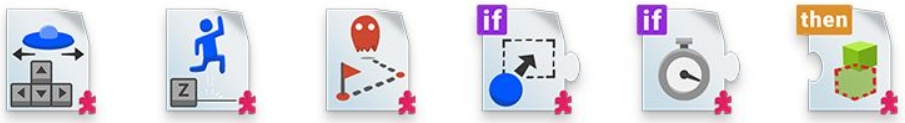
Playground [beta]

Welcome

Whether you're an aspiring maker or an educator in game development: **Welcome to Unity!**
Unity Playground is a framework to create 2D, physics-based games, and it's perfect for teaching beginner game developers to make games in Unity without coding. It can also be used to introduce to game design or level design.

This project is intended to be as flexible as possible, not enforcing a specific game genre apart from being 2D and physics-based. It contains scripts that perform "atomic" tasks (that is, they do mostly only one thing) so you can combine them to create any type of gameplay.



The icons of some of the scripts

Enjoy using the Unity Playground!

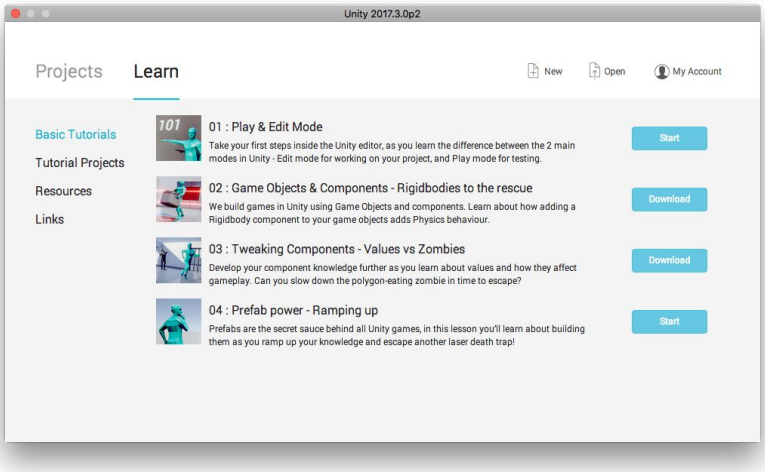
Table of Contents

Getting Started

The Playground is intended to be very simple to get into, and you can get acquainted with the basics in less than 30 minutes. You just need to understand the project structure (see Project Structure) and the concepts that the Playground implements (General Concepts). Then you probably want to take a look at the scripts, starting with the ones grouped in the Movement category.
If you want more information on a specific script, the Reference Guide is a good place to go (on the Learn website, in another .pdf included in the project, or online here).

Prerequisites

This said, before using the Playground you should already have a good grasp of Unity's core concepts. A very good resource are the Interactive Tutorials that you can find in the editor launch window or the Unity Hub, under the Learn tab. As an alternative, you can also go through the Interface & Essentials section in the official Tutorials.



The 4 Interactive Tutorials in the launch window

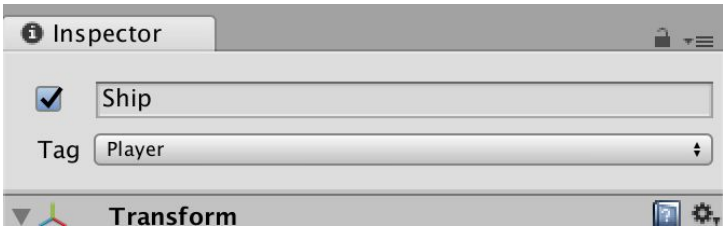
If you are an educator and you are looking to use the Playground for a workshop or a course, you can find interesting pointers in the Teaching section.

Making your first game

Making a game with the Playground is super easy! Let's make a very simple one.

Creating the player

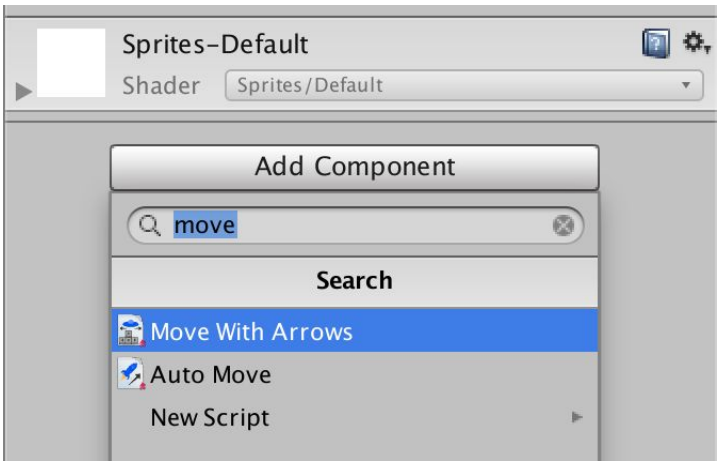
Start by dragging a spaceship Sprite from the Images folder, directly into the scene. Because it's a Sprite, Unity will create a GameObject for you. This is going to be the player, so let's tag this GameObject as 'Player' at the top of the Inspector window.



The object is now tagged as Player

Let's make the ship move now. We need two components: Move to provide the interactivity, and a Rigidbody2D to make sure that the ship follows the laws of physics. Let's drag a Move script from the /Scripts/Movement folder, onto the Inspector of the ship.

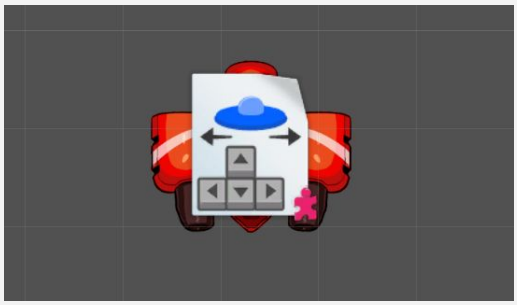
If dragging is a problem, you can also use the Add Component dropdown menu directly at the bottom of the Inspector, then type 'move'. The script should appear as a first result.



The Add Component dropdown in the Inspector

You will notice that as soon as you add the Move script, a Rigidbody2D component is also added. This is because Move requires the Rigidbody2D to work.

Scene Gizmos
It might happen that the icons for the Playground script are huge, covering your graphics.



The Move icon is covering the whole ship!

If this happens, you can reduce their size by using the dropdown Gizmos in the Scene view. Drag the 3D Icons slider to the left until the icons are the right size.

The 3D Icons slider controls the size of the gizmos in the scene

As a first thing, we want to modify the gravity to 0, so the ship doesn't fall down. Also feel free to choose the preferred control method (arrows or WASD), and check "Orient to direction" if it makes sense for your graphics, then choose the orientation axis. Now press Play at the top of the editor to test the game.

You will probably notice that the ship drifts a lot, making it hard to control. Adjust the "Friction" on the Rigidbody2D, putting it to 10.

Play Mode
If you edit component values while in Play mode, you will lose your changes once the game is stopped. Remember to make changes only if you are not in Play mode! You should make changes in Play mode only if you are testing temporary values that you don't mind losing.

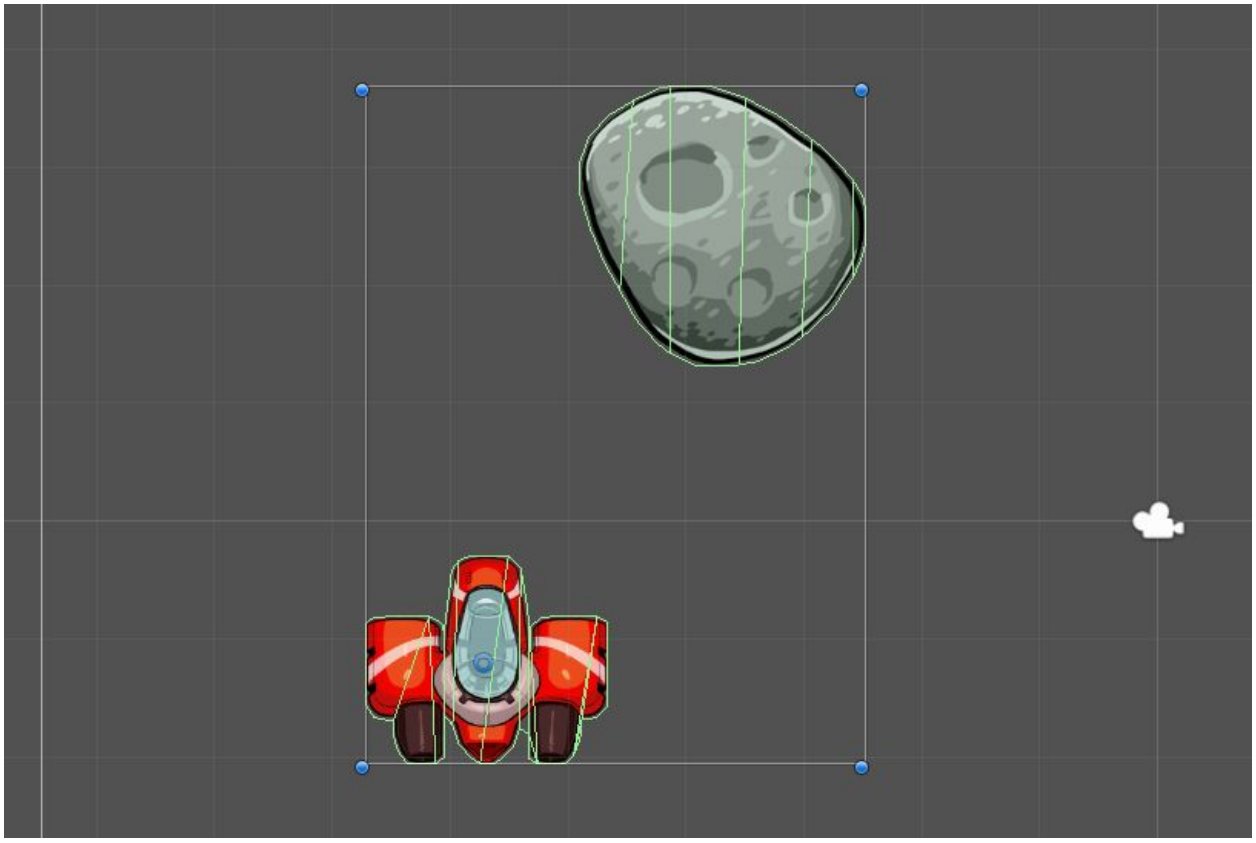
If you play now, you will see that the drift is gone, but the object is also slower, way slower. This is because the force that was moving it is now countered by the friction.

As such, we need to adjust the "Speed" parameter too in the Move component. Bring it to 8, and press Play again. You will see the ship is way easier to control now. We just made our very first gameplay tweak! Congratulations: **you are a game designer**!

Adding obstacles and collisions

We have something playable, but it's not really a game. Let's add some obstacles that the ship has to navigate around by dragging an asteroid Sprite from the /Images folder. Like before, Unity will create a GameObject for us.

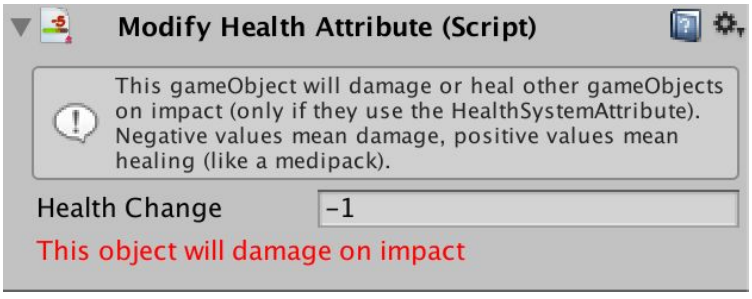
We need to add two components: Rigidbody2D and PolygonCollider2D. The PolygonCollider2D will make it so that this object is able to collide (touch) other objects. We need to add this component to the ship too.



Ship and asteroid show a green border: the collision shape

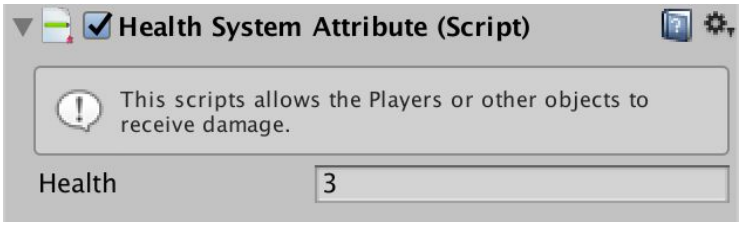
Once colliders are added, try pressing Play. You will notice the ship can now push the asteroid around. Don't forget to tweak the "Gravity" of the asteroid to 0, or it will fall down! Also, feel free to tweak the asteroid's parameter: "Friction", "Angular Friction" and "Mass", to make it behave the way you want. Let's set the "Mass" to 10, so it becomes heavier and doesn't fly away when touching the ship. An asteroid that big must be very heavy!

Let's make this asteroid a threat now. You need to add a script to the asteroid called ModifyHealthAttribute, found under /Scripts/Attributes.



The ModifyHealthAttribute Inspector

Then, we need the ship to be able to detect this damage. To do that, there's another script called HealthSystemAttribute (still under /Scripts/Attributes) that we need to add to the spaceship.



The HealthSystemAttribute Inspector

This not only gives the player health, but allows us to set the initial amount. Finally, duplicate the asteroid (shortcut: Ctrl+D) and create a small asteroid field around the ship.

Prefabs
Before you duplicate the asteroid, you probably want to make it into a Prefab, so you can easily edit all asteroids at once at a later point.

If you don't know what a Prefab is and want to know more, you can read about them in the Manual, or watch a quick video tutorial.

Prefabs are a fundamental concept in Unity if you are making a big game or working in a team, but for now you can also leave them aside. You can focus on your first game, and then come back to them later when you have time.

We don't have any feedback when the player gets hit though. So let's add a UI to visualise the player's health. Drag the UI Prefab from the /Prefabs folder into the scene. You will automatically see a UI popping up in the Game View, with Score and Health.

If you play the game now, you will see that hitting an asteroid will subtract one point of health! And if you hit too many, it's Game Over!

Now that you have a full asteroid field, it's time to test again if you can actually navigate around it without hitting too many. The game has to be difficult, but not too difficult! By repositioning the asteroids and testing the game, you are now basically working on the **game balance** and effectively doing **level design**.

Adding a goal

So what is this game about? Let's say we want the ship to collect some stars without crashing into the asteroids. Once all stars are collected, the game is won. But if you crash too much, it's game over!

Let's add a star from the /Images folder by dragging it into the scene. Add a Collectable script from the /Scripts/Attribute folder. This will make the star a collectable item, so getting one will award one point to the player.

But how do we detect if the star has been collected? We will use collisions again. Add a PolygonCollider2D to it, but this time we want to make it a trigger so enable the "Is Trigger" property.

Triggers
Triggers are a special type of collider. They make an object intangible, so it looks like it can't touch other objects. But Unity will detect when the two objects touch each other, so you can perform actions as a result.

For instance, triggers are very useful to detect if a player has reached the end of the level. Place one just before the exit or end point, and when the player touches it, you display a "You Win" message. The player doesn't see anything, but the trigger provides you (the developer) the logic to detect the winning condition.

If you play the game now, you will notice the star gets collected by the ship. At this point, you might also want to make the star a Prefab (see note about Prefabs above), and then duplicate it as many times as you think it's necessary. Let's say we have 5 in the scene. Now we want to distribute them around so some are easy to get, while the furter ones are harder. This way we also create a ramping difficulty for our little game.

Finally, you want to select the UI GameObject, and on the UI script make sure that the type of game is Score, and that the required score is 5. If this doesn't match the number of stars in the game, then you might actually create a game that's impossible to win!

Press Play again, and check that you can win the game. If you get all 5 stars, you should see a message saying "Player 1 wins!".

Congratulations: **you have just made your first game**. Good job! Keep tweaking it until you're satisfied with the controls, the difficulty, and the level layout.

Next steps

Now that you have an initial grasp of Unity and the Unity Playground, you can go ahead and make more complex games with it. As a first step, you probably want to take a look at the General Concepts of the Playground to understand it better.

Remember: the Playground is meant to be flexible, so come up with your own game genres or copy some existing game, anything is fine!

Tip: When learning how to make games, copying the gameplay of old games from the 80s (like Asteroids, Arkanoid, Space Invaders, Frogger, and so many others…) is usually a good idea, because they were simple. Then as you improve, you can add more and more details and refine the interaction.

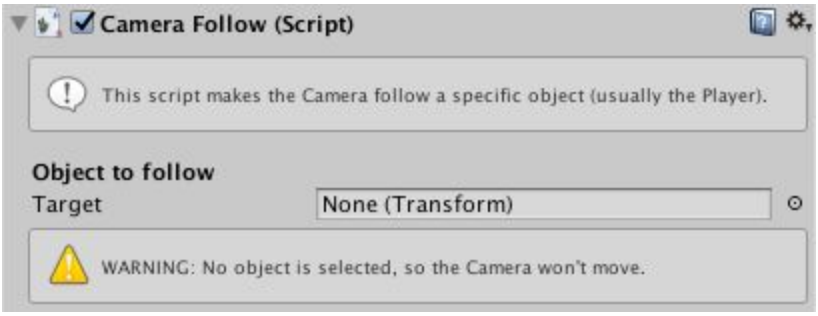
If you need inspiration, open the Examples folder and launch one of the games. Inspect the GameObjects and see how we have made those, then try and create something similar.

General Concepts

In this section, we cover very quickly a few fundamental concepts of the Playground. If you are new to it, you can use them to create more complex games. Before you read through these, we suggest you go through the step-by-step tutorial.

Info and Warnings

All of the custom scripts in the Playground feature a small info box at the top, which explains in a few words what the script does.



Info and a warning in the Camera Follow script

Similarly, many scripts have several warning messages (the ones with the yellow danger sign) which appear is something is not right with the setup. Keep an eye out for these, they might give you a good pointer on why a script is not working.

Collisions and Triggers

Almost all of the logic scripts in the Playground use collisions to create gameplay. This means that things happen when 2 objects with Colliders touch each other, or when an objects with a Collider enters another object (which also has a Collider, but marked as Trigger).

So for instance you can have damage that happens on collision with an enemy, win a level when touching an object, collecting coins and powerup by walking into them, or open a door by smashing the character into it - provided you have the right key!
Similarly, you can trigger a line of dialogue when your character enters an area in front of another character (like in the Roguelike example scene).

When something doesn't work as it should, ask yourself: did I add the appropriate colliders?

Tags

Tags allow us to divide object into categories, so that scripts can perform actions only if they touch the right objects. Without tags, there wouldn't be a way to differentiate between objects.

Like for collisions, many scripts depend on the object to be tagged with the right tag to work. If you are in doubt and your script is not doing what it should, read the Reference Guide to check if you are missing a crucial tag.

Importing Custom Graphics

The Playground contains a lot of nice sprites to play with, located under the Images folder. However, you are free to use any 2D graphics you can find, or make your own!

To use an image in the Playground, you just need to drag it into the Assets folder anywhere. The image will be imported as a Sprite, and then will be ready to be used. Simply dragging it to the Scene or the Hierarchy will create a new GameObject, and then you will be able to use it in the game.

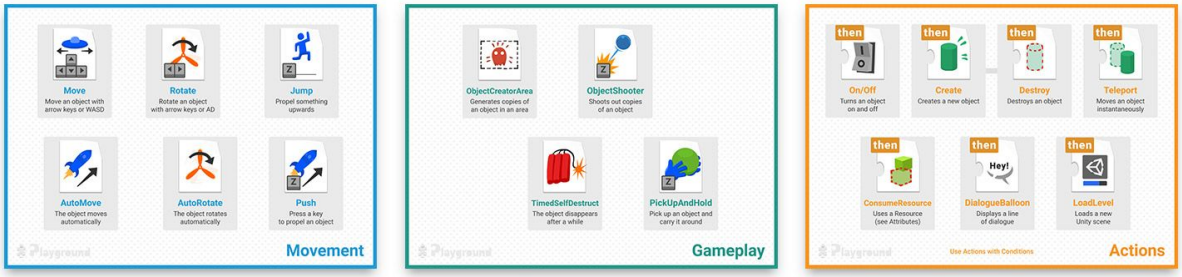
Just remember that to look good, images that are not square need to have **transparency**, otherwise they will display a white background. Good formats that allow transparency are .png or .gif, while .jpg doesn't have it. Also remember, Unity doesn't play animated Gifs.

Advanced Concepts

Are you a teacher or educator? This section is a must-read for you! Also suggested for advanced users who want to understand the Playground better.

Cheatsheets

It might be hard to understand how many scripts are in the Playground at first glance. For this, the Playground comes with a series of “cheatsheets”, that is 6 pages with the icons for all of the scripts and a one-line description. They are organised in categories (Movement, Gameplay, etc.) and are colour-coded.



A preview of some pages

The Cheatsheets are included in the Unity project in both **.pdf**, for printing, and individual **.jpgs** for the screen. In workshops with users very new to Unity, it might be nice to print out a few copies and bind them together, to give the users a sense of the scope and show them all the possibilities they have.

In addition to these, a seventh page features some **extra challenges** of different nature. If you are running a workshop, you can use these to encourage participants to try something new, or give them constraints (kinda like the theme of a game jam).

Project Structure

To use the project in a workshop, you can either have people download it from the Asset Store (for free) or, if internet is an issue, you can manually distribute the Assets and ProjectSettings folders to the students.

Assets folder

The Documentation folder contains **documentation in .pdf**, including this very document. You can use this ("Getting Started") as a starting point, then dig into the details of each script by opening the "Reference Guide". Finally, the "Cheat Sheets" is a document you can print out and distribute to the learners, so they can have a quick glance at the scripts and what they do, for inspiration.

Images and Particles folders contain **graphic assets** which could be used as character, enemies, or to compose the scene, but the developers are free to import new graphics if they so desire.

The heart of the Playground, the **scripts**, is located in the Scripts folder organised by category. Most of them should work out of the box, although some require objects to be tagged in a specific way to work (read more about them in the Tags section).

There's also a folder called Examples, in which you can find a handful of **little games** already made. You can use them as learning material, or as a starting point to customise.

There's a "special" folder in the project called _INTERNAL_. As the name implies, it shouldn't be touched unless you want to mess with the inner workings of the Playground. It contains base scripts, fonts, gizmos, and other things that students shouldn't really care about. It needs to be in the project though, for the Playground to work correctly.

This section goes a bit deeper into some of the parts of the Playground, to better understand its inner workings. Recommended for educators who have to assist students and fix problems in their games.

Tags

Tags are used by some scripts to filter objects and decide when to produce their effects. Some scripts filter objects OnCollisionEnter2D or OnTriggerEnter2D, while others (like HealthSystemAttribute) behave differently depending if the object is tagged as Player or not.

If you import the ProjectSettings folder at the start, some extra tags are already defined. Specifically:

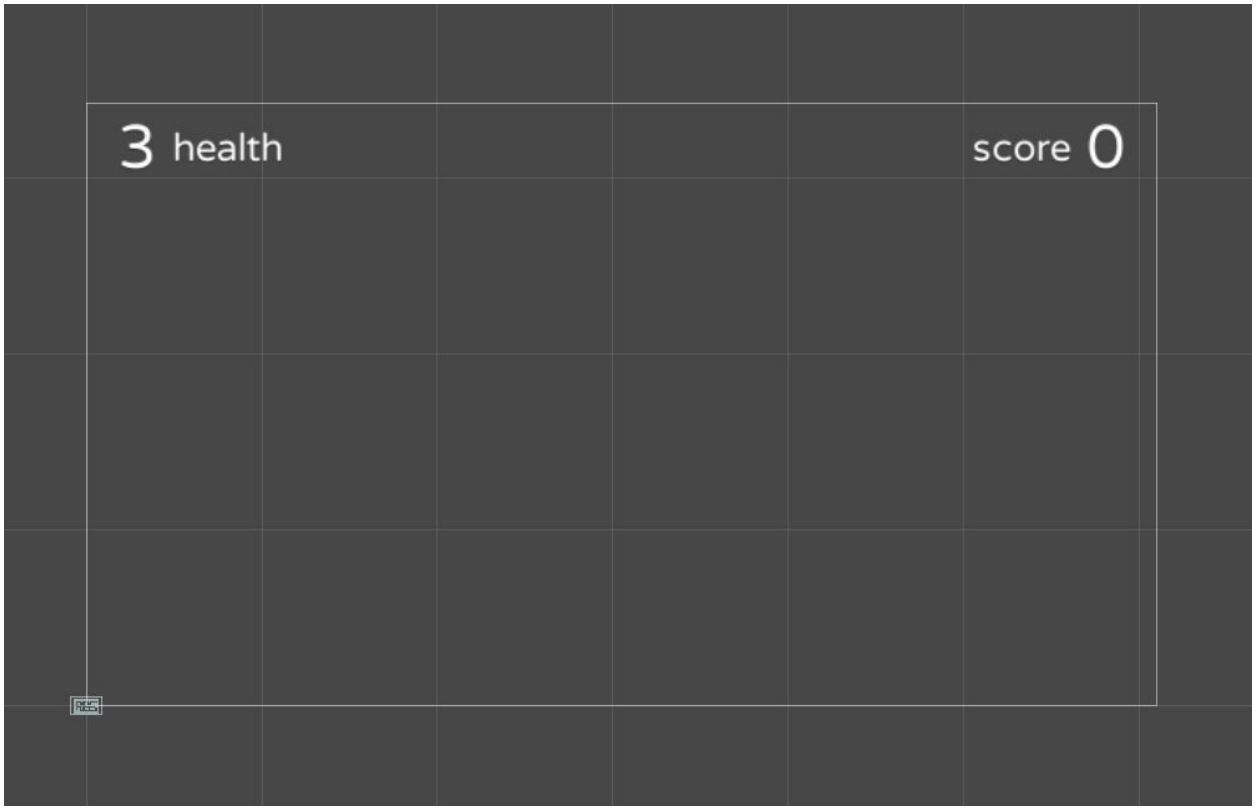
- Player and Player2 have to be used on the two players to enable the UI to work properly, damaging of players, scoring points, etc.
- Enemy, though not currently used in any script, is useful to define enemies on which to apply the effects of bullets and weapons.
- Bullet for projectiles.
- Ground used for checking what is ground when the player jumps.

With the exception of the Player tags, many scripts allow you to define which tag to look for. For instance, the Jump script asks you to define what is considered ground, so you don't necessarily need to chose the "Ground" tag. Take them only as a suggestion.

You can filter Tags in Condition scripts, so you can have something happen only when you collide with objects tagged in a certain way, etc. The tag list updates automatically to show the whole list of Tags so there's no possibility of making typos.

User Interface

The UI is implemented in a prefab, contained in the Prefabs folder. By just dragging the UI into the scene, you automatically get health and score displayed for Player 1, but you can also choose that the game is for 2 players, and then score or health will be displayed for both players, depending on which game mode.



The UI Canvas against an empty Scene background

The UI also allows to define the type of game, between Score, Life and Endless. Depending on the condition, the Game Over and You Win screens will be displayed:

- In Score mode, if the Player reaches the score chosen, the Win screen will be displayed. When two players are present, the UI will only display scores and the first one to reach the score wins.
- In Life mode, if the player reaches 0 health, Game Over will be displayed. With two players, the health of both is displayed. There's no way to win.
- In Endless mode, no end game screen will be displayed and there's no way of winning or losing the game.

Custom game types

When using Condition scripts, you can wire the GameWon and GameOver functions of the UIScript to a UnityEvent. This way, you can create custom win and lose conditions by leveraging collisions and other events.

Similarly, you can tap into the AddOnePoint and RemoveOnePoint functions to do the same with score.

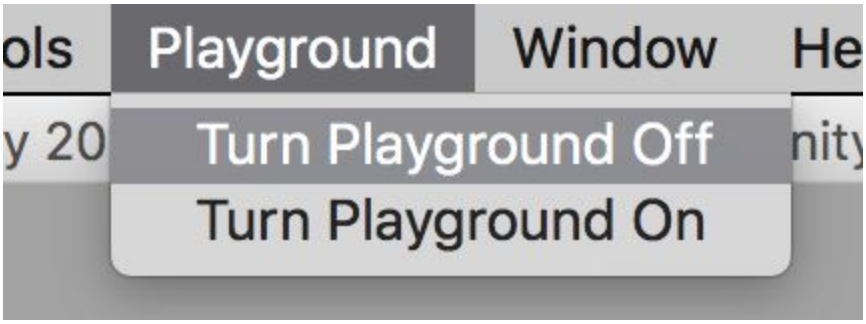
Custom Inspectors

Unity Playground makes heavy use of custom Inspectors, both for the Playground's own scripts (Move, Jump, etc.) and the default Unity Components (Transform, Collider, SpriteRenderer etc.). The goal is to remove some complexity from the Unity UI, giving an easier time to new users.

For this reason, some variable names have been changed too (i.e. Drag in the Rigidbody2D has been changed to "Friction").

Disabling the Playground

You can turn the custom Inspectors on and off from the top menu bar. This allows you to visualise what the Playground scripts are hiding. You can always make your changes, and then turn the Playground back on. Since Playground is only customising Inspectors, it will keep working fine.



Turn Playground off from the top menu bar

This trick is recommended for teachers who need to temporarily make changes to properties that the Playground is hiding (for instance, editing the shape of a Collider of any type).

Tilemaps

TODO

Appendix

Contributing

For questions, suggestions, feel free to email Ciro Continisio (or get in touch through Twitter for quick comments). If you want to contribute to the project, check it out on Github, fork it, and create pull requests.

Note
All suggestions will be considered, but keep in mind that the goal of the Playground is **not** to have as many scripts and features as possible. Having a reasonable amount of content makes it more focused and easier to learn. We believe that once the learners have explored the entirety of the Playground and want more, it's time for them to move on and create their own game from scratch.

Credits

Unity Playground has been created by Ciro Continisio. Graphics by Stefano Guglielmana.

Special thanks to
Unity Technologies, and especially the Brighton Content team for helping with the final push to put the Playground on the Asset Store.
Kenney.nl for the free graphics which have been used for the first iteration of the Playground.
Dioselin Gonzalez, John Sietsma for believing in the project and using it in their workshops first.
Stine Kjærbøll for some precious early feedback, and the tip for the logo design.
Nikoline Høgh and Nevin Eronde for more UX early feedback and enthusiasm.
Phil Jean for suggesting an important change which would be a turning point for the project.
Numerous contributors to the Github repository: Sophia Clarke, Ethan Bruins, Mark Suter, Jim "Jimbo" Picton, "Arche-san", and all the others I'm forgetting right now!
The kids of Coderdojo Brighton and the attendees of New Employee Orientation at Unity, who have been the first involuntary test subjects.
Finally, all the teachers and educators who are using the Playground in any capacity today!