Objects in Unity

How does the game loop works in Unity

What are frames

In gaming, frame rate is the rate in which the image on the screen is refreshed. For example, movies are made up of moving images and most are projected at 24 frames per second, there are 24 still images shown, one after the other, in one second. This creates a very natural feel – most people would not be able to tell the difference between 24fps and 48fps. This value has been carefully chosen for the best experience and space requirements since a 48fps movie file would be twice the size. In games actual objects are updated every frame and everything has to be rendered by the graphics card again and again, so the frame rate does not affect the size of the game but is itself affected by the hardware. A high end graphics card would be able to render more frames in one second than a low end one. Most computer games will be run on a variety of hardware which will render the game in different frame rates, so it is important to for example, make sure that the physics of the game are not dependent on the frame rate. <https://www.lifewire.com/optimizing-video-game-frame-rates-811784>

Physics

Game physics engine is a software that introduces and simulates laws of physics into a game. These laws can either be based on real-world physics or can be entirely new. It simulates the collision detection, rigid or soft body dynamics and fluid dynamics. Unity provides a built-in physics engines for 3D and 2D games that handle the physical simulation. The main components it provides are the Rigidbody2D and the Collider. These are explained in … but in essence, Rigidbody2D allows the game objects to be affected by various forces like the gravity. Collider allows various objects to collide with each other and if paired with Rigidbody, allows objects to affect each other.

<https://docs.unity3d.com/Manual/PhysicsSection.html>

MonoBehaviour

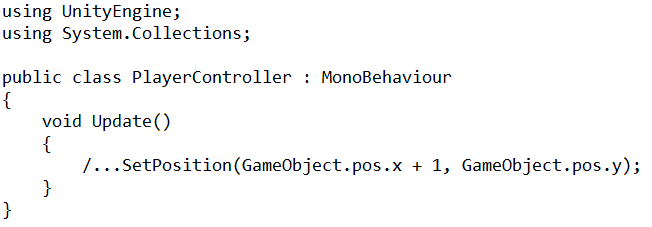
When using C# in Unity it is important to derive every class from MonoBehaviour because it is the base class from which every other class derives. MonoBehaviour has all the important functions that are used by the Unity in order to work properly. The methods are Start, Update, FixedUpdate, LateUpdate, OnGUI, OnDisable, OnEnable.

The last three were not used in this project so there is no need to explain them in detail. OnDisable is called when the object is disabled or destroyed and is usually used for cleanup code. OnEnable is the opposite, it is called when the object is enabled.

OnGUI as the name suggests is used for rendering Graphic User Interface Events like clicking a button on a screen, using GamePad to navigate the GUI etc. Therefore it may be called for each event – several times per frame. This was also not used in this project because the Graphic User Interface was implemented by using the built in User Interface objects, this is explained in ….

The most important functions are Start and the three Update methods. Start method is called in the same frame that the script is enabled and before its Update methods. Start is great to initialize the game objects by enabling the developer to set up the object before the object is updated. It is ran only once in the lifetime of its script. Start is used similarly to a constructor and should be used instead of it when deriving from MonoBehaviour to avoid issues or unexpected results as explained in. <http://ilkinulas.github.io/development/unity/2016/05/30/monobehaviour-constructor.html>

The Update methods are very similar to each other with slight variations in how they are called. The Update method is called every frame and is used, as the name suggests, to update the game object it is attached to. The update is most commonly used method along with the Start, because it allows adding a behavior to an object. For example, if an object has to move from one side to the other, its x or y position could be changed by adding a number to it in the update method. The code in figure … would make the player move by one pixel to the right every frame. The update methods are called only if the MonoBehabviour is enabled.



LateUpdate is just like the Update method – it is called every frame but it is called only once all Update methods, inside the object and inside other objects, have been called. An example where this would be used is when following the player with a camera, updating the camera position should be done only once other objects’ position has been updated to avoid issues and unexpected behavior.

The last method, FixedUpdate, is called every fixed time not dependent on the frame rate. It is mostly used to calculate physics and to avoid issues various issues with time dependent methods. For example if the player has to move right at a certain speed, its position should be updated using the FixedUpdate method. In figure … the speed in which the object would move would be dependent on the frame rate, if the game was running at 25 frames per second, the value would be updated 25 times in one second but if the game was running at 100fps, it would be updated 100 times. Therefore the higher the frame rate, the faster the object would move. FixedUpdate gets rid of this issue by updating the object in a fixed time. This time value can be accessed by using the Time.fixedDeltaTime which shows the in game time in seconds.

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

In Unity scenes contain environments, objects and the menu for the game. Each scene is like a level with its own objects, to easily separate the game into small pieces. For example, almost every object that will be used in the main game level is not needed in a menu therefore there is no need to have it in the game scene.

<https://docs.unity3d.com/Manual/CreatingScenes.html>

In Unity game object is a base class which all objects like the characters, props and scenery in the scene implement. It provides all the necessary functionality that allows the object to be affected and act like containers for other components. For example, attaching controls, Rigidbody and a Collider to an object makes it a playable character. All object come with a Transform component which describes it position in a game scene.

<https://docs.unity3d.com/ScriptReference/GameObject.html>

<https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html>

How User Interface is implemented in Unity

The user interface in Unity is easily implemented by creating a Canvas. All user interface elements should be inside the Canvas area. As explained in … Canvas is a basic Game Object with a Canvas component attached to it. Adding a UI element to Unity automatically create a Canvas and sets it as a parent to that element. EventSystem object has to be added to use as a messaging system otherwise it would not be possible to interact with the objects.

Event system is used to send event messages to various objects by using a mouse, keyboard or other input systems. Event system has a very basic functionality exposed only because it is designed as a manager and coordinator of messages between the modules. Its main objectives are to manage the selected game objects, manage and update the input modules and other like managing Raycasting.

Unity sets up everything that is needed to make the user interface work properly automatically. Default settings were used for this project therefore there was no need to set everything up manually.

In game UI

Game Setup

Designing the UI - HUD

The User Interface is should be easy to use, intuitive and convey lots of information in a very easy to understand way. Modern games range from excessive information on the screen to as little as possible. An example of a game where there is an excessive amount of information on the screen is the Horizon Zero Dawn action role-playing game developed by Guerrilla Games and released in 2017. Figure… shows the UI and as it can be seen, the player is given all the important information on the screen like health points, stamina, weapons, map and quick access items. The game has RPG elements to therefore giving the player all these information is very important. Role-Playing Games have a much more focus on the game mechanics and the game is driven by the player. That is, the player makes all of the decisions, chooses what to do, the player character skills and usually makes in game decisions which steer the story in a unique direction.

On the other end there are games that are more like the movies, which means that in a way they play themselves. Uncharted 4, an action adventure game developed by Naughty Dog and released in 2017, is an example of a game where the player does not have too much impact on the game world and almost every event is scripted. This allows the game designers to have as little information on the screen at one time as possible. In fact, most of the time the HUD is hidden except the times when the player is using weapons.

The game in this project will be in the middle, there will be enough information for the player to understand what is going on but not too much where the player is distracted. This is especially important because the game is very fast paced and competitive so the player will only be able to take a brief look at the HUD every some time. It takes an inspiration from older arcade games like Donkey Kong or Space Invaders. It will also be kept as basic as possible to avoid spending too much time on implementation since only few days will be given for that. The plan is to have a bar for health, stamina and shield in one of the top corners of the game and the weapon and ammo in one of the bottom corners.

The Menu will be broken down into four scenes; Title Screen, Game Setup, Main and End Game.

Title screen will be the first thing the players see when they open the game. It will be kept very basic, with a name and a logo of the game, creator name and start button. It is also planned to have a theme song playing in the background just like in the games mentioned above since this makes the game much more memorable and keeps with the theme of arcade games.

Start button in the Title screen would send the user to the game setup screen; here the user will be able to setup the settings of the game to his likings. User will be able to change settings like the maximum game score, time of the game, starting stats (health, stamina etc.), type of opponent (another player or CPU) and other, more trivial settings like shield size or ball size. Since the user will see this screen after every match, the total game score for each team will also be shown here.

Game setup has been taken from more modern games like Call Of Duty where players are given an option to create custom games that fit their play style. In arcade games there were usually fixed settings of the game settings will be preset to default settings which will be tested and which the game will be designed around. It is also planned to give user an option to save the settings of the game and the game score to a file although this will not be a priority since time will be very limited.

The End Game screen will be kept very simple; it will have the name of the team that won the game or “Draw!” if there is a draw. It will also have the score of the current game and the Continue button which will send the user back to the Game Setup. This will be like a transition screen between the game and setup screen so there is no need to add too much information.

Additional

The user will be able to transition between the game scenes by using the available buttons. Figure shows in detail which scenes the user will be access from each scene. One thing worth pointing out is that the user should not be able to access the End Game screen from other scene than the Main game. End Game screen shows the end of the game scores which are unknown until the end of the game.

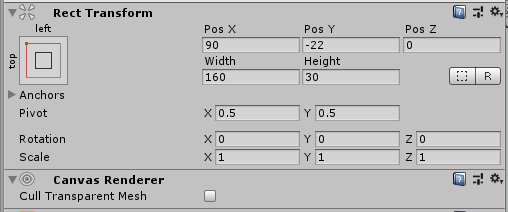
<https://guides.gamepressure.com/horizon_zero_dawn/guide.asp?ID=38878>

Implementing the UI

The first User Interface that has been implemented is the Head-Up Display (HUD). The information that was meant to be shown to the user as mentioned in the design has been implemented but had to be simplified. Instead of a rectangle or a bar, a numerical value is shown for the health, stamina, shield and ammo. Unfortunately there was not enough time to have two separate HUDs for player vs. CPU and player vs. player, game modes. So, instead one of the sides is for the first player and the other, right, side is for the second player/CPU. It was useful to do it that way when implementing the game because it allowed seeing what the AI’s status is.

However, this way is very limited because if there would be more than two players on each team, it would be hard to find space to show each player’s status. The best solution would be to have a separate HUD for each of the game modes; one HUD would have the AI’s opponent health only displayed above its character and hide everything else the other HUD would be designed for a player vs. player and have the HUD as it is but show the weapon and ammo in the bottom corners. FIGURE

The HUD has been implemented in a very basic way – a Canvas and Event System object has been created and default Text objects have been added as a child of the Canvas object. The Text objects allow positioning the text with the Rect Transform component and to add text by using the Text script component. The settings have been kept default with only the font and font size changed.



To show the score a new game object has been created with a Score UI Controller component inside. It:

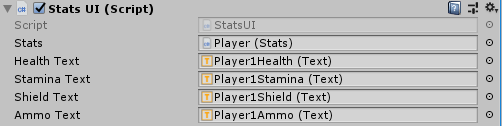
* takes two TeamController components
* extracts the scores

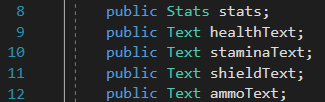


* creates a string out of them
* passes them to a Text component

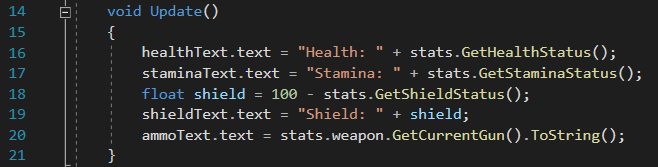


Stats have been done similarly, a StatsHUD object with two StatsUI components for each player have been created. StatsUI is very similar to Score UI Controller but it takes StatsUI object, which gives all of the player’s statistics like health or stamina, and turns them into a string and passes to Text objects accordingly. Figure … shows a Stats UI object with everything set up. As it can be seen it is very general and can be easily duplicated in case the teams are expanded.



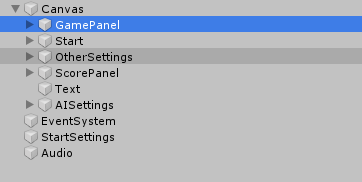


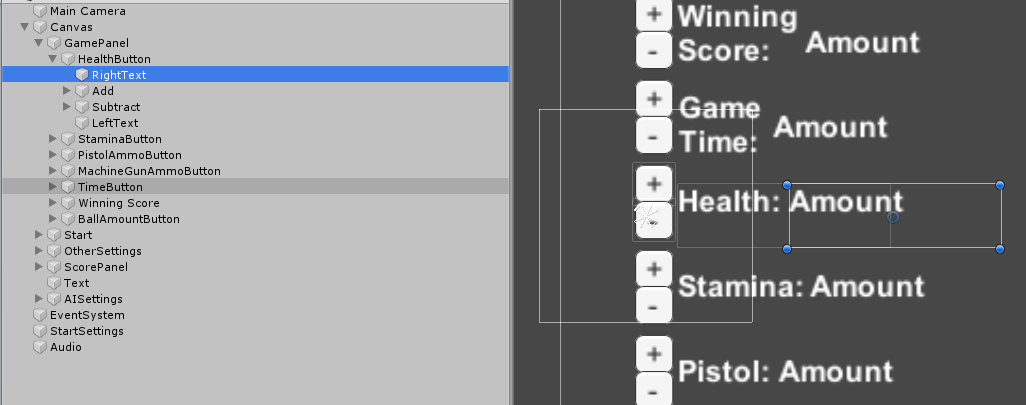
Figure… shows the update loop which is very basic since it only has one task – to convert the status into a string.

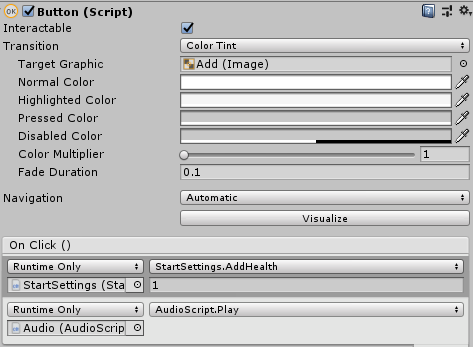
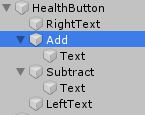


The Game Scene

The game settings have been set up very similarly to above, update method creates a string from the status info that it takes from the GameStartSettings component. The







GameStartSettings component is a static class that holds the information of the game as static variables. This was implemented that way because this information will have to be passed between the game scenes and while Unity gives an option to pass information between the scenes, this was the fastest and easiest way to implement it. It plays a role of a Configure file.

GAME MDOES

How many maximum players

Player vs player

Player vs cpu

Issues