

WITCH ESCAPE

User Manual

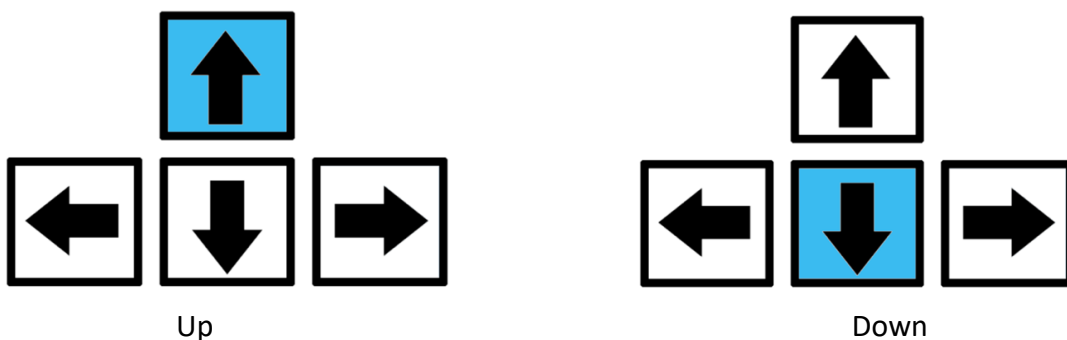
A long time ago, in a faraway place, a witch has cast a spell on our hero, the Prince, turning him into a ghost and scattering his limbs across the village. Help the Prince on his quest to find his limbs and become human again. Along the way you will encounter monsters with whom you will battle, increasing your strength as you rediscover your missing limbs. Move up and to the right to increase the level.

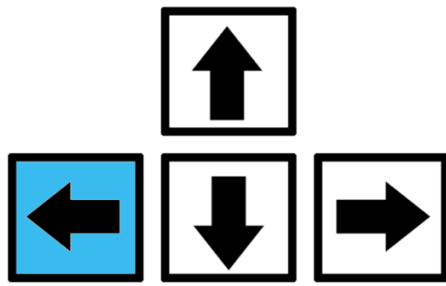
At the title screen, press **START** to proceed to the game, or **INSTRUCTIONS** for an overview of the game controls



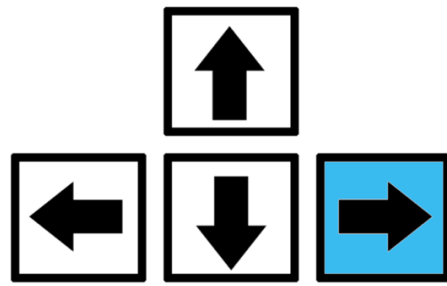
Dungeon Controls

Witch Escape utilises simple keyboard controls and mouse clicks. During the dungeon levels, direction is controlled using the **Arrow** keys on your keyboard or laptop.





Left



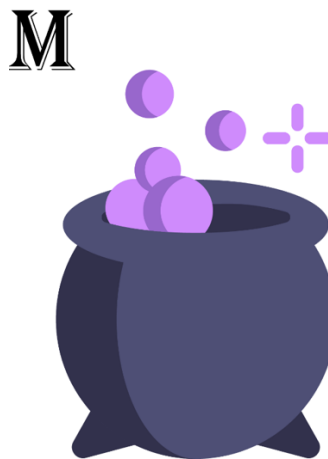
Right

Battles

During card battles, cards are selected by using the mouse. Drag cards from the row at the bottom of the page, into the grid on the left side. In battle, you create a spell by combining cards, increasing the spell's power. There are three types of cards:



Using this card blocks your opponent from making their next move.



The Magic card buffers the cards around it, increasing their score.



The Skull card does not have special properties, but carries a higher score.

The enemy will take an action between each of your card drops, increasing their strength, decreasing yours, or both. You win the fight if your score is greater than the enemies after all nine cards have been placed.

The game ends after you fight the witch (win or lose), or you lose all your lives in battle.

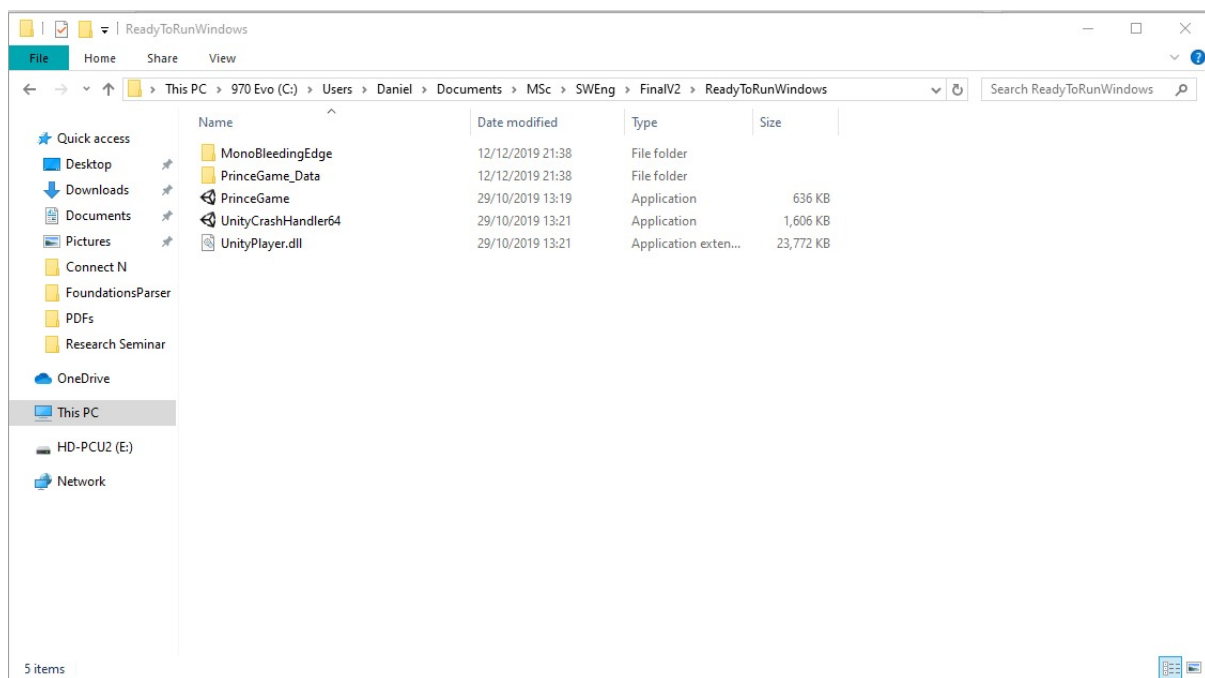
Installation Guide

A copy of Witch Escape can be downloaded from <https://github.com/Daniel-C-Edwards/Software-Engineering-Prince-Game>

Please select and download the appropriate version for your operating system. Once the game file has downloaded, please follow the instructions below:

Windows

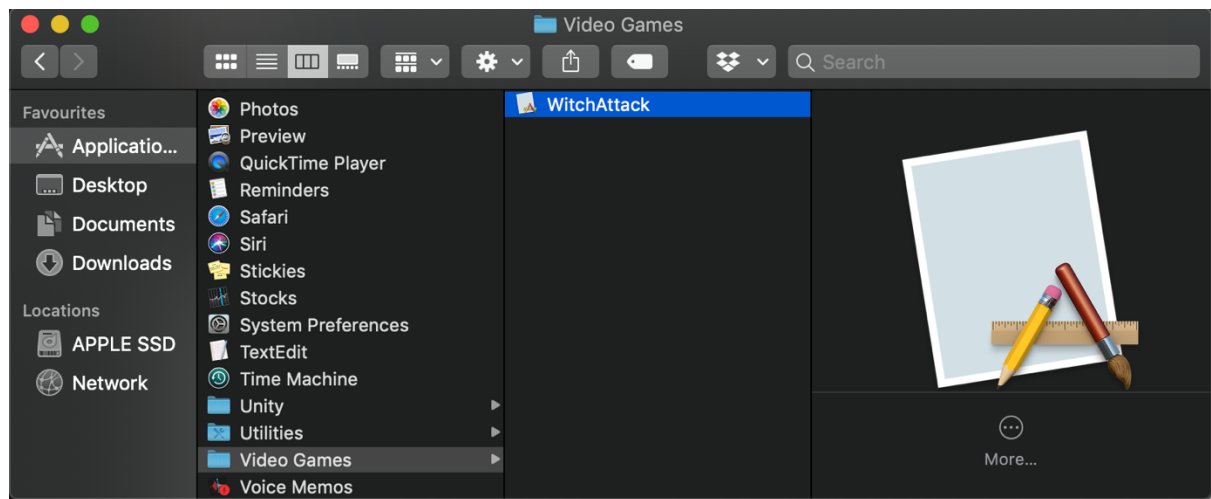
To run, unzip the ReadyToRunWindows.zip into your desired install location. From here, double click the application icon to run the game. From here you can create a shortcut to **PrinceGame** and place this on your desktop or another convenient location.



Remember to ensure that the application files contained within the original .zip folder remain in the same folder on your computer.

MacOSX

Witch Escape will download as an Application. We suggest moving the **PrinceGameOSX** application to your Applications folder, as shown below:



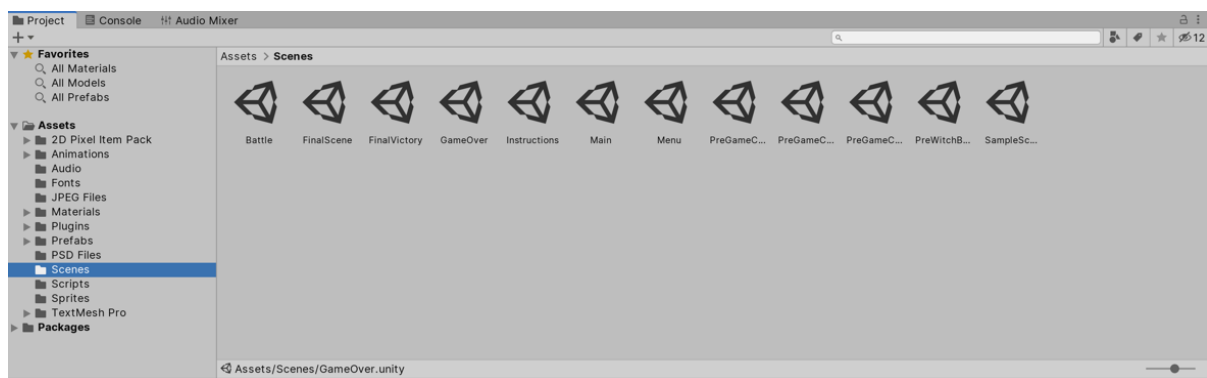
Right click the game and select **Open** to proceed to the game's main menu.

Maintenance Guide:

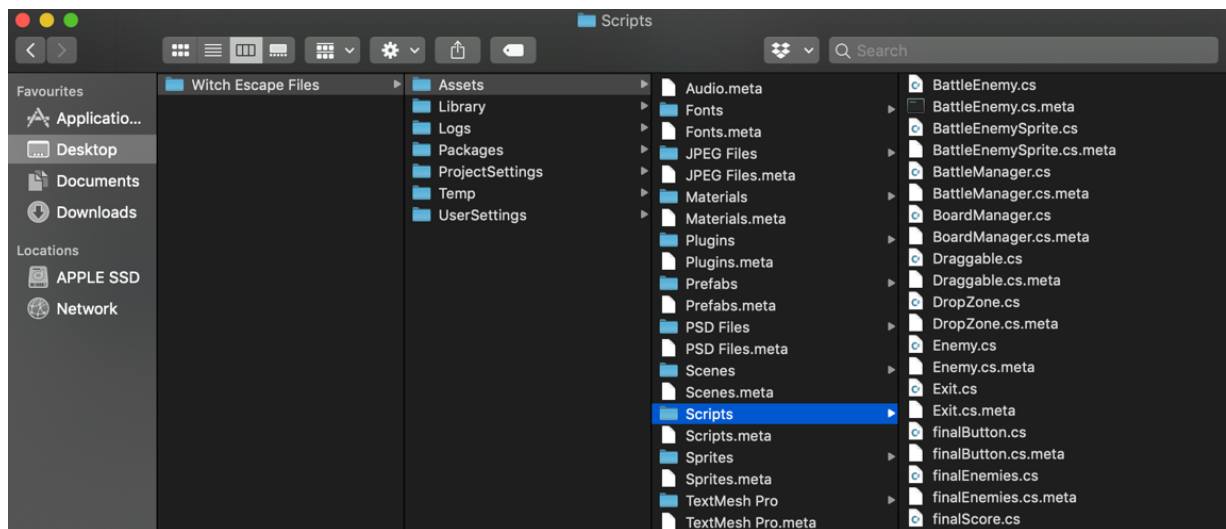
Witch Escape has been developed in Unity, using the development language C#. Whilst the game code is robust and should function without failures, there are opportunities to expand and refine the game if you have a little coding experience. We recommend installing Unity, which can be downloaded from: <https://unity.com>

Asset Locations

There are two methods for editing Witch Escape: In Unity, using the Prefabs, Scenes and Sprites found in the **Assets** folder; or via an external development environment (such as Visual Studio Code) using the aforementioned **Scripts**. Scripts can also be found in the **ASSETS** folder in Unity, or alternatively in the system folder containing the Unity files. Scripts can also be found in the **ASSETS** folder in Unity.



Navigating Assets in Unity



Locating Scripts within a file system

Editing in Unity

Perhaps the simplest way to edit game objects, character actions and level architecture is through the **Scenes**, **Sprites** and **Prefabs** files found in Unity.

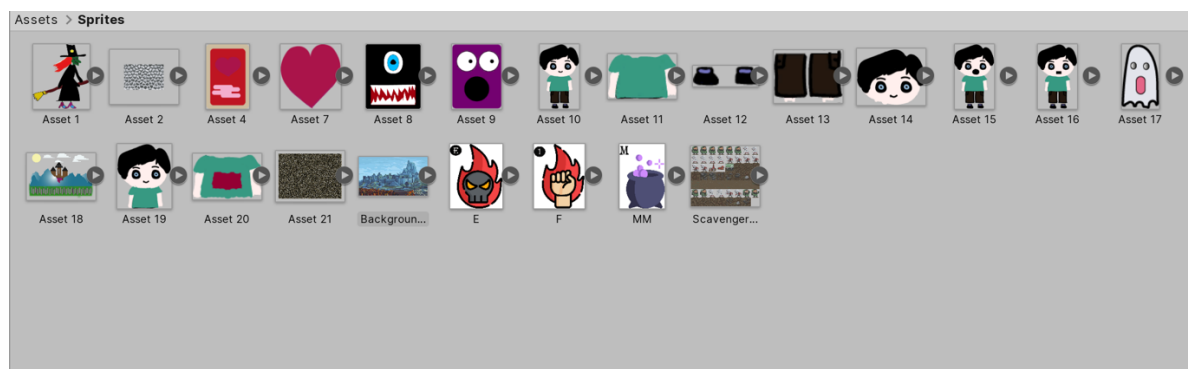


Example of editing tools found in Inspector.

An efficient way to change the behaviour of game characters is to locate the relevant **Prefab** and then navigate to the **Inspector** sidebar. From here, you will find Script settings which can be quickly and easily adjusted without an in depth knowledge of game development. The example shown here enables you to edit the speed, damage and points associated with the Prince colliding with an enemy.

For beginners, editing existing game assets or adding new Prefabs from the Unity store may be a good way to get to grips with modifying a simple game.

Where Prefabs are concerned with the way a game object or asset behaves, accessing an individual **Sprite** will give you access to the visual attributes. Each Sprite provides functionality which enables you to change a character's appearance, create new cards, or import new backgrounds. Selecting a Sprite will give you access to the attributes in the inspector pane.



Contents of the Sprites folder

Scripts – External Editing

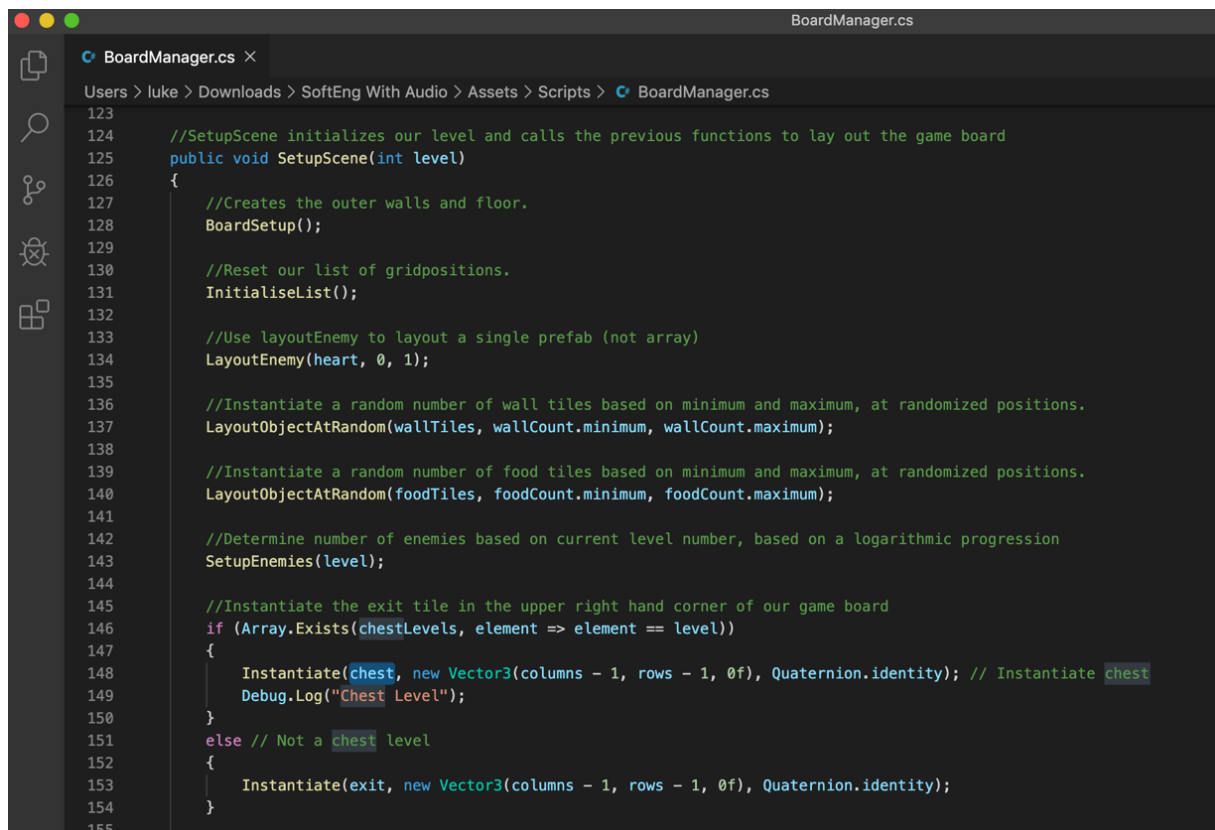
If working in Unity, each C# script can be found in Assets > Scripts, either from Unity's navigation pane, or your computer file system. To edit these files you may require additional development software. We recommend Visual Code. Visual Studio Code can be downloaded from <https://code.visualstudio.com/>

There are several notable edits which can be made via the games **Scripts** which will enable you to change the game more dramatically, from the length of the game to the game difficulty.

Board Manager Script

Accessing the **Boardmanager.cs** script will provide access to 'Chest Levels' – the levels in which the Prince collects his limbs. Editing variables here will affect the length of the game.

Witch Escape is a procedurally generated game. Modifications to the size or shape of the game, or the character's route through it can be made by accessing the **Game Manager** Prefab and then the **BoardManager.cs** script, which handles procedural generation.

A screenshot of a Visual Studio Code editor window showing the BoardManager.cs script. The window title is 'BoardManager.cs'. The file explorer on the left shows the path 'Users > luke > Downloads > SoftEng With Audio > Assets > Scripts > BoardManager.cs'. The script content is as follows:

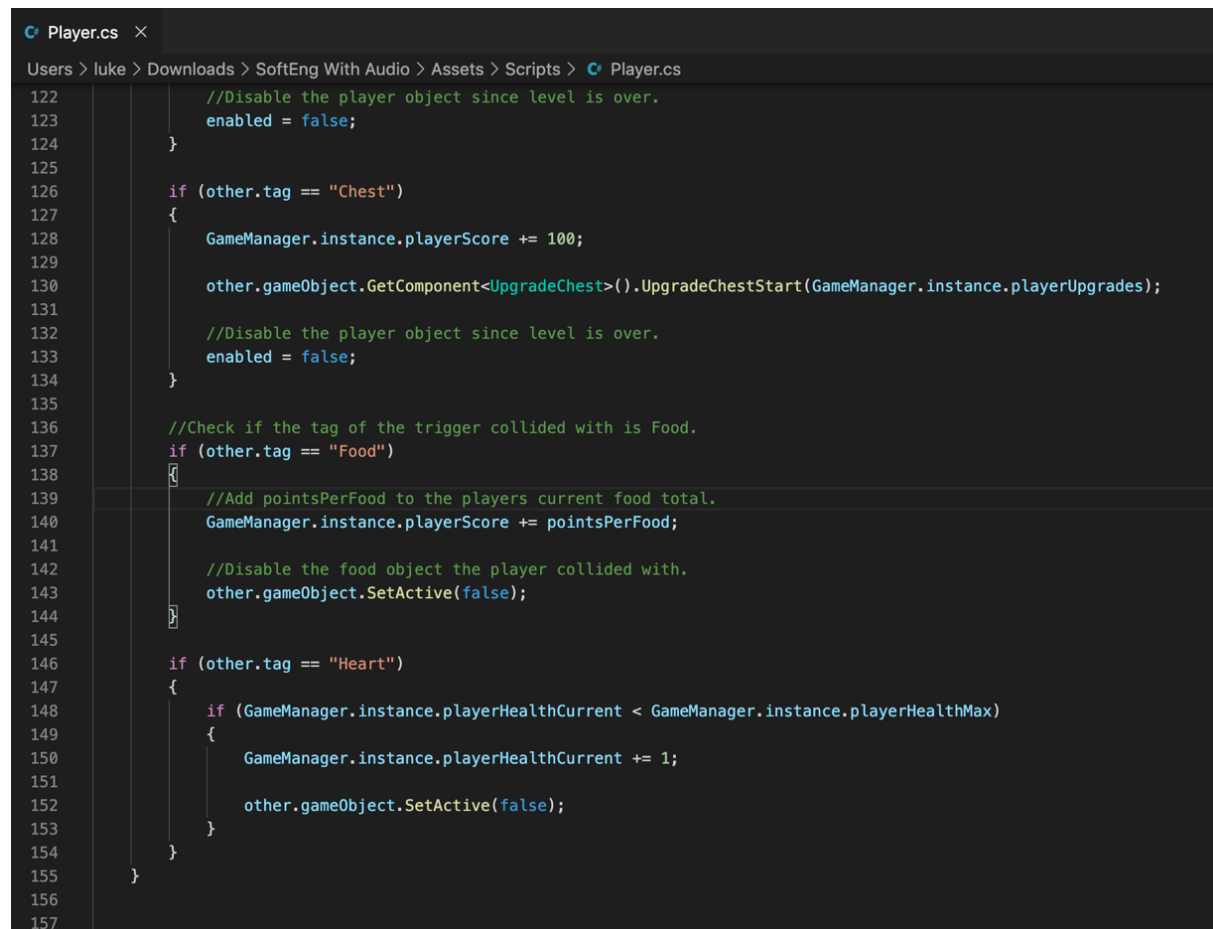
```
123
124 //SetupScene initializes our level and calls the previous functions to lay out the game board
125 public void SetupScene(int level)
126 {
127     //Creates the outer walls and floor.
128     BoardSetup();
129
130     //Reset our list of gridpositions.
131     InitialiseList();
132
133     //Use layoutEnemy to layout a single prefab (not array)
134     LayoutEnemy(heart, 0, 1);
135
136     //Instantiate a random number of wall tiles based on minimum and maximum, at randomized positions.
137     LayoutObjectAtRandom(wallTiles, wallCount.minimum, wallCount.maximum);
138
139     //Instantiate a random number of food tiles based on minimum and maximum, at randomized positions.
140     LayoutObjectAtRandom(foodTiles, foodCount.minimum, foodCount.maximum);
141
142     //Determine number of enemies based on current level number, based on a logarithmic progression
143     SetupEnemies(level);
144
145     //Instantiate the exit tile in the upper right hand corner of our game board
146     if (Array.Exists(chestLevels, element => element == level))
147     {
148         Instantiate(chest, new Vector3(columns - 1, rows - 1, 0f), Quaternion.identity); // Instantiate chest
149         Debug.Log("Chest Level");
150     }
151     else // Not a chest level
152     {
153         Instantiate(exit, new Vector3(columns - 1, rows - 1, 0f), Quaternion.identity);
154     }
155 }
```

Board Manager Script

Player Script

Editing the **Player.cs** script will allow you to dictate the player characteristics – movement, speed etc, as well as game points collected when the player acquires prizes located in the dungeon levels.

From the **Player.cs** script you will also gain access to the ‘chests’ which contain items and limbs, allowing you to upgrade chests with new items, or using random values for added complexity and unpredictability.



```
122 //Disable the player object since level is over.
123 enabled = false;
124 }
125
126 if (other.tag == "Chest")
127 {
128     GameManager.instance.playerScore += 100;
129
130     other.gameObject.GetComponent<UpgradeChest>().UpgradeChestStart(GameManager.instance.playerUpgrades);
131
132     //Disable the player object since level is over.
133     enabled = false;
134 }
135
136 //Check if the tag of the trigger collided with is Food.
137 if (other.tag == "Food")
138 {
139     //Add pointsPerFood to the players current food total.
140     GameManager.instance.playerScore += pointsPerFood;
141
142     //Disable the food object the player collided with.
143     other.gameObject.SetActive(false);
144 }
145
146 if (other.tag == "Heart")
147 {
148     if (GameManager.instance.playerHealthCurrent < GameManager.instance.playerHealthMax)
149     {
150         GameManager.instance.playerHealthCurrent += 1;
151
152         other.gameObject.SetActive(false);
153     }
154 }
155 }
156
157 }
```

Example shows access to ‘Chests’

Creating Additional Enemies

To create additional enemies, start by copying one of the existing enemy prefabs and edit that. Change the Enemy Class string to their new identifier, and experiment with the values relating to Player Damage, Victory Score and Starting Value (The power value the enemy starts at in battle).

To spawn a new enemy, you will need to add it to the Enemy Tiles array on the Board Manager (Game Manager prefab). You will then need to navigate to the **boardmanager.cs** script, and add lines to the **SetupEnemies** function. The next step is spawning **enemyTiles[3]**. Finally, you will need to navigate to the **battleEnemy** script, and add the AI for your new enemy in battle. We recommend starting by copying one of the existing loops in the **EnemyTurn** function and experimenting with inputting new values.