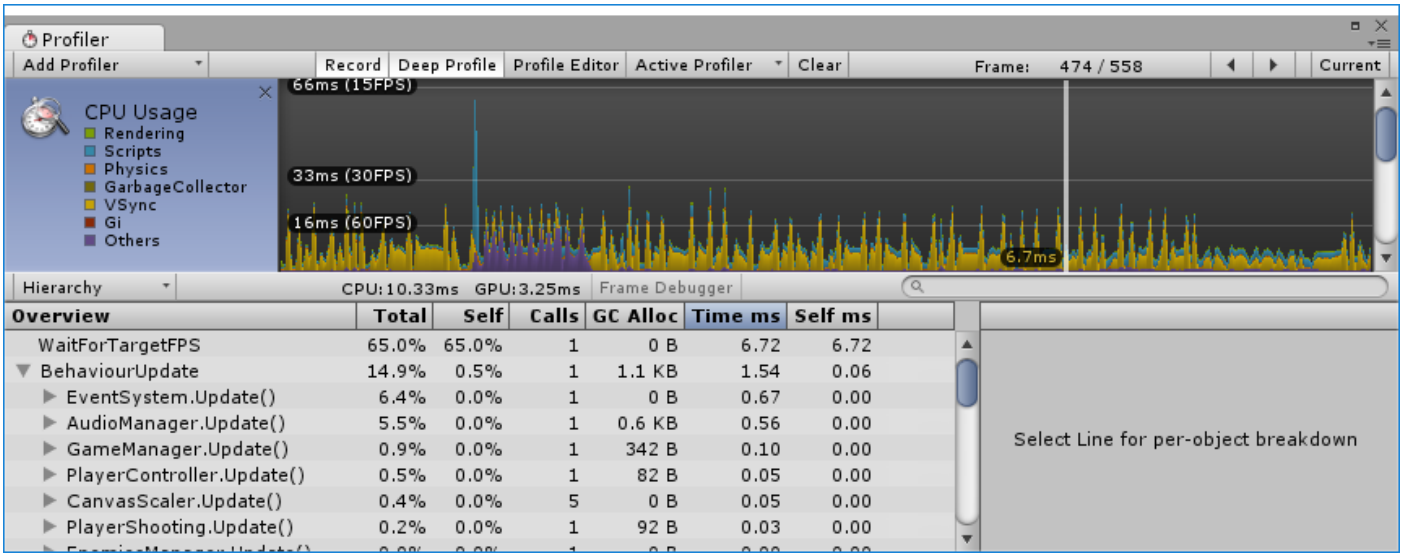
- القسم الثاني Add Profiler: يتيح لك اضافة بروفايلرات لعدة وظائف معينة مثل رؤية ذاكرة التخزين و معلومات الـ CPU و GPU.
- القسم المحدد بالأخضر: يعرض لك المعلومات المفصلة مع مزيد من الخيارات وعرض زمن الاستهلاك و الوقت المنقضِ و العديد من الاشياء.
- الخطط البياني عرض لك المعلومات على شكل رسم بياني بحيث ان كل خط يعرض شيء معين و بلون معين يمثل الألوان الموجود في الجانب الايسر، تجد ايضاً الخط الابيض وهو عمود الفرمات و تستطيع تحريك بشكل افقي للانتقال الى اي جهة و عرض المعلومات التي يحتويها هذا الفريم فمن الشريط العلوي يظهر رقم الفريم الذي وقفت عنده ويمكنك الانتقال فريم فريم من خلاله.

- القسم المحدد بالأزرق يعرض البروفايلرات التي اضفتها بحيث ان كل واحد يمثل شيء معين فمثلا بروفايلر CPU يعرض عملية الريندارينج و الخواص الفيزيائية و ايضاً الملفات البرمجية فمنه يمكنك الغاء تفعيل جميع الخيارات و ابقاء الملفات البرمجية Scripts و سيعرض لك جميع الدوال التي تم تنفيذها في المخطط و من القائمة السفلية يعرض لك المزيد من التفاصيل عن حالة الدوال و ايها التي تستهلك وقت و معدل اطارات اكثر.

هذه كانت واجهه البروفايلر فقط فهنا سنقوم بالتعمق اكثر في كل قسم بحيث تأخذ معلومات اكثر عن عملة في البرنامج.

اقسام الـ Profiler:

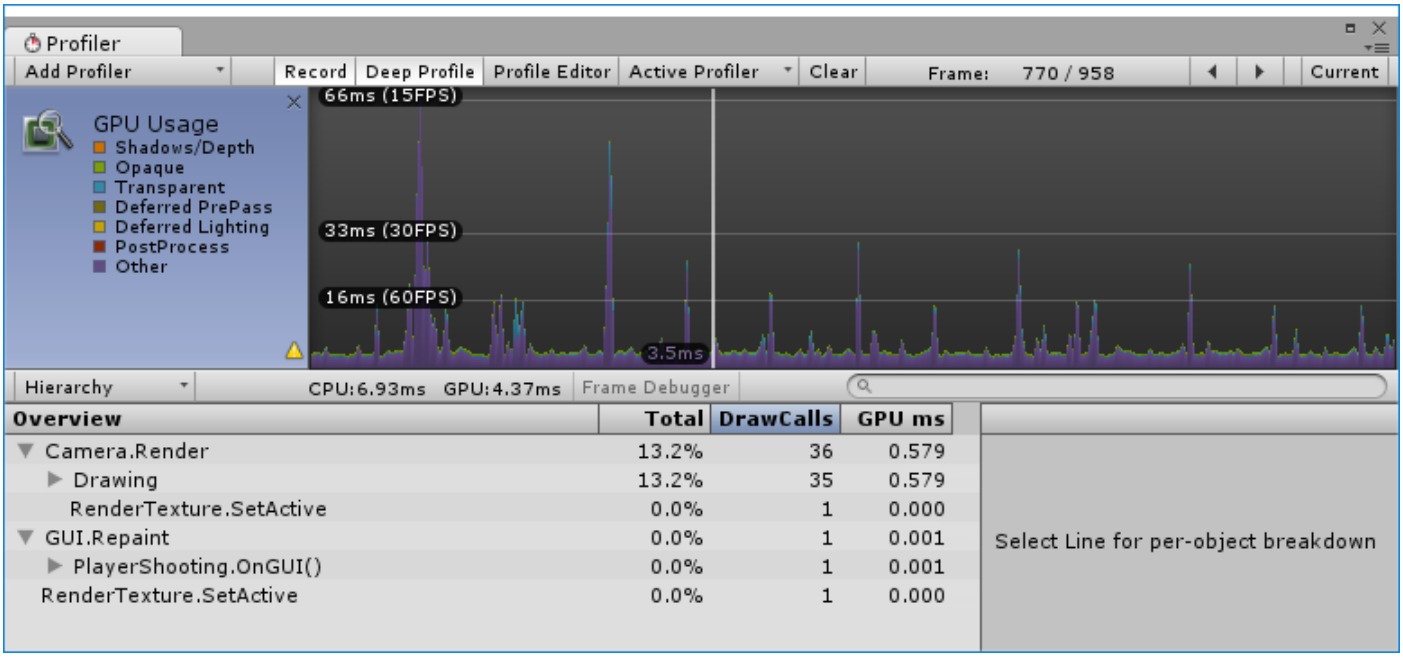
CPU Usage:



الـ CPU يعرض لنا مدى استهلاك المعالج للموارد في اللعبة (عملية الريندار، الملفات البرمجية، الخواص الفيزيائية، الـ GC، ايضاً VSync و اخيراً Other مثل البارتكلز و الاصوات و الاشياء الجانبية) فمن القائمة السفلية ترى البيانات في الهايروكي.

Hierarchy mode: من خلاله ترى جميع البيانات في التسلسل الهرمي، ومنه تجد طرق لعرض البيانات بشكل مختلف مثلا total يعرض المعدل الكلي لاستهلاك جميع الدوال وهو على شكل نسب مئوية، Self يعرض زمن تنفيذ الدالة لوحدها فقط، Calls ويعرض عدد المرات التي استدعيت فيها الدالة وتجد الدالة CanvasScaler استدعيت 5 مرات اثناء عمل اللعبة وهو يدل على الكائنات التي تحتوي عليه، ايضاً الاعداء يحملون CanvasScaler يعرض نسبة الدم وهذا يفسر سبب عرضة بهذه النسبة، GC Allco ووظيفة هذا الخيار هو عرض النسبة التي قامت بها الذاكرة لتخزين الكائنات في الفريم الحالي و لا يفضل ان يرتفع معدلة بشكل كبير.

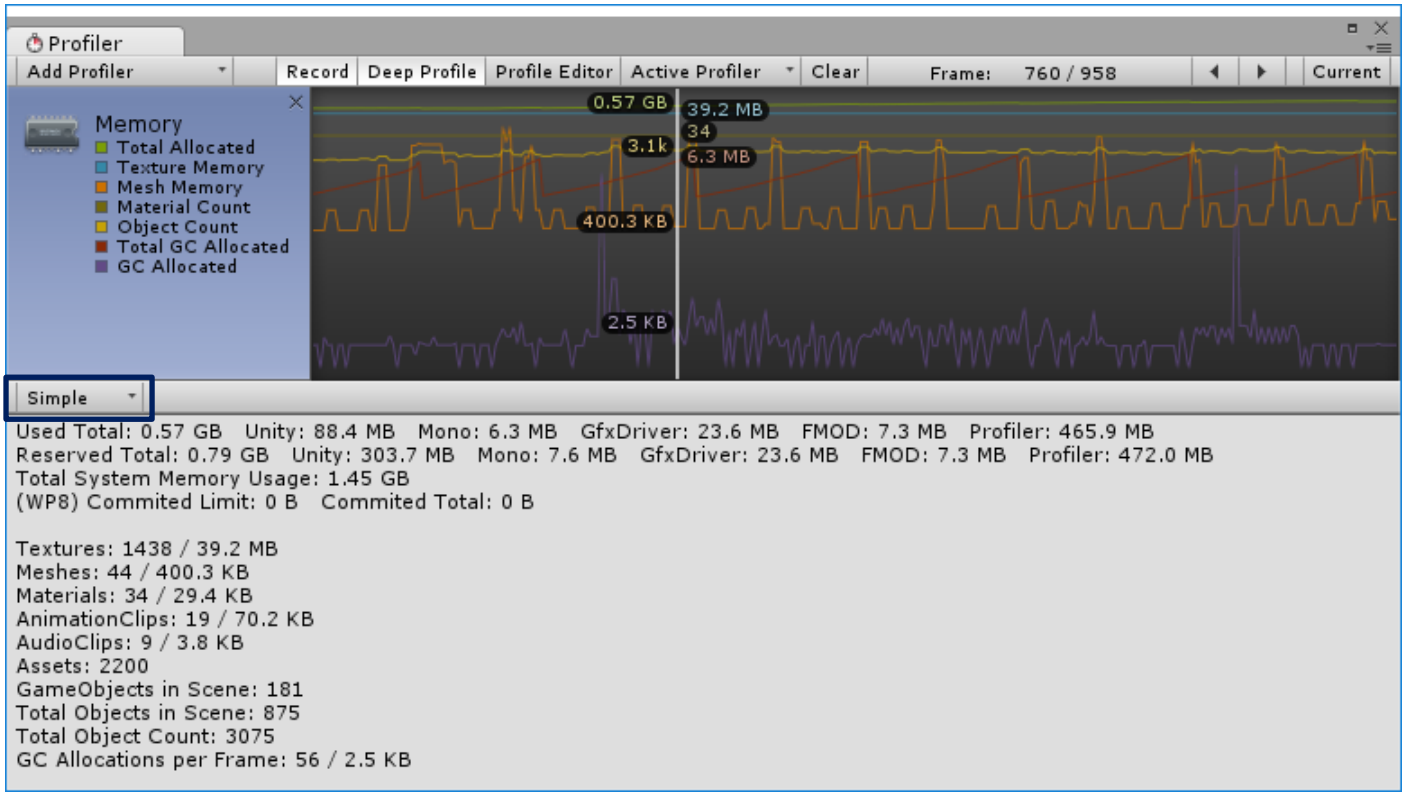
GPU Usage:



وهذا القسم غالباً لا يكون مضاف لهذا قم بإضافته من Add Profiler ومهمته عرض المواد و الدوال التي تستهلك معدل اكبر من الوقت وسحبها لقوة معالجات الرسوميات (كروت الجرافيك) فكما تلاحظ انه يعرض لك بعض المعلومات عن الريندار مثل الظلال و الـ Shaders و المؤثرات المرئية وهو لا يختلف كثير عن بروفايلر CPU و يمكنك رؤية المعلومات في الهايروكي واين يضيع الوقت و اشياء من هذا القبيل.

على الماك الخيار بروفايلر متاح فقط على OSX 10.9 Mavericks و الاصدارات الاحدث.

:Memory



هذا القسم يعرض لك معلومات التخزين و هناك نوعان لعرض معلومات التخزين وهما:

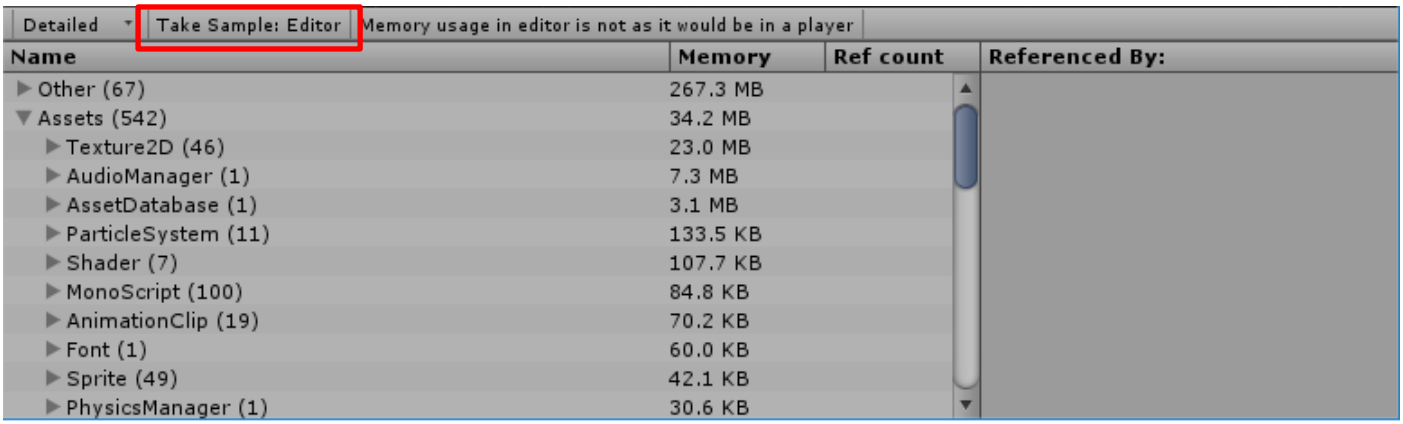
النوع الاول Simple: الوضع البسيط يبين لك المعلومات بشكل بسيط، فكما تلاحظ انه ينقسم الى قسمين (قسم التخزين، احصائيات المواد المخزنة).

القسم الاول: يعرض مدى تخزين جميع المواد و الكائنات في ذاكرة اليونتي و تلقائياً يقوم بوضع مساحة كبيرة لعملية التخزين لكي يتم تفريغها لاحقاً ولكي لا يضطر الى انشاء مساحة جديدة و من ثم تفريغها و هذه العملية مكلفة، فيه تجد التفرعات التي تعرض لنا مدى تخزين المواد وهي:

- Unity و Mono: يعرضان مقدار الاشياء المخزنة في ذاكرة اليونتي و المساحات التي تم حجزها في الذاكرة لعمليات التخزين القادم و لكي لا يضطر اليونتي الى وضع مساحة جديدة ثم تفريغها.
- GfxDriver: هي الذاكرة المتاحة لمعالجة الرسوميات مثل Textures, Render targets, Shaders and Mesh data.
- FMOD: وهو محرك المؤثرات الصوتية.
- Profiler: وهي الذاكرة المخزنة لعمليات التي يقوم بها البروفايلر.

القسم الثاني: يعرض احصائيات الذاكر للمواد الاكثر استخداماً في اليونتي و تستطيع ان ترى عدد الكائنات و الموجودة في الـ Assets و الموجود في المشهد، الاصوات المستخدمة، ذاكرة الفيديو، مقاطع الانميشن، التكستشرات و الـ Meshes و كلها احصائيات تقوم بها الذاكرة لأعلامك بالمواد التي ستقوم بتخزينها.

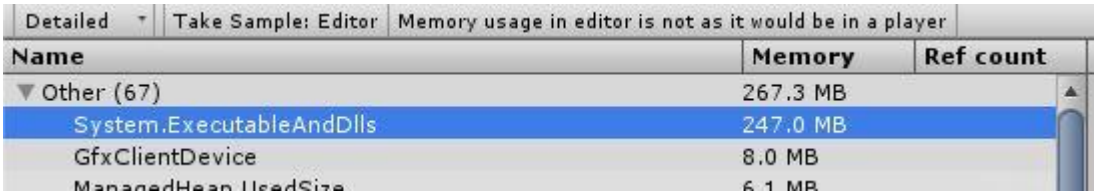
النوع الثاني Detailed: الوضع المفصل و يعرض لك جميع الاشياء بشكل مفصل جداً وهو لكن تستطيع رؤيته حال الضغط على الفريم فهو لا يعمل في كل فريم لأنه يطلب الوقت للحصول على البيانات.



الوضع مختلف قليلاً هنا حال الضغط على Take Sample Editor ستلاحظ ظهور البيانات على شكل تسلسل هرمي و يمكن رؤية معدل تخزين المواد و عددها بين قوسين وتتفرع المواد من عدة ملفات تمثلها و تعرض جميعها بشكل فردي، ايضاً تجد الاشياء الغير محفوظة في Not Saved.

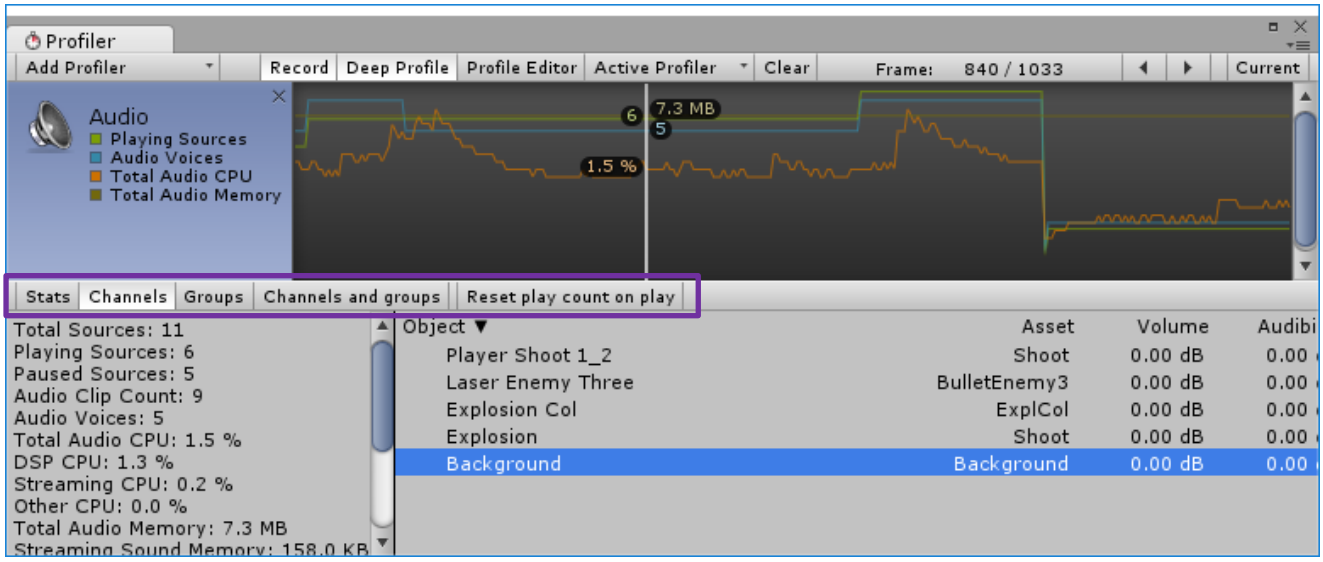
حال تحديدك على كائن معين ومن ثم الضغط عليك سينقلك الى موقع هذا الكائن، مثلاً القائمة Texture2D و تجد فيها العديد من الصور، عند الضغط على اي صورة سيتم نقلك الى موقعها و كذلك الحال مع باقي المواد.

الذاكرة System.ExecutableAndDlls المبالغ فيها هي للقراءة فقط ويمكنك تجاهلها ولكن حل رغبتك في رؤية التخزين في شكلة الصحيح يفضل استخدام العملية عند بناء اللعبة اي خارج المحرك.



Audio:

ضمن البروفايلرات تجد البروفايلر Audio فمن اسمة هو يختص بالاصوات و لكن هذه المرة بشكل دقيق فمثلاً الاصوات التي تعمل في اللعبة و استخدامها في المعالج وامور اخرى.



تجد هنا قائمة تتيح لك رؤية الاصوات و حالاتها فالخيار State يعرض حالة الاصوات مثلاً:

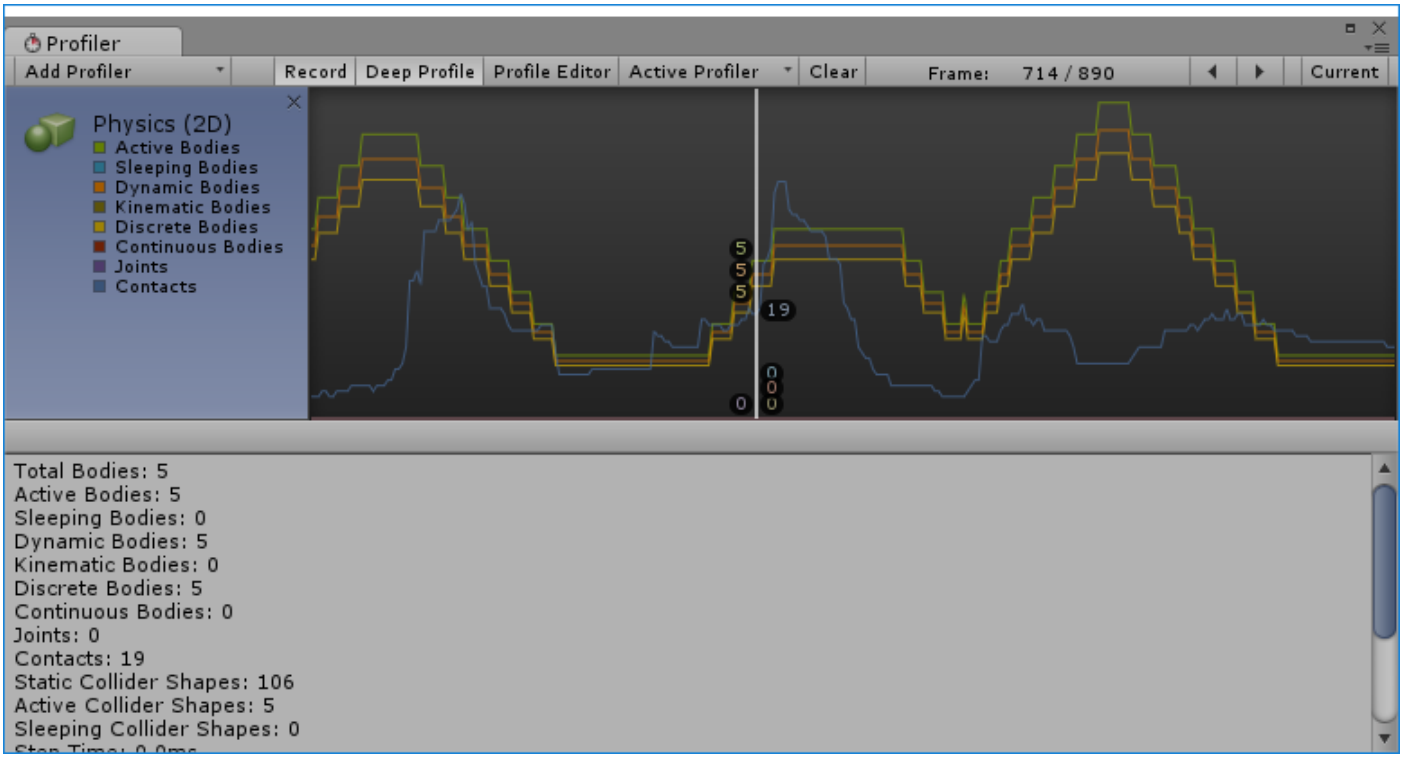
- Playing: الاصوات التي تعمل في المشهد.
- Paused: الاصوات الموقوفة مؤقتاً في المشهد.
- Audio Voice: عدد الاصوات التي تعمل حالياً
- Audio Memory: الذاكرة السمعية المخزنة للأصوات.

بجانب هذا الخيار يمكنك تفعيل خيار اضافي مثل Channels، Groups بحيث ان Channels يعرض الاصوات التي على شكل Channels (اي ليست)واحدChannel التي عملت اثناء اللعب و عند التحديد على صوت معين انتظر لثواني و سينقلك الى موقعة في المشهد ادا كان موضوع على كائن معين، بينما الخيار Groups يعرض لك الاصوات التي على شكل مجموعات.

هناك عدة خيارات يمكن تجربتها وهي لا تختلف كثير و لكن تبقى مراجعة حالة الاصوات ضرورية لمعرفة الاصوات التي بالفعل عملت و التي لم تعمل و لماذا !؟.

:Physics2D

صراحةً هناك بروفايلران لمحرك الفيزياء الاول للـ 3D و الثاني للـ 2D ففي كلا الحالتين تعرض المعلومات نفسها و لكن حسب استخدام محرك الفيزياء.



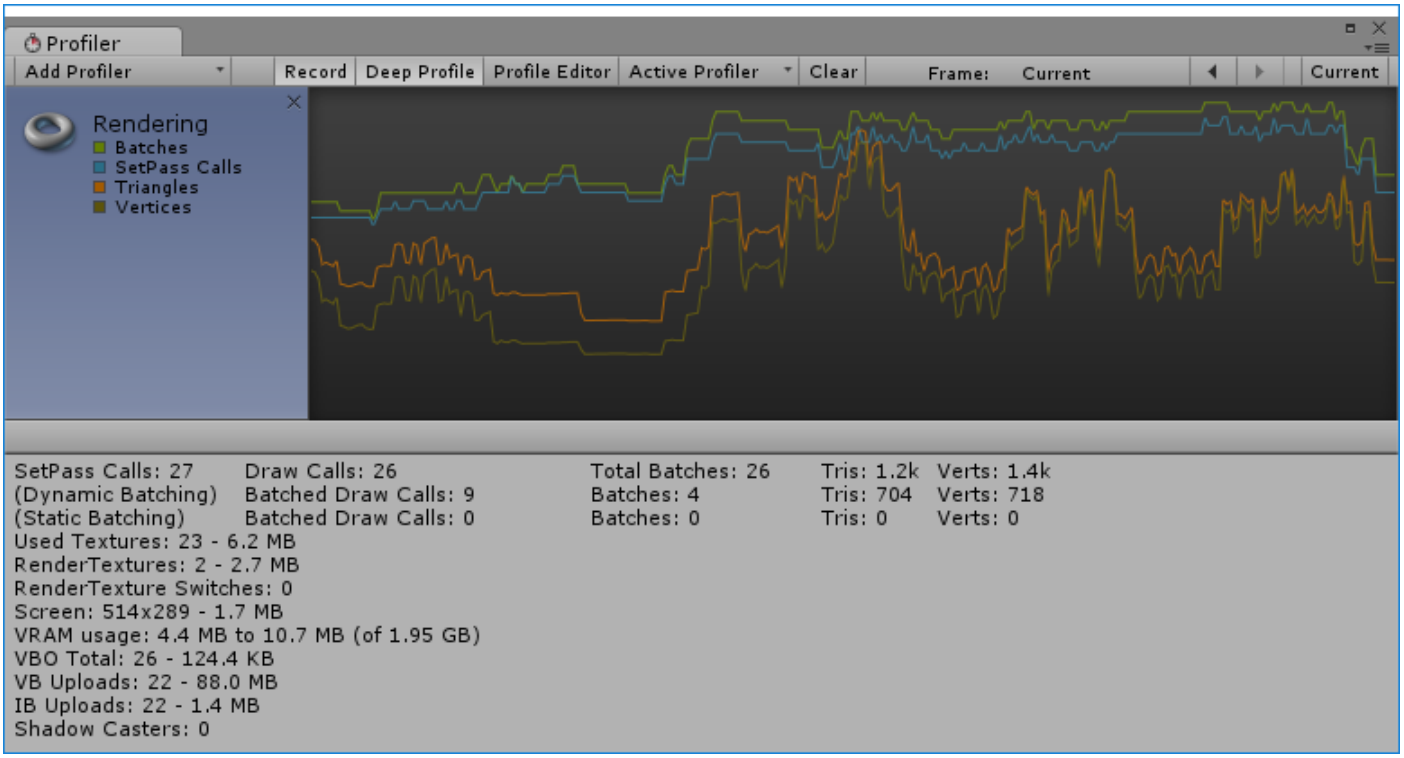
لاحظ المخطط البياني وهو يعرض الخطوات الفيزيائية و تدل على هذا شكل الخطوط وكل خط ملون يمثل مكون معين، اما القائمة السفلية فهي تعطيك تفاصيل اكثر عن معلومات محركة الفيزياء في كل فريم ويمكنك مراجعتها على هذا الشكل:

- الاجسام الصلبة الفيزيائية التي تتحرك (الغير نائمة).
- الاجسام النائمة التي لا تحتاج لتحديث نشط بواسطة محرك الفيزياء.
- عدد التصادمات لجميع الكائنات الفيزيائية.
- الكولايدرات التي توضع على الكائنات دون الـ Rigidbody.

- التصادمات الديناميكية و التي توضع على الاجسام الجامدة (الغير فيزيائية).

مفهوم هذا البروفايلر سهل وتستطيع الاعتماد على هذه النقاط لحساب الحالة الفيزيائية للكائنات او بالطريقة التي تراها مناسبة.

:Rendering

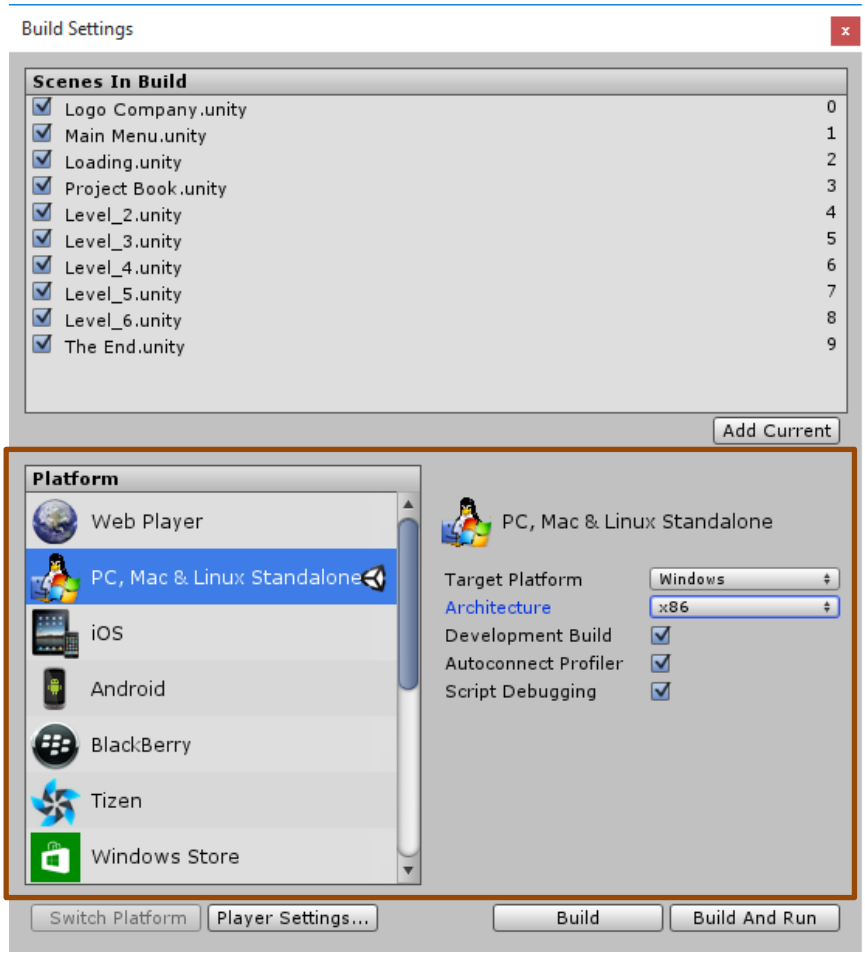


هذا القسم يعرض الاشياء التي يتم عمل Rendering لها ويمكنك رؤيه الـ Batches وهي متعلقة بالألعاب ثلاثة الابعاد اكثر من الثنائية، SetPass Calls وتظهر في Draw Calls ويمكنك ملاحظتها، ايضاً تقسيم المربعات لعمليات الريندار فاليونتي يقوم بتثليث كل مربع لتسهيل العملية.

تجداً ايضاً الـ VRam وهي الكميات التكستشرات التي يتم معالجتها و المزيد من المعلومات يمكنك مراجعتها.

ربط الـ Profiler خارجياً:

عند ربط الـ Profiler يجب ان نقوم ببناء هذه للعبة و لكن على شكل اختبار لها وليست النسخة النهائية لهذا فمن القائمة Build Setting ستظهر نافذة بناء اللعبة، هنا دعنا نناقش واجهتها:



قائمة Palatfrom وهي تحدد المنصة التي ستصدر اللعبة لها وتجد العديد من المنصات ويجب ان تكون المنصة التي نعملها عليها هي PC, Mac & Linux و تستطيع اختيار احد هذه المنصات من الخيار Target Platform فقم باختيار المنصة التي تعمل عليها، الخيار Architecture او نسخة المنصة و توفر نوعان ويفضل وضعها على الحالة الافتراضية.

في حالة اردت الانتقال الى منصة اخرى يمكنك بكل بساطة التحديد على المنصة المرادة ثم Switch Platform.

اخيراً تجد خيارات تصدير اللعبة للفحص فقط فعند تفعيل خيار Development Build سيتم اتاحة استخدام الخيارات الاخرى، Autoconnect Profiler يسمح لك بعمل Profiler للنسخة الخارجية، اما الخيار Script Debugging يتيح لك فحص الملفات البرمجية من النسخة المبنية. اذن هذه خيارات الفحص عند تصدير اللعبة يجب ان تكون كما هو موضح في الشكل السابق مع اختيار نوع المنصة.

MonoDevelop و عملية الـ Debugging :-

وصلنا للجانب البرمجي في اللعبة، ان عملية البرمجة ضرورية جداً ولا تكاد اي لعبة تخلوا من الاوامر البرمجية، هذا الشيء منطقي و العملية البرمجية هي التي تسير عملية اللعب و ترتيب المشهد بحد ذاته.

عند الحديث عن برمجة المدفع فأن المدفع سيطلب الكثير من الامور البرمجية (الحركة، الدوران، الصحة، اطلاق النار...الخ) و علينا ترتيب الاشياء بحسب اسمها في البرمجة فمثلاً دالة اطلاق النار باسم PlayerShoot او شيء من هذا القبيل، و هذا سيسهل عليك البحث عن وظائف المدفع، هذا الشيء لا يقتصر على المدفع فقط بل عل اي كائن في اللعبة يحتاج الى مثل هذه الوظائف لعملة.

لمحة عن محرر النصوص MonoDevelop:

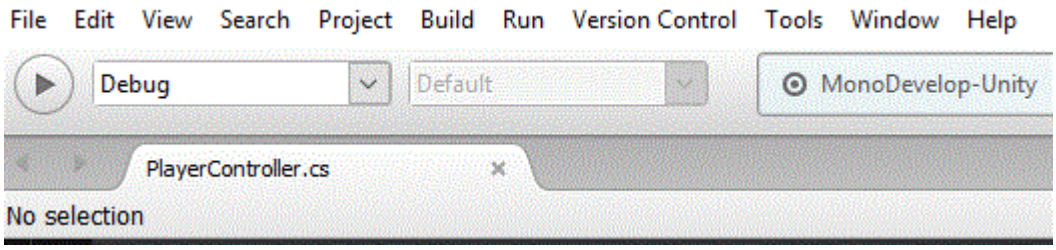
ان MonoDevelop بيئة محرر النصوص المرفق مع Unity وهو المحرر الافتراضي للبرنامج، يتيح لك كتابة النصوص و عمل فحص للملفات و الأكواد البرمجية و يساعدك على التقليل من استهلاك الموارد في المحرك، يوجب به العديد من الخصائص و التي تتيح لك التعامل مع مشاريعك بدقة وحرص اكثر، ايضاً يتيح لك تغيير خلفية الشاشة لكتابة النصوص و اشياء اخرى.

اعدادات الـ MonoDevelop:

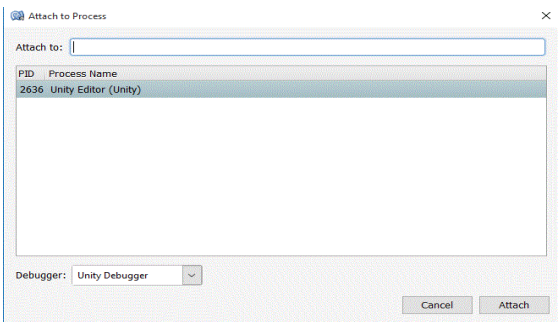
يتم اضافة محرر النصوص MonoDevelop تلقائياً مع محرك Unity و يمكنك تغيير المحرر الى محرر آخر و غالباً يتم استخدام الـ Visual Studio، يمكنك تغيير المحرر من Edit> Preferences> External Tools> External Script Editor من خلاله يمكنك استعراض بيئة محرر النصوص الذي تريده و استخدامه كمحرر افتراضي في البرنامج.

عملية الـ Debugging:

ان هذه العملية تساعدك على فحص الملف البرمجي بشكل كامل في حالة نسيت اغلاق الكود او أي خطأ برمجي يتم فحصة قبل تشغيل اللعبة و هذه العملية تساعدك على فحص الملف البرمجي قبل رؤية عملة في اللعبة و ظهور المشاكل، لعمل Debugging للملف البرمجية نقوم بأنشاء ملف برمجي C# ولاحظ الان هذه الصورة:



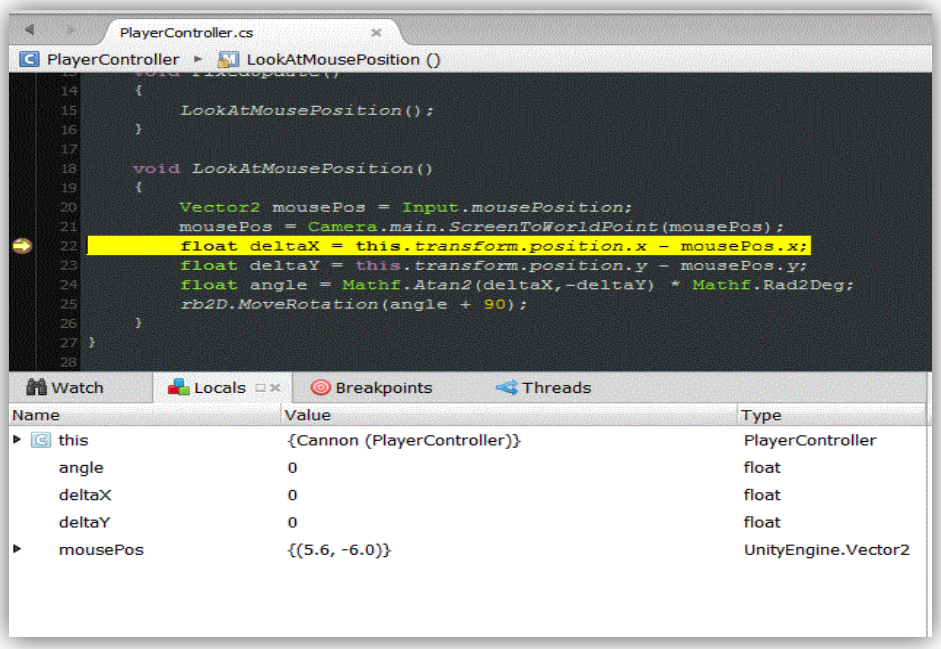
تلاحظ في هذا المحرر العديد من القوائم و التي سيكون شرحها صعب بهذا الشكل لكن سأعطيك لمحة عن بعض منها و سيتبقى لك دراسة الباقي، حسناً لعمل Debugging للملف البرمجي نقوم بالذهاب الى القائمة Run و الضغط على الخيار Start Debugging او باختصار قم بالضغط على F5 هنا ستلاحظ ان محرر النصوص يقوم بعمل فحصل للملف البرمجي او قد ربما لا تلاحظ عملية الفحص لأنها لا توجد اخطاء برمجية في الملف، لهذا عند وجود خطأ برمجي سيقوم بتحديده البرنامج بخط احمر.



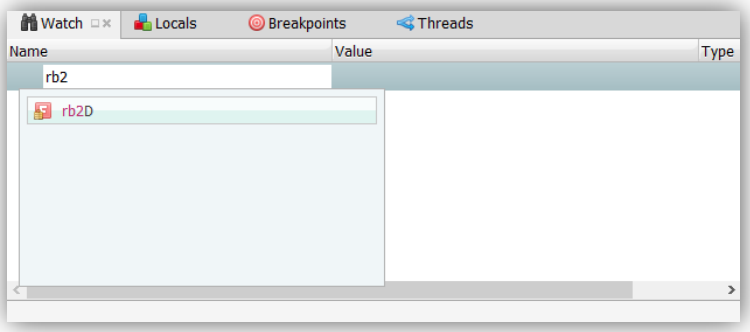
لكن في البداية يتم ربط الـ MonoDevelop في نسخة اليونتي التي تريدها لهذا اذهب الى Run ثم الخيار Attach To Process هذا الخيار سيساعدك على ربط الملف الحالي باللعبة عند تشغيلها لكن في البداية سيطلب منك تحديد النسخة التي تريدها، و بكل بساطة قم بتحديد النسخة التي تعمل عليها ثم Attach و ستبدأ عملية الـ Debugging تلقائياً.

وضع الـ Breakpoint:

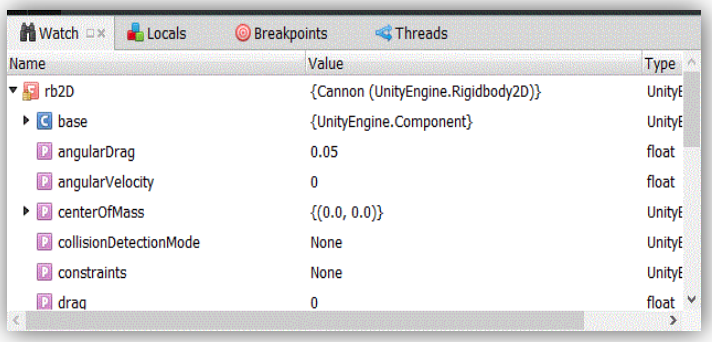
عملية وضع الـ Breakpoint تساعدك على معرفة ماذا يحتوي هذا الكود وماهي الوظائف الذي يعملها لهذا الكائن، وضع الـ Breakpoint سيؤدي الى توقف عملية الفحص عند هذا الكود البرمجي و سيتم استخراج جميع قيمة ويمكنك مشاهدتها من القائمة Locals، طريقة وضع الـBreakpoint سهله جداً فكل ما عليك فعلة هو الضغط على الجانب الايسر للكود المراد و سيتم وضع الـ Breakpoint.



لاحظ جميع القيم التي يحتويها هذا الكود فمن القائمة Locals يمكنك رؤية سير عملية الفحص، لكن و قبل كل هذا تأكد من انك قمت بتشغيل المشروع اثناء عملية الفحص لكي يتم التحقق من جميع الأكواد في المشروع.



القائمة Watch تتيح لك معرفة قيم متغير معين او فئة او دالة بشكل سريع و مباشر، كل ما عليك فعلة هو فتح القائمة Watch كما تلاحظ في الصورة و سيطلب منك الضغط للمشاهدة و كتابة اسم الفئة او الدالة او المتغير لعرضها في القائمة، لاحظ الصور في الاسفل:



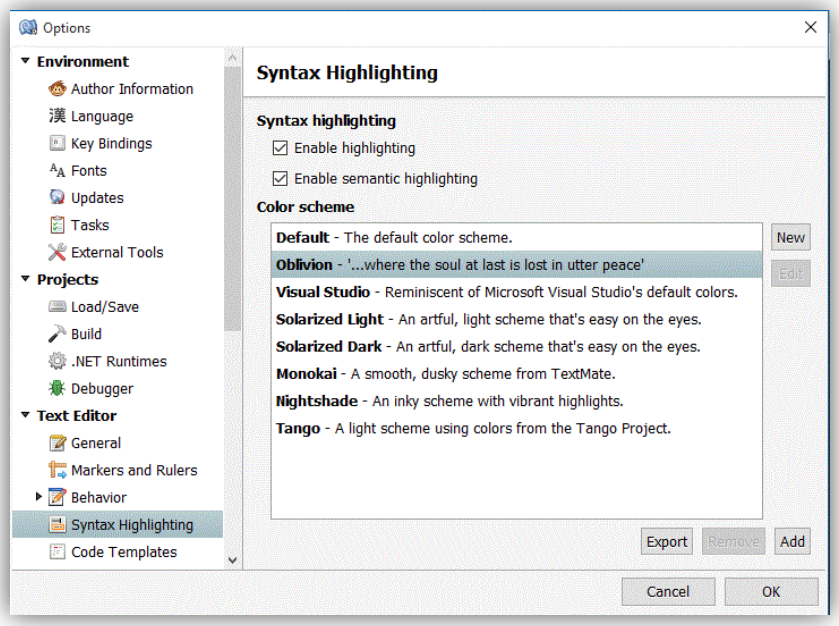
لاحظ مدى التفصيل الذي تتيحه لك عملية الفحص، من خلال فحص المتغير rb2D والذي يمثل الفئة Rigidbody2D يعرض لك المتغيرات و الدوال التي تم استخدامها و ماهي قيمها الحالية و الفعلية، شخصياً ارى ان استخدام عملية فحصل الملفات البرمجية تتم في وقت تكون الملفات في حالة تشعب و لهذا تتم عملية الفحص للتأكد من ان جميع الملفات تخلوا من المشاكل او الأكواد التي ليس لها مكان في الملف (اي موضوعة دون عمل).

في حالة اردت الخروج و اعادة التنفيذ فهناك طريقتان، اما ان تقوم بالخروج من العملية بشكل كامل، او ان تقوم بإيقافها و استئنافها في وقت لاحق.

في الصورتين في الاعلى تلاحظ القائمة التي تتيح لك تشغيل و ايقاف عملية الفحص، في الصورة اليسرى ستجد زر ايقاف و استكمال عملية الفحص، يمكنك بكل بساطة الضغط علي للتأكد من انه تم ايقاف عملية الفحص و اعادة تشغيلها.

بالنسبة للقائمة اليمنى ستجد الثلاث ازرار و هي خاصة بالخروج و فحص الدوال او المتغيرات التي تريدها، بالنسبة للزر الاول و الذي يسمى Step Over ووظيفته القفز من السطر الحالي الي السطر الذي بعدة و ستتم عملية فحص هذا السطر، اما الزر الثاني و يسمى Step Into و هو الذي يقوم بجلب جميع المعلومات لهذا الكود بالتفصيل و هو مفيد جداً في حالة اردت معرفة عمل كود معين و بدقة، بالنسبة للزر الاخير ويسمى Step Out و هذا الزر يساعدك على الخروج من هذا السطر بكل بساطة لتبدأ الفحص من البداية، يمكنك ايضاً ان تجد هذه القائمة من Run و ستجدها في الاسفل و يمكنك التحكم بعملية الفحص عبرها.

ان عملية فحص الأكواد و تصحيحها يختلف من محرر نصوص الى آخر فكل محرر له قوائمه و خصائصه المختلفة عن باقي المحررات، في اغلب الاحيان و عند البدا بعملية البرمجة يتم استخدام الـ Visual Studio لأنه يعتبر من اقوى الـ IDE نظراً للإضافات و السرعة في عمليات الفحص و التصحيح و الربط و غيرها، يضاً يتيح لك ربط محرك Unity فيه، فنسبياً ارى ان محرر MonoDevelop جميل و سريع و دائماً تقوم الشركة بتحديثه، شيء اخير بالنسبة لهذه القوائم قد تجدها مختلفة في النسخ السابقة ولأنه تم اضافتها في النسخة 4.6 الى النسخ الحالية من المحرر.



اعدادات الـ Syntax Highlighting:

بالحديث عن خلفية المحرر تجدها بيضاء و ربما العديد منا يواجه مشكلة الخلفية البيضاء في المحرر و دائماً تتعبك في تركيزك على الملف البرمجي، في محرر MonoDevelop تستطيع تغير الخلفية و باللون الذي يتناسب معك و يوجد

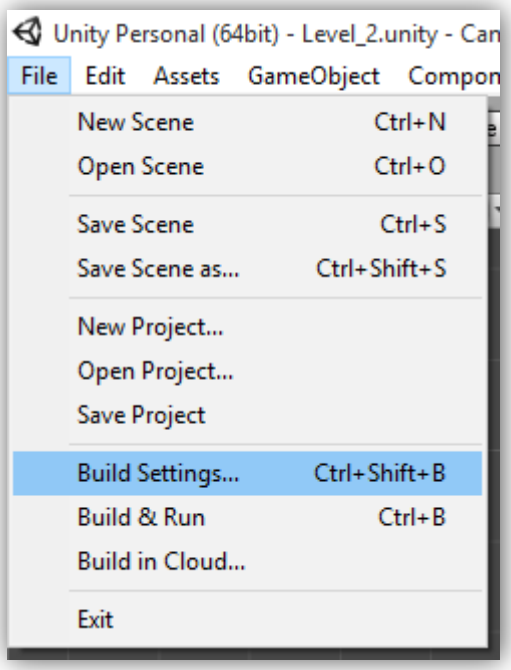
افتراضات يضعها المحرر لاستخدامها، لتغيير لون خلفية المحرر اذهب الى Tools> Options ستظهر لك نافذة فيها العديد من القوائم، لاحظ الصورة:

هنا تجد العديد من الخيارات يمكنك استكشافها لاحقاً، لكن الان دعنا نذهب لتغيير لون الخلفية، اذهب الى القائمة Text Editor و التي تجد منها العديد من التفرعات، قم بالضغط على الخيار Syntax Highlighting من خلاله يمكنك تحديد لون او إضاءة الخلفية، ستجد العديد من الخيارات يمكنك استكشافها، بالنسبة للخلفية التي اراها مناسبة فهيا الخلفية الثاني من الاعلى كما هي محددة في الصورة.

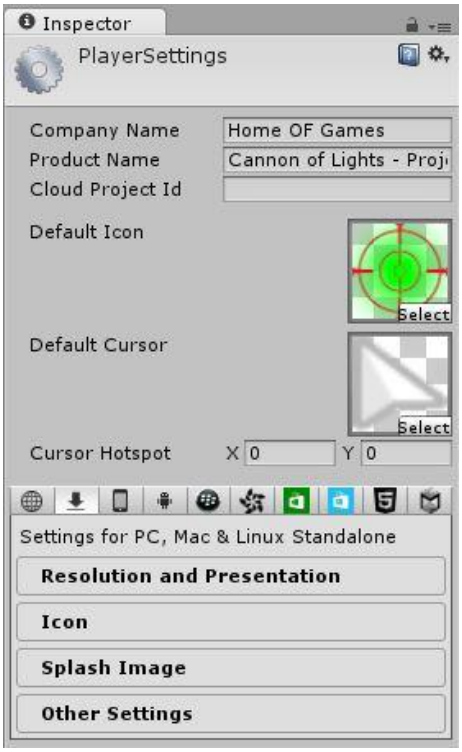
خيار Enable Highlighting يتيح لك تفعيل الاضاءة للفئات و المتغيرات و الدوال و التي تساعدك على التمييز بينهم، يمنك الغاء تفعيلها لتتأكد بنفسك، بالنسبة للخيار الثاني Enable semantic Highlighting هذا الخيار له لمسة جميلة انه يقوم بعمل ميلان للكود الذي تم كتابته بشكل صحيح او الذي تم أقفاله بفاصلة منقوطة ان صحت العبارة، عند تفعيلة ستلاحظ ان جميع الأكواد التي تم اقفالها مائها و التي لم يتم اقفالها او تواجه مشاكل تجدها غير مائلة، هذا الغير ربما قم يعلمك بهذا الشيء في حال غفولك عن وفي النهاية لك امر تفعيلة و ايقافه.

بناء اللعبة:-

وصلنا للفقرة الاخيرة زهي عمل Build للعبة اي بناء اللعبة و تصديرها على المنصة التي تريدها، سابقاً تعرفنا على القائمة Build Setting ايضاً هناك اوامر الحفظ السريع و تجدها في اسفل الخيار:



Build & Run يقوم بناء اللعبة مع تشغيلها بشكل مباشر، و لكن اعدادات البناء هي من القائمة المعتادة Build Setting ومنها اضغط على الزر Player Setting فهذا الخيار لا يعني اللاعب بل اعدادات مشغل اللعبة فمن الانسباكتور ستظهر لك هذه القائمة:



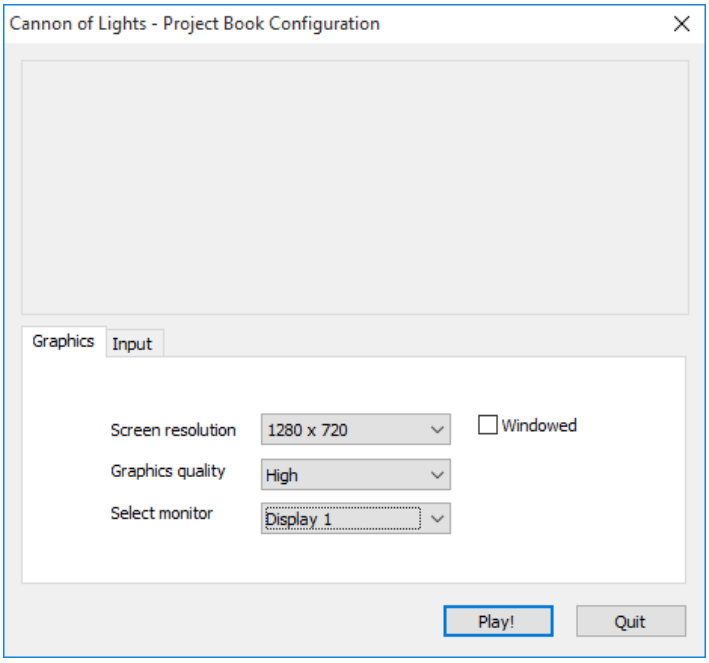
لاحظ ان القائمة مقسمة الى فرعين، بحيث ان الفرع الاول يطلب منك تحديد اسم اللعبة بالاضافة الى اسم الشركة المصنعة، وضع شعار اللعبة الرمزي ووضع صورة لمؤشر ففي حال كان يوجد لديك صورة للمؤشر قم بجعل نوعها هو Cursor لكي يتعرف عليها اليونتي و تستطيع ان ترى المؤشر حال بناء اللعبة فقط، الخيار Cursor Hotspot يعطيك احداثيات وضع المؤشر ففي حالة لم يكون موقعة صحيح فقم بجعل القيم 1 او بحسب رؤيتك له.

الفرع الثاني هي الاوامر الضرورية مثل التصدير الى الهواتف او الويب وغيرها اما حالياً فأجعله على المنصة الحالية و تأكد من ان اللعبة ليست مبنية للتجربة، تلاحظ انه مقسم الى 4 قوائم فرعية وهي:

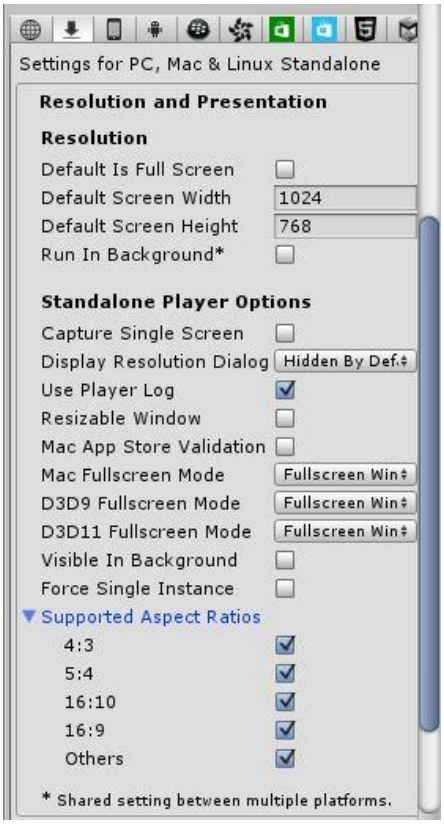
- **Resolution and Presentation**: اعدادات دقة الشاشة وظهور قائمة الاعدادات ايضاً حجم الشاشة.
- **Icon**: وضع ايقونة للعبة ويتم تحجيمها لتتناسب مع حجم الشاشة.
- **Splash Image**: صورة تعرض شكل للعبة او شيء من هذا القبيل.
- **Other Settings**: اعدادات وخيارات عشوائية.

Resolution and Presentation:

بالنسبة لهذا الخيار فوظيفته هي التعامل مع شاشة العرض و يتيح لك ايضاً خيار تفعيل الشاشة المنبثقة (شاشة الاعدادات):



حسناً هناك عدة خيارات توفرها لك هذه القائمة، اولاً دعنا نلقي نظره عليها:



لاحظ ان القائمة مقسمة الى قسمين، القسم Resolution اعدادات الشاشة، وقسم Standalone Player Options اعدادات المنصة المستقلة، دعنا نبد.

القسم الاول Resolution:

- Default is Full Screen: يجب ان تبدأ اللعبة في وضع شاشة افتراضي؟
- Default Screen Width, Height: طول و عرض الشاشة بالبيكسل.
- Run in background: ادا خرجت من التطبيق هل تعمل اللعبة ام تتوقف؟

كما تلاحظ ان الاوامر بسيطة جداً و يمكنك وضع اعداداتك التي تريدها.

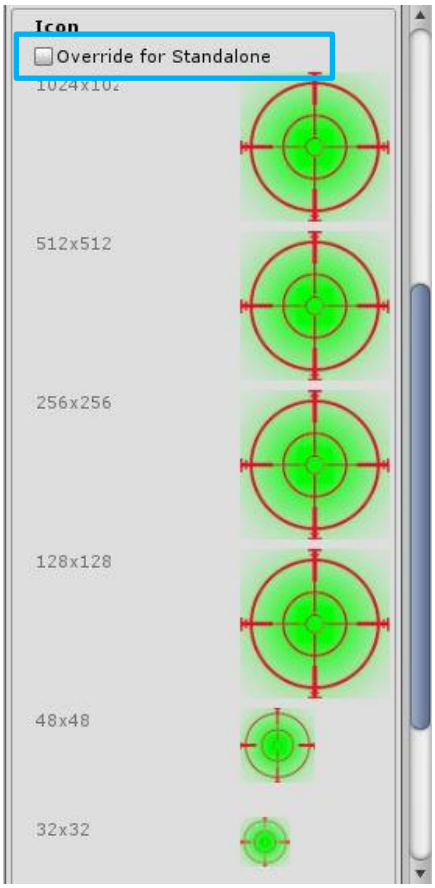
القسم الثاني Standalone Player Options:

- Capture Single Screen: وهذا الخيار للأجهزة ذات الشاشتين و حال تفعيلة لن يجعل الشاشة الثانية سوداء.
- Display Resolution Dialog: هذا الخيار يحدد هل سيتم عرض شاشة الاعدادات ام لا و انا صراحةً لا اجده احترافي و يفضل الغاء تفعيلة ولك الامر، وهناك خيار اخير يتيح لك اظهار الشاشة و اخفائها عند الضغط على مفاتح Alt.
- Use Player Log: هذا الخيار يرسل ملف بالتقارير حول اللعبة اثناء تشغيلها ويفضل الغائه، و للعلم انه لا يعمل في الماك.
- Resizable Window: سيمح للاعب بتغيير حجم الشاشة عن طريق الماوس.
- Mac App Store Validation: مهم لنشر اللعبة على متاجر الماك.
- Mac Fullscreen Mode: ملء الشاشة في اجهزة الماك.
- Supported Aspect Ratios: يحدد الطول والعرض المتاح من اللعبة وتجد فيه عدة خيارات.

وستجد ان هذه الخيارات مهمه لتحسين بناء اللعبة ويجب المرور عليها على الاقل.

Icon:

هذه القائمة تساعدك على وضع شعار للعبة يظهر في سطح مكتب جهازك وحال فتح القائمة ستظهر بهذا الشكل:



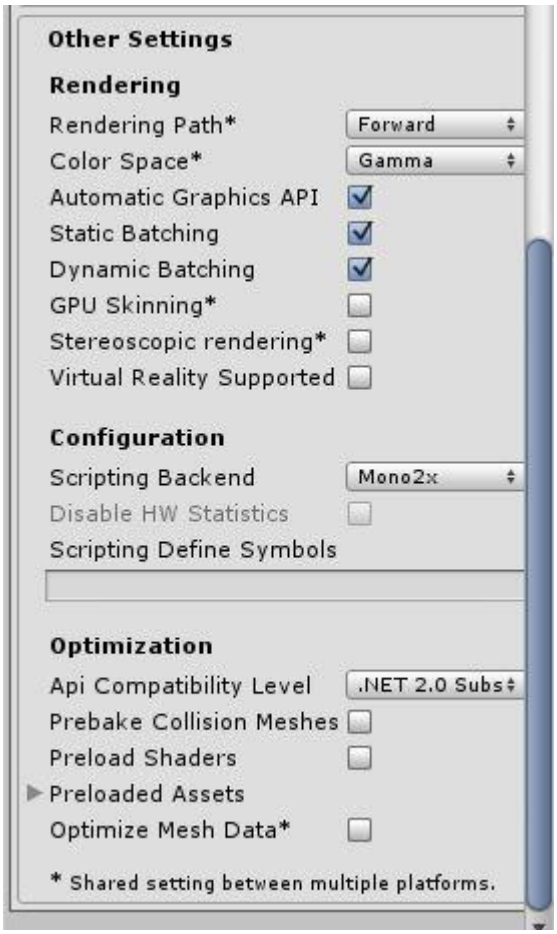
تلاحظ موجود الخيار Override for Standalone ويساعدك على ضافة اكثر من الايقونة و على اكثر من حجم و يمكنك بكل بساطة سحب الايقونة الى الحجم الذي تريده كما هو موضح.

Splash Image:

هذا القسم يتيح لك وضع صورة تظهر في قائمة الاعدادات عند تشغيل اللعبة و حال الغاء قائمة الاعدادات فلا حاجة لاستخدامه.

قد تواجه مشكلة ظهور الصورة بشكل كبير عند بناء اللعبة و هذا الامر يحل بجعل حجم الصورة هو 163 x 432 بيكسل ويفضل استخدام الفوتوشوب لمثل هاكذا عمليات.

:Other Settings



تجد ان هذا القائمة مقسمة الى 3 اقسام وهي: Rendering, Configuration, Optimization. و كل خيار يمثل اعدادات معينة.

القسم الاول Rendering:

- Rendering Path: يحدد نوع عملية الـ Rendering الخاصة باللعبة وتستطيع الاختيار فمثلاً Vertex Lit الاضعف و يفضل استخدامه للهواتف، Forward المتوسط و الذي يدعم الظلال بنسبة بسيطة، Deferred الخيار الاقوى ويستخدم في الالعاب الضخمة.
- ColorSpace: مساحة الألوان وهل هي Gamma او لا ويفضل استخدامه.

الخيارات الاخيرة غير مفيدة حالياً و لكن تستخدم لتتبع حركة العين و استخدام نظارات الواقع الافتراضي.

القسم الثاني Configuration:

وهذا القسيم يتيح لك استخدام رموز برمجية للاستخدامها لاحقاً وهو غير مهم.

القسم الثالث Optimization:

- API Compatibility Level: يحدد نوع مكتبة Network التي ستتضمنها اللعبة ويوفر خياران وهما: Net 2.0 استخدام اكبر مكتبة دون نت ولكن سيؤدي الى جعل حجم اللعبة اكبر، Net 2.0 Subset استخدام المكتبة الصغيرة مع جعل حجم اللعبة اصغر.
- Prebake Collision Meshes: يضيف بيانات التصادم للـ meshes اثناء بناء اللعبة.
- Optimize Mesh Data: وقوم هذا الخيار بإزاله اي بيانات غير مستخدمة مثل الظلال الالوان و الاشعة الفوق بنفسجية ويفضل تفعيلة.

اخيراً اترك لك امر اختيار الاعدادات الضرورية للعبة و عند بناء اللعبة تأكد من انها ليست للاختبار.

مراجع:

- مستندات اليونتي: http://docs.unity3d.com/Manual/index.html
- ورشة عمل لعبة Shoot 'em Up (للكتابة محمد طلحان):
https://drive.google.com/file/d/0BzRjTVZk4vwWMEFXZ0R2VU1rdVE/view?usp=sharing
- اسأله في اليونتي: http://answers.unity3d.com/

جزء كبير من اللعبة انهيته حسب خبرتي ففي الاخير اترك لكم هذه المراجع الغنية عن التعريف التي من خلالها تستطيع توسيع معرفتك اكثر بقراءتها.

مستندات اليونتي ضرورية وتجد فيها كل شيء بشكل مفصل و يفضل المرور عليها من وقت لأخر.

كتاب الاخ محمد طلحان كتاب مفيد جداً و اردت نشرة هنا لأهميته في جانب تصميم الالعاب و الحصول على الكثير من المعلومات، ايضاً هو باللغة العربية و اضن انه ضمن احد الكتب النادرة باللغة العربية.

المرجع الاخير سيساعدك بشكل كبير و ربما قد تجد جميع تساؤلاتك هناك فيفضل زيارته من وقت لأخر اما بطرح الأسالة او بالبحث عن اجوبة.

الخاتمة:-

في النهاية اتمنى ان تكون هذه الورشة قد اعطتك فكرة عن عالم تطوير الالعاب وماهي الصعوبات و المشاكل و طريقة حلها وتخطيها و بالأخص مبدأ ان كل شيء جامد و يجب بث الروح فيه، عند لعبك للعبة سترى انك تستطيع انهائها في دقائق و لكن فكر مقدار الجهد و العمل الذي كرسته في سبيل صناعة هذه اللعبة.

ان تصميم اي لعبة بسيطة لا يعني انه قد تم انهائها بشكل سريع، فكر في الالعاب الضخمة و مقدار العمل و الوقت التي يتطلب لإكمالها بالاضافة الى انه هناك فرق تعمل عليها و لكن الامر يتطلب اكثر من سنه في طبيعة الحال.

ان هذه الورشة تهدف الى تعليمك طريقة تصميم الالعاب و ترتيب افكارك ابتداء من وثيقة التصميم الى بناء اللعبة، واتمنى ان تكون هذه الورشة قد ساعدتك على فهم الكثير في هذا المحرك من جانب التصميم الى البرمجة ثم البناء ففي النهاية تستطيع عمل منصب للعبة ويمكنك البحث في الانترنت عنها.

بهذا الشكل نكون قد اتممنا العمل على اللعبة وانهينا جميع هذه الفقرات يفضل الله القادر على كل شيء ففي النهاية كل ما اطلبه هو الدعاء لي ولوالدي و لجميع المسلمين و اتمنى نشر هذا الكتاب لتعم الفائدة على الجميع و من جانب هو نشر الدروس التي تشرح تصميم الالعاب باللغة العربية، و الى هنا اتمنى ان اراكم في ورشة قادمة بأذن الله، وصلى الله وسلم على سيدنا محمد و على آله وصحبه اجمعين.