

A LEVEL

Exemplar Candidate Work

COMPUTER SCIENCE

H446

For first teaching in 2015

**H446/03 Summer 2017
examination series
Set A – Medium**

Version 1

Contents

Introduction	3
Exemplar 2	4
Commentary	109

Introduction

These exemplar answers have been chosen from the summer 2017 examination series.

OCR is open to a wide variety of approaches and all answers are considered on their merits. These exemplars, therefore, should not be seen as the only way to answer questions but do illustrate how the mark scheme has been applied.

Please always refer to the specification (<http://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf>) for full details of the assessment for this qualification. These exemplar answers should also be read in conjunction with the sample assessment materials and the June 2017 Examiners' Report to Centres available on the OCR website <http://www.ocr.org.uk/qualifications/>.

The question paper, mark scheme and any resource booklet(s) will be available on the OCR website from summer 2018. Until then, they are available on OCR Interchange (school exams officers will have a login for this).

It is important to note that approaches to question setting and marking will remain consistent. At the same time OCR reviews all its qualifications annually and may make small adjustments to improve the performance of its assessments. We will let you know of any substantive changes.

Exemplar 2 – Set A (Medium)

Programming project (non exam assessment)

Learners will be expected to analyse, design, develop, test, evaluate and document a program written in a suitable programming language.

A2 Computing Project

Contents

Section 1 - Analysis	4
Problem Definition	4
My Clients	4
Current solutions	5
Limitations	6
Diagram of current systems	7
Investigation and analysis	8
Specification Requirements	11
Software & Hardware Requirements	12
User requirements	12
Section 2 - Design	13
Design objectives	13
Graphic Design	15
Interface Design	20
Main Menu scene	20
Game Scene	20
Paused Menu Scene	21
Game Over Scene	22
Final Designs	22
Modular Design Diagram	23
Use Case Diagram	25
Data Structure Design	25
Algorithms	28
Testing data	31
Section 3 – Developing the coded solution	32
Game Screen	32
Player	32
Camera	45
Enemy	46
Game Controller	48
Main menu	49
Game Over	50
Testing	52
Screenshots	54
Result of Testing	68
Final Settings and Code	73
MenuScreen	73
GameScene	78
GameOver	87
Scripts	91
Section 4 - Evaluation	103

Section 1 - Analysis

Problem Definition

Year 12 and year 13 students are working extremely hard to prepare for A level exams to gain high qualifications. Students spend many hours in school working and when they go home they want something to relax and disconnect from their studies for a while. I have read articles that cite research studies talking about how videogames help people relax and reduce level of hostility, an extract from the study: "In this study, 103 young adults were given a frustration task and then randomized to play no game, a nonviolent game, a violent game with good versus evil theme (i.e., playing as a good character taking on evil), or a violent game in which they played as a "bad guy." Results indicated that randomized video game play had no effect on aggressive behaviour; real-life violent video game-playing history, however, was predictive of decreased hostile feelings and decreased depression following the frustration task. Results do not support a link between violent video games and aggressive behaviour, but do suggest that violent games reduce depression and hostile feelings in players through mood management." From The Hitman Study

I asked class mates and students in other sixth forms what did they do to, and found that a large majority of them played videogames up to a certain level, from mobile games to full simulation games. What I can understand from this is that most if not all students are gamers.

The problem is that in recent years, computer games have shifted away from simple and challenging games to games with a wider range of game mechanics and intricate stories but easier to complete, giving opportunity to give in game achievements or rewards quite readily. I have investigated more into this by asking class mates and students from other sixth forms to get an understanding of how gamers feel about this shift towards a lack of new challenging classic styled games. From this, I concluded that from the perspective of a 16-19 year old, on the last five to six years' games have gotten less challenging to fit the broadest audience apart from a couple of exceptions. Then I went to ask what kind of game they would play if they had a constrained time. Most of them said that they did not have time to play a story-based game and rather would play a simple, fun but hard game. Therefore, my proposal is to make a game that will be simple yet engaging and interesting thus filling in this gap.

My Clients

My users are a group of students between the age of 16 and 19 that are interested in videogames. They want a game that is straightforward to play as supposed of some very complex solutions that are available, but it still is fun, challenging and has good game mechanics and playability. They also wanted a game that you are able to open it and play and didn't have to worry if you did not have enough time to finish the game or didn't want to bother with progress and story. They will use the game in their personal computers for enjoyment. Therefore, I will develop a simplified structure with a similar playstyle but with the same playability. I will produce a survey for my clients this way I can make sure I have met the user requirements, and get a list of basic requirements. I can further develop and refine these requirements by doing one-on-one interviews.

Current solutions

'The Binding of Isaac', it is a top-down 2D dungeon crawler game where the player controls the character through random and procedurally generated levels in which it will fight monsters to continue to the next room. It includes an item system: equipment is used to increase stat points; weapons with different abilities; and single use objects which when used have immediate effect. This style of game does allow pausing the game but it cannot save between levels, so every time you finish the game or lose you start from the beginning losing all progress. When you win, you don't gain anything in particular because the game is based on achievements such as: "Have seven or more Heart Containers at once" or "Defeat Satan", which unlocking game characters, these achievements can only be done if the game is beaten multiple times.

In game: player far right



"The Legend of Zelda: The Minish Cap" is another game with similar play style, it differences in that it is a story driven adventure game with a set map, parts of which have to be unlocked through completing objectives. The object system is based on tools and weapons that once unlocked are yours to keep, most of them are awarded after completing objectives but some have to be purchased from the shop or found in a chest. Acquiring a new object allows you to further progress in the game.



Analysing both games I extracted features, which I believe, made the games succeed in the industry. Firstly, the games although they had very basic graphics compared to what we are able to produce, showing that high quality graphics are not necessary for a game to be good or popular. What is obvious is the intuitive and simple mechanics, which made the game very friendly in terms of difficulty. Both games get straight into gameplay after a short story introduction, this made the games quite straightforward. Other games which are more story driven, have many cinematic cut scenes which from what I talked to my clients and personal experience, might make some end users get bored and skip the story part, making it irrelevant and redundant. Finally, both games have an equipment system which allows for the game to be played longer as difficulty increases, furthermore each upgrade gives a sense of reward and accomplishment for all the time they spent killing monsters and earning currency.

Limitations

Of course, I need to take into account any group of potential clients that won't be able to use the software. People with visual impairment will not be able to navigate the user interface therefore making it impossible to use the software. In order to cushion this I can have high contrast shapes and letters, with vibrant colours so people with less severe visual impairment are able to use the software. All the inputs are taken via the mouse and keyboard, meaning people with missing hands such as an amputee won't be able to input data to the software therefore making it quite complex for them to use the software. To minimize this I will make sure I keep all different inputs to a minimum number of keys to make it more accessible for handicapped clients that want to use the software.

Diagram of current systems



This stripped-down diagram shows basic functionality of most current solutions for games in this genre. This is something I will refer to when designing and developing the application.

Investigation and analysis

I plan to interview my 10 clients throughout the investigation by using questionnaires, in the first questionnaire I wanted to get an idea of what they were most interested of this genre of game, so I decided to ask these questions:

1. Have you ever played a roguelike/dungeon crawler game before?
2. If applicable, how many different games in this genre have you played?
3. What components of the game did you like?
4. What components did you dislike?
5. Any features that you have not seen you would like implemented in a game of this genre?
6. Comments and ideas of games (Plot, setting, style of gameplay, etc.)

Results of first questionnaire

Have you ever played a roguelike/dungeon crawler game before?

- Yes [10]

If applicable, how many different games in this genre have you played?

- 1 [4]
- 2-3 [4]
- +4 [2]

What components of the game did you like?

- Art [2]
- Music [2]
- Fast paced [1]
- Lots of items [4]
- Character size [1]
- Character selection and customisation [3]
- Story [2]

What components did you dislike?

- Difficultly, too easy or too hard [4]
- Repetition of types of quests and enemies [2]
- Resolution is too low [3]
- Restricted access to some areas [1]

Any features that you would like implemented in a game of this genre?

- Multiplayer [1]
- Good interface [3]
- Custom characters [2]
- Quick saves [1]
- Infinite mode [1]
- Cheat codes [1]
- Large pool of mechanics [2]

Comments and ideas for the game (Plot, setting, style of gameplay, etc.).

- Horror [2]
- Fast paced [3]
- Different Enemy types [1]

Analysis of the first questionnaire

From the first and second question, I can gather that all my interviewees are in some level of understanding with this genre of game and can help me to further advance in the right direction and improve my end product. My interviewees liked a range of features of the games they played; I will intend to please the entirety of the request, but within reason as long as the features don't interfere with each other. There might be some features which are out of my reach and understanding so I will try to come as close as I can to the desired feature.

Questions three to six were completely open, because I didn't want to make them to constrain their answers so I decided to give a space where they could freely write; this did mean that some answers were poor and not in much detail, some even being empty. The purpose of my questionnaire was to get an idea of what to aim for and make my first set of specification requirements, as I was unsure of the minimum specification I needed.

From the questionnaire, I generated this list of base specification requirements, taking into account compatibility with each feature:

- User inputs control the actions of the character.
- 2D Design
- The character can move in four directions: up, down, left and right.
- When the character collides with enemy or harmful object, the player's health reduces.
- When the character hits the enemy with a weapon, the enemy's health reduces.
- When an enemy is defeated, points are awarded.
- The game progresses in waves: a number of enemies will appear after they are all defeated another wave of increasing strength and number of enemies will appear.
- Game ends when character's health is depleted.
- Points are stored under highscore if the score was higher at the end of the game.

Second questionnaire

For my next questionnaire, I will have multiple-choice questions instead of spaces to write which will give me answers that are more accurate so I can focus in making a more refined specification requirement list. The answers are going to be quite concise so to compensate for this I will have a wider range of questions addressing all the majority of parts, features and characteristics.

1. How big would you like the game screen to be?
2. How would you like to control your character?
3. How would you like the view angle to be?
4. Would you like different classes?
5. Would you like to unlock new abilities when a certain wave is reached?
6. How would you like the item system to be implemented?
7. How much health in comparison with enemies would you like to have (scalable)?
8. How many hits should an enemy take to kill (medium difficulty)?
9. How would you like the learning curve to be? (how quickly it gets hard from the start of the game)
10. What kind of music would you like?

Results of second questionnaire

How big would you like the game screen to be?

- Full screen [9]
- Windowed [1]

How would you like to control your character?

- Mouse [1]
- Mouse and keyboard [8]
- Keyboard [1]

How would you like the view angle to be?

- Bird view [2]
- Oblique [8]

Would you like different classes?

- Yes [9]
- No [1]

Would you like to unlock new abilities when a certain goal reached?

- Yes [10]

How would you like the item system to be implemented?

- Items dropped by monsters and objects [6]
- Monsters and objects drop gold which can be used to buy items [4]

How much health in comparison with enemies would you like to have (scalable)?

- Enemy variance [10]

How many hits should an enemy take to kill (medium difficulty)?

- 3-4[4]
- 5-6[5]
- 7-8[1]

How would you like the learning curve to be? (How quickly it gets hard from the start of the game)

- Steep [7]
- Medium [3]

What kind of music would you like?

- Suspense [2]
- 8-bit [2]
- Adventure [5]
- Folklore [1]

Analysis of the second questionnaire

The questionnaire gave me insight into what my clients wanted in terms of design and gameplay. Most of them tended to a similar design, so I will be able to focus more on one specific set of characteristics. In the case of the preferred features such as "hardness" in terms of the enemies I will try to develop a solution that satisfies the clients yet is still within my abilities.

In regards to the previous list of specification requirements and features all my clients were satisfied, so I will use it along with the results of the questionnaire to aid me in creating the second requirement specification list.

Specification Requirements

- Game window will be full screen.
- The inputs will be through the keyboard and mouse.
- User inputs control the actions of the character.
- The character can move in four directions: up, down, left and right.
- The character will be able to attack.
- The character view is oblique.
- The camera follows the player.
- The map will be a closed area where the player can move in.
- There are different starting characters with different base attack and health points.
- When the character collides with enemy, the player's health reduces.
- When the character hits the enemy with a weapon, the enemy's health reduces.
- When an enemy is defeated, points are awarded to the player.
- The enemy will follow the player until the player is dead.
- There will be different enemy types.
- The game progresses in waves: a number of enemies will appear after they are all defeated another wave of increased number of enemies will appear.
- Game ends when character's health is depleted.
- Game over will display the user total score.
- Points are stored under highscore if the score was higher at the end of the game.

I will use this list of requirements as a pseudo success criteria, meaning that if I accomplish these I would consider my solution successful since it meets my clients need.

Software & Hardware Requirements

The system will be written in C# in the Unity IDE. I obtained the basic hardware and software requirements of the system from the Unity website; I have also taken into account the hard disk space that it would take up 100Mb.

Minimum Requirements

Software Requirement	Justification
Windows XP or above	Operating system required to run Unity

Hardware Requirements	Justification
Intel Pentium 4 (2.8GHz) and above or Athlon 64 X2 3000+(2.0GHz) and above	CPU that supports SSE2 instruction set
GeForce 6800 or RADEON X800 and above	Graphics card that supports: DX9 (shader model 3.0) or DX11 with feature level 9.3 capabilities

User requirements

A tutorial will be created so all clients will be able to learn and play the game. Also, a help page will be created in order to answer any queries the client might have in game.

I showed all the requirements to my clients and asked for a signature if they were happy with them as a



Handwritten signatures of clients, including Simeon, Jones, and Svennhyzen, are shown below their names.

Simeon Jones Svennhyzen

validation method to make sure I produced a solution that fulfilled my clients requirements and would be program they would use.

Section 2 - Design

Following the list of the produced specification requirements by the interviews I will now produce a list of design objectives. There are many kinds of games and genres but talking with my clients we have settled with a 2D top-down roguelike arena design. I will develop the game using the programming language C# in the Unity game IDE. The player will control a character in the arena, with a wave system (the enemies spawn in increasing numbers after each wave is defeated). When they are defeated. Defeating monsters will award points, which will go towards a score system, the more enemies you are able to defeat the more points will be awarded. When the life counter goes down to 0 it will show a screen saying that it's game over, at this point the score will be restored and then redirected to the main menu.

To make all of this work I will have to use object-oriented programming, this is so I am able to give different properties so objects such as the player, enemies, projectiles and weapons, are able to interact with each other how they should. Each object will have different properties, which is what will differentiate for example a wall from an enemy. Additionally, all inputs will be from the keyboard and mouse.

Design objectives

I. Aesthetics

General features:

1. Window will be full screen
2. Black and white text
3. Black background
4. Grey buttons

Specific features:

- Main Menu
 1. Title of the game
 2. Black background
 3. Black text
- Setting
 1. Simple layout of settings so it is intuitive
 2. Black background
 3. Grey buttons
- Gamescreen
 1. Ground tiles background
 2. Entity tiles such as player and enemies
- Pause
 1. Slight dark tint overlaid on the game screen
 2. Pause text
- Game over
 1. N/A
- Help
 1. Help section
 2. Controls section
- HighScores
 1. Organised list of the high scores

II. Input

- Main Menu

1. Button to go to play
2. Button to go to settings

- 3. Button to go to help
- 4. Button to exit the game
- Setting
 - 1. Button to turn music on or off
- Gamescreen
 - 1. Button to pause game
 - 2. The program will listen to keyboard inputs to control the player
- Pause
 - 1. Button to resume game
 - 2. Button to access settings
- Game over
 - 1. Button to continue to main menu
- Help
 - 1. Button to go back to the main menu

III. Processing

- Main Menu
 - 1. When any button is pressed, it will take you to that screen
- Setting
 - 1. If the button for music is clicked it will turn music on or off
- Gamescreen
 - 1. Load all entities
 - 2. Load images and sprites
 - 3. Display the player and enemies
 - 4. Spawn enemies in waves
 - 5. Detect collisions between entities (walls, enemies, player)
 - 6. Detecting keyboard input and adjusting the actions of the player accordingly
 - 7. Checking that the player's health is always more than 0 or it will be game over
 - 8. The enemies will move towards the player
- Pause
 - 1. Stopping all enemies and players
- Game over
 - 1. Terminating the game screen
- Help
 - 1. List all controls and help
- HighScores
 - 2. Retrieving the table with the highest scores

IV. Output

Text boxes will be in place in all screens to display relevant text

Background will be black

Background music will be in place

- Main Menu
 - 1. Image to display the name of the game
- Setting
 - 1. Image of music icon
- Gamescreen

1. Images for textured background
 2. Images for player and enemies
 3. Images for health status
 4. Lives score counter
- Pause
 - 1. "Paused" text
 - Game over
 - 1. "Game over" text
 - Help
 - 1. List all controls
 - 2. List any help about gameplay

Graphic Design



First iteration

I made the above character using my imagination and knight/Viking graphics. Even though I was happy with the dimensions of the character and the design follows the retro theme I felt like detail was lacking since it is such a small resolution. Before continuing with this design and giving it colour I wanted to make sure the clients were happy with it. So, I asked them to comment on the design. Their response:

"It missing defined features that define a knight"

"It looks too cute, it doesn't look like a fighter"

"It should have bigger horns, and a more defined body"

"Could you make it with higher resolution?"

These are some comments that stood out to me and made me understand what my clients were looking for. So in the next iteration I will use a slightly higher resolution so I can accommodate for more defined body and I will also give it a less cute and more like a knight or fighter.

Second Iteration

Following the comments on my first design, I made a set of 3 designs for each different body part to see if which design was desired by my clients. Some iterations of each body part had more obvious difference than others, with this I wanted to find out how slight differences might affect the people decision when choosing a specific design.



I was expecting to see choices to be spread out and not be one sided, but after I queried my clients about which one they like more for each body part the winners where rather apparent:

Horns	3	6	1
Head	3	5	2
Arms	2	6	2
Body	3	4	3
Feet	2	0	8

- ✓ Horn n°1
- ✓ Head n°2
- ✓ Arms n°2
- ✓ Body n°2
- ✓ Feet n°3

These results make my job of designing the character easier, since I can focus on one design and improve on it as supposed to make 5 designs that are not as good. I understand that some of my clients might not prefer some parts of the design and that is ok since is not finished and I can further develop it, above all I will always check with my clients before making anything final.

Final Iteration

I assembled the highest voted parts to form the final basic design. I decided to give the horns some colour as they lacked contrast from the helmet. I added some miscellaneous parts like shoulder pads, tabard and gloves; gave them some colour inspired in Templars. Furthermore, I improved the feet as they looked too small in comparison to the rest of the body. After that I generated a back, right and left side views of the character which will be used to animate movement.



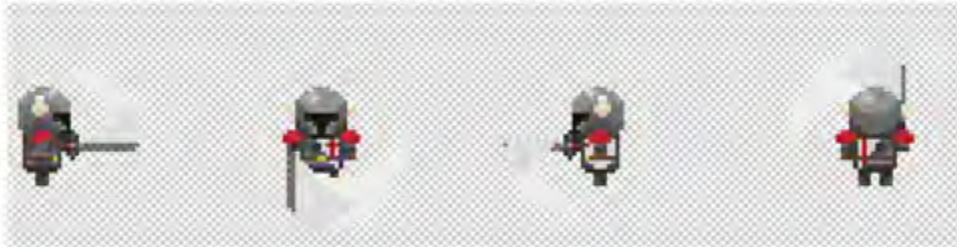
After having the basic static model of the character, I needed to give it a walking motion, so I developed a sprite sheet that contained the frames necessary to animate a walking motion in each direction since each direction has to be animated separately.



After the Sprites were finished, I realized that I needed to give the player the weapon which is going to fight with so I gave it a sword for each perspective.

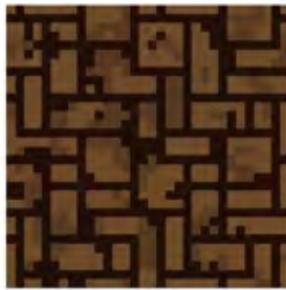


The attack animation will be circular meaning that when the user attacks the player will spin on itself to deal damage.

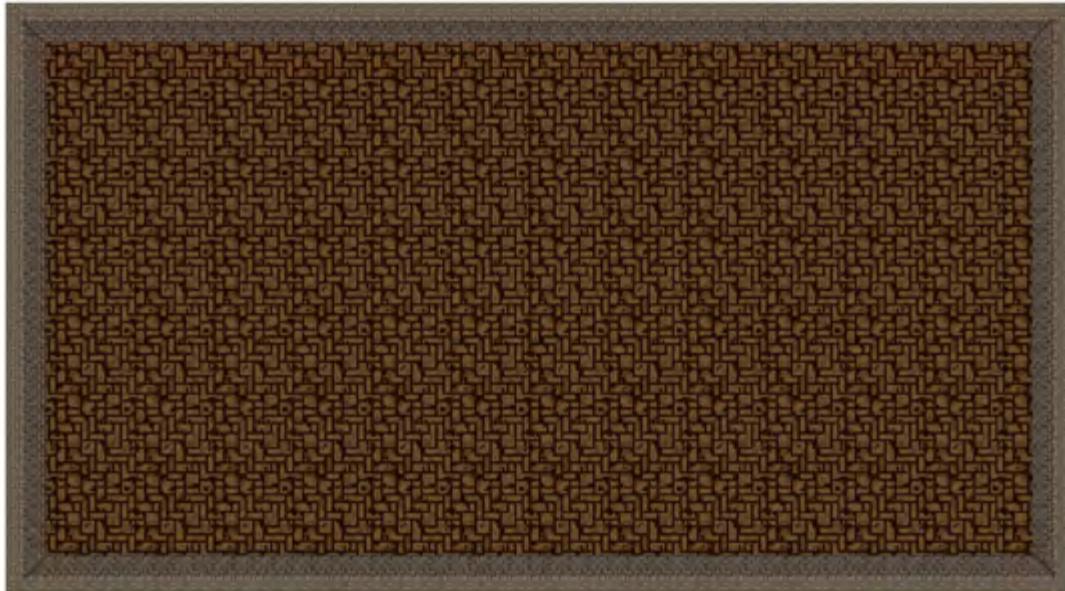


When creating the enemy I wanted to make it clear that it was the enemy. In order to achieve this I designed the enemy with a zombie/ monster design which is more associated with an enemy character. Some clearly defined yellow teeth and teared clothes also help with this.





The map was designed by creating a tile and repeating the tile to create a large area. When creating the tile, I took into account edges so when the right side was next to the left side it would look natural like a puzzle. After I created the walls and the corners for the map following the style of the floor with shading for depth.



Finished map:

[Client reviewed design](#)

When all the graphics were finished, I asked my clients for approval and they were all very satisfied with these.

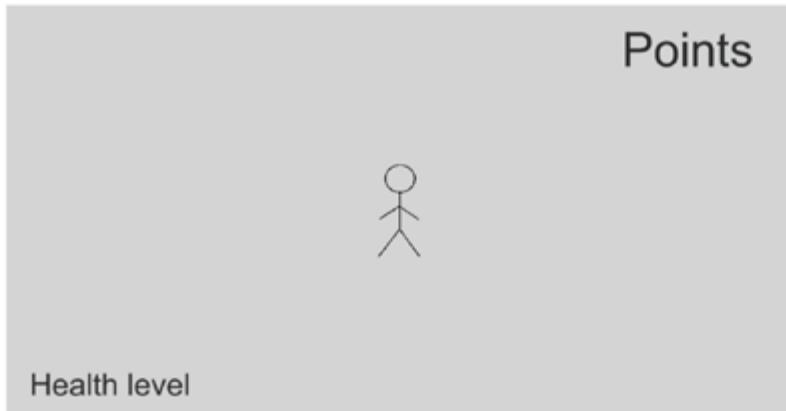
Interface Design

Main Menu scene



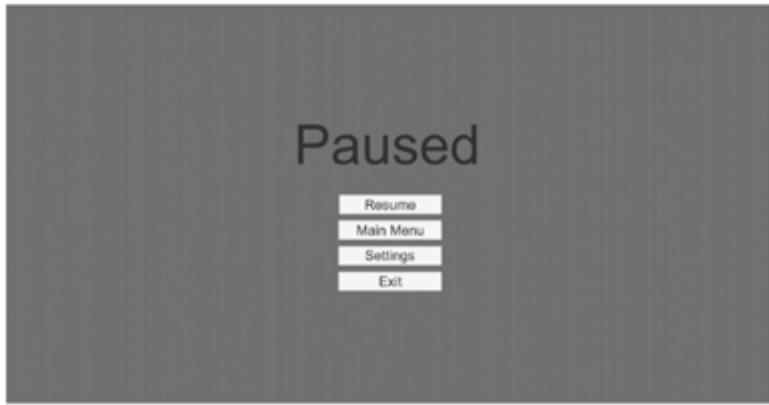
The main menu scene is going to be simple, intuitive and straight forward. This is a basic layout I created in order to fulfil these main points. The title of the game, image or logo will go above the different buttons (options) such as: play, settings and exit. The background is going to be black, but an image or in game animation would also be a possibility. The positioning of the buttons can be moved to the left or right if an in-game animation or image is to be put in place so space is used more efficiently and also more pleasing to the eyes.





The game scene will display the player, enemies, map and GUI (score counter and health level). As the player moves the camera will also move as to follow the player, furthermore the GUI should be stationary with respect to the camera so they are always visible. The background will be the map itself.

Paused Menu Scene



The paused menu will only be accessed while in the "Play" screen. The screen will have a text box telling the player that the game is paused; and an array of buttons giving the user different options from the pause menu like: resume, main menu, settings and exit.

Game Over Scene



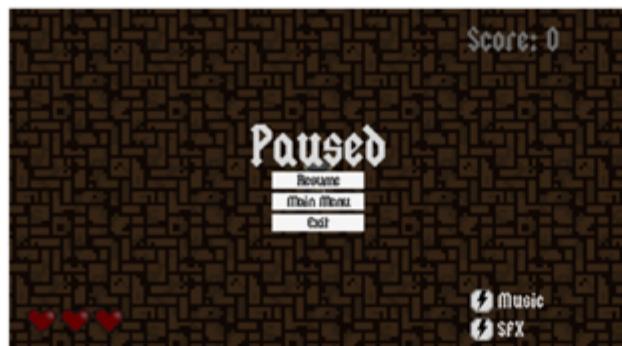
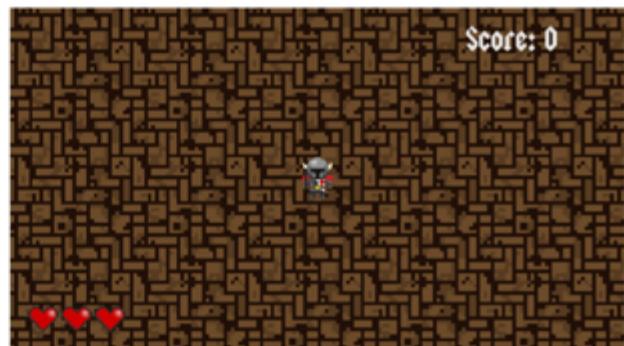
The game over scene will be displayed after the player's health reaches 0. This screen will display the player's end score which is saved under "YourScore" in the game screen and is called in game over to be retrieved. It will also retrieve the public static variable "Highscore" which hold the highest score ever recorded in the game.

Final Designs



The final design was very similar in structure to the original layout. The settings button and screen was removed since the only settings are music the settings scene would take unnecessary space in the memory so I decided to ditch it and go for a simpler approach to just have it accessible from the main menu. I obtained a commercial font "Squealer" and I have used it throughout my program (refer to licenses). I used the floor graphic to improve the title.

The game screen pretty much hasn't changed. The player on the middle, the score counter in the upper right hand corner and the health level on the bottom left hand corner being displayed with hearts 6 points in total each represented by half a heart.



After analysing the possibilities, the "Paused" scene, it seemed a better idea to have the paused screen as an overlay to the game scene with a pseudo transparent canvas to obscure the background. Also, I

did the same I did in the main menu and made the settings accessible from the pause menu, and it will also have SFX from the players so I also added that setting in the pause menu.

The game over scene is exactly the same with no other changes apart from slight changes to font and font size. After the user presses the enter key they will be taken to the main menu, where they can keep playing the game.

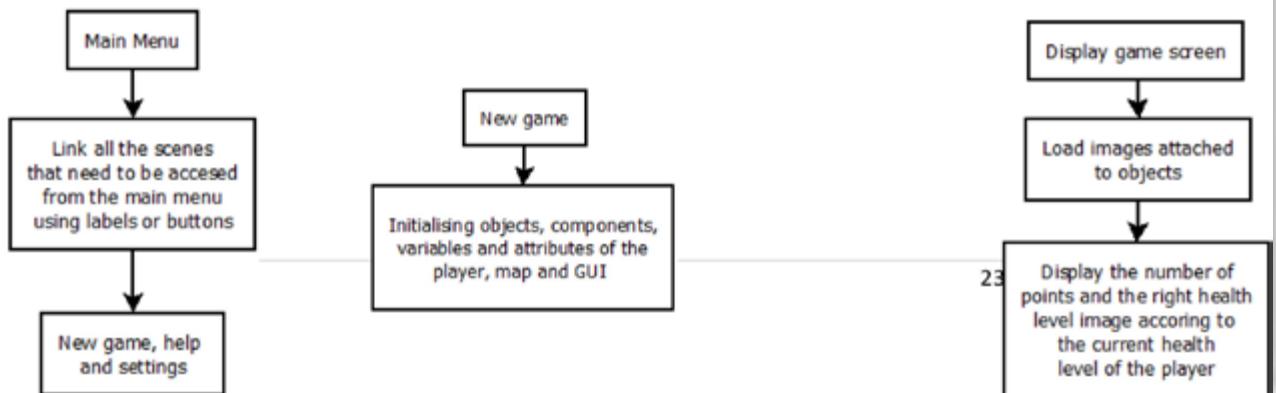


Client reviewed final designs

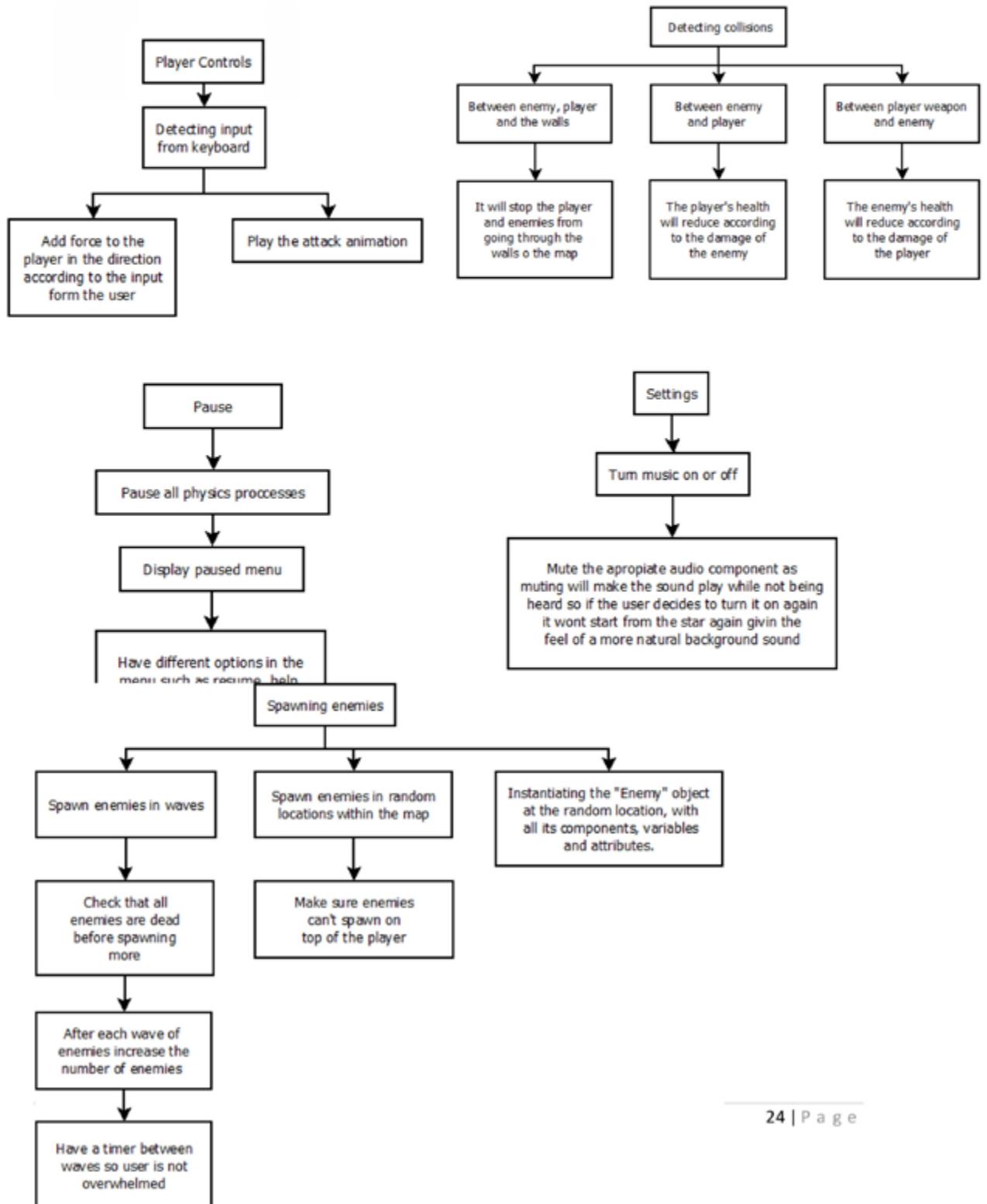


I showed the designs for review and they all agreed.

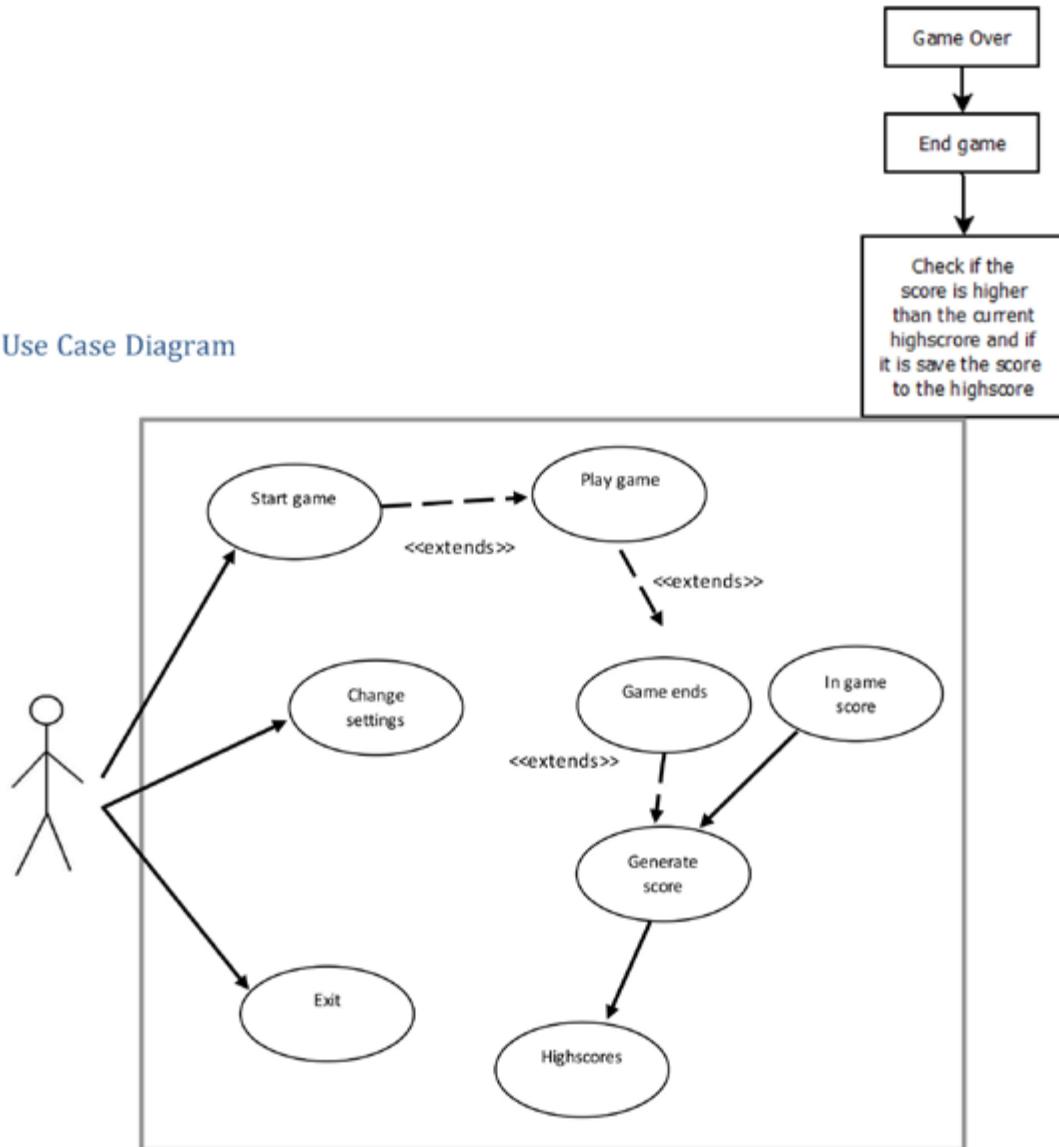
Modular Design Diagram



23



Use Case Diagram



Data Structure Design

Objects:

Player

Variable	Type	Identifier	Description
Speed acceleration	Float	speed	Used as a multiplier to add a force to the player to enable movement
Speed maximum	Float	speedcap	To make sure the player doesn't go over the speed maximum
Idlespeed	Float	idlespeed	To check if the player is going at idle speed
Player health	Integer	playerhealth	Stores the current player's health
Maximum player health	Integer	maxhealth	To give the player the max health at the start of the game
Current speed x	Float	curspeedx	Stores the current speed of the x component
Current speed y	Float	curspeedy	Stores the current speed of the y component
Damage	Integer	playerdamage	stores the damage the player will deal to enemies
Attack clock timer	Float	attacktimer	will be used to create a simple timer to control the attack speed
Attack cool down	Float	attackcd	stores the value between attacks of the player, to prevent perpetual attacking while the "attack" key is pressed
Attacking	Boolean	attacking	stores if the player is attacking to prevent starting the attack method again before it has finished

Apart from the script components it will also have various component attached to itself that are:

- Rigidbody2D
- Sprite renderer
- Animator
- Colliders
- Audio Source

These are subject to change and tweak during implementation, these are the basic components needed for development.

Enemy

Variable	Type	Identifier	Description
Enemy health	Float	enemyhealth	Stores the current enemy's health
Max health	Float	maxhealth	To give the enemy max health when is instantiated
Speed	Float	speed	used as a multiplier to enable movement
Score value	Integer	scoreval	Stores the value used to award points to the player
Damage	Integer	enemymdamage	Stores the damage the enemy will deal to the player

Attack clock timer	Float	attacktimer	will be used to create a simple timer to control the attack speed
Attack cool down	Float	attackcd	stores the value between attacks of the enemy, to prevent the enemy from dealing damage continuously

Apart from the script components it will also have various component attached to itself that are:

- Rigidbody2D
- Sprite renderer
- Animator
- Colliders

These are subject to change and tweak during implementation, these are the basic components needed for development.

Camera

Variable	Type	Identifier	Description
camera movement multiplier for x	Float	smoothmultx	value used to delay the camera movement
camera movement multiplier for y	Float	smoothmulty	value used to delay the camera movement

Apart from the script components it will also have various component attached to itself that are:

- Camera
- Flare Layer
- GUI Layer
- Audio Listener

These are subject to change and tweak during implementation, these are the basic components needed for development.

Game Controller

Variable	Type	Identifier	Description

Enemy number	Integer	enemynum	Stores the number of enemies to be spawned
Current enemy number	Integer	curenemynum	Stores the current number of "Enemy" objects in the scene
Initial wave delay	Float	iinitialwait	Used to delay the time the first wave of enemies is spawned
delay between waves	Float	wavewait	Used to delay the time between waves so the player has some time to catch up

Various variables to be used by the program:

Variable	Type	Identifier	Description
score	Integer	score	Stores the current score of the player, and is updated as the player kills enemies.
score	Integer	highscore	Stores the highest score ever achieved
Paused	Boolean	paused	Stores the value if the game is paused

Algorithms

Player

Movement

```

IF (input == up) THEN
    player movement = positive y
    Animation = up
ENDIF
IF (input == down) THEN
    player movement = negative y
    Animation = down
ENDIF
IF (input == right) THEN
    player movement = positive x
    Animation = right
ENDIF
IF (input == left) THEN
    player movement = negative x
    Animation = left
ENDIF

```

This algorithm deals with the player movement. The players position is stored in an X, Y grid what this is doing is checking for the direction and adding a velocity in that direction. After that you want to set the animation to the direction of the key so the movement and animation correlate and make sense.

```

IF (input == space) and (attacking == false) THEN
    attacking = true
    attacktimer = attackcd
    Attackcollider.enabled(true)
ENDIF
IF (attacking) THEN
    IF (attacktimer > 0) THEN
        attacktimer = attacktimer - Timepassed (time passed since last frame)
    ELSE
        attacking = false
        Attackcollider.enabled(false)
    ENDIF
ENDIF
Animation = Attacking

```

Attacking

This algorithm deals with enabling and disabling the collider which is used for attack. It has a clock so the player can't attack consecutively.

Taking damage

```

public procedure damage (enemydamage)
    playerhealth -= enemydamage
endprocedure

```

Death

```

IF (playerhealth <= 0) THEN
    die ()
ENDIF

procedure die ()
    yourscore = score
    IF (yourscore > highscore) THEN
        highscore = yourscore
    ENDIF
    scene.gameover()
endprocedure

```

Enemy

```

oldpos = enemy.position.x
Enemy.position = vector2.MoveTowards(enemy.position, player.position, distancePerStep)
newpos = enemy.position.x
IF (oldpos > newpos) THEN
    enemy.scale= new vector3(-1,1,1)
ENDIF
IF (oldpos < newpos) THEN
    enemy.scale= new vector3(1,1,1)
ENDIF

```

Animation and Movement**Attack**

```

attacktimer += Timepassed (time passed since last frame)
IF (attacktimer >= attackcd and AttackCollider.enabled(false))
    attacktimer = 0
    AttackCollider.enabled(true)
ENDIF
IF (attacktimer >= attackcd and AttackCollider.enabled(true))
    attacktimer = 0
    AttackCollider.enabled(true)
ENDIF

```

Death

```

IF (enemyhealth <= 0) THEN
    score += scoreval
    Destroy(gameObject)
ENDIF

```

Taking damage

```

public procedure damage (playerdamage)
    enemyhealth -= playerdamage
endprocedure

```

```

float positionx = Mathf.SmoothDamp(camera.position.y,
player.transform.position.y, ref velocity.y, smoothmultx)
float positionx = Mathf.SmoothDamp( transform.position.x,
player.transform.position.x, ref velocity.x, smoothmultx)

camera.position = new Vector3 (positionx, positiony, camera.position.z)

```

Camera

```

procedure spawnwave()
    waitforseconds(initialwait)
    WHILE (true)
        curenemynum= Gameobjects.enemy
        IF (curenemynum == 0) THEN
            FOR (i = 0, i <= enemynum, i++)
                spawnPosition = new Vector3 (Random.Range (-
spawnValues.x, spawnValues.x), Random.Range (-
spawnValues.y, spawnValues.y), spawnvalues.z)

                spawnRotation = new Quaternion ()
                Instantiate (Enemy, spawnPosition, spawnRotation)
        ENDFOR
        enemynum++
        ENDIF
    waitforseconds(wavewait)
ENDWHILE
endprocedure

```

Game Controller

These algorithms are the minimum for the application to be functional and fulfil the specification requirements.

Testing data

I will be testing my program using tables to make sure all aspects of the program are functioning accordingly to the requirements.

Function/Method being tested	Scenario	Input (optional)	Expected results
Player movement	In game	up	player will move up, and play the walk up animation
		down	player will move down, and play the walk down animation
		left	player will move left, and play the walk left animation
		right	player will move right, and play the walk right animation

Player attack	In game	space	player will attack
Player damage(taken)	In game	-	players health will reduce according to the damage done by the enemy
Player death	In game	-	game will end
Enemy movement	In game	-	enemy will move towards the position of the player, is a animation will move respectively to the left or right in the direction of movement, and colliders will be flipped for each direction
Enemy attack	In game	-	enemy will attack every a set amount of time
Enemy damage(taken)	In game	-	enemy's health will be reduced by the player's damage
Enemy death	In game	-	points will be awarded to player and the enemy object will be destroyed
Camera movement	In game	-	camera will move towards the player, as to follow it. With the stationary point of the player in the middle
Game controller	In game	-	x enemy will be spawned after all are destroyed x+1 are spawned, Initialdelay of "initialwait" and delay between waves of "wavewait"

Scene	Action	Expected results
Main menu	Play button	load game scene
	Exit button	application closes
	Music toggle	music is muted/unmuted
Paused menu	Resume button	un-pauses the game
	Main menu button	load mainmenu scene
	Exit button	application closes
	Music toggle	music is muted/unmuted
	SFX toggle	SFX is muted/unmuted
Game over	press enter	load mainmenu scene

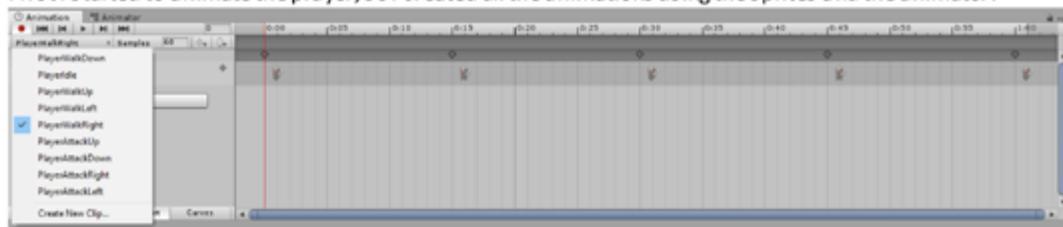
Section 3 – Developing the coded solution

Game Screen

Player

As I imported all graphics and sprites and font to the solution on unity to produce the interface, I can get to create the scripts for all the game object, because as for right now they are just objects with only images or text.

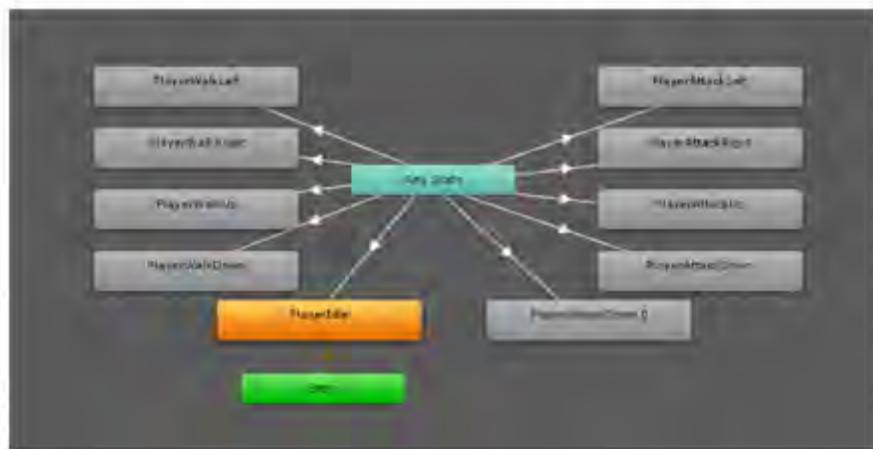
First I started to animate the player, so I created all the animations using the sprites and the animator.



I needed parameters that will be met when the player moves in a direction or attacks to play that animation. So I created the following booleans:



Then I created transitions between animations so each animation can go to the next animation.

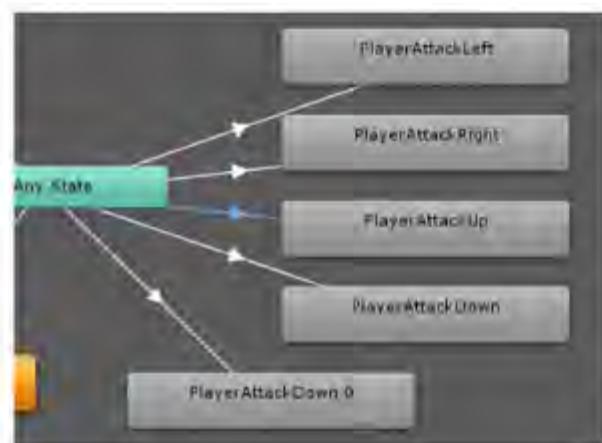
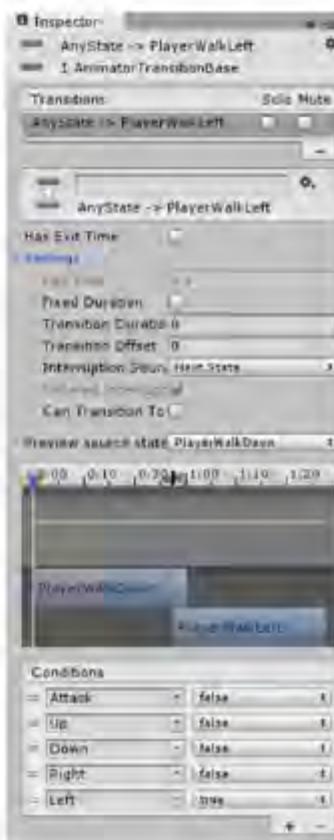


Then I attached the parameters/conditions for the transitions between animations to happen.



Walk example

This is the transition for PlayerWalkLeft, it can go from anystate meaning it can transition from any animation (also called states). For the settings, I unchecked fixed duration for continuous animation, transition should be instant so transition duration is 0 it can be interrupted by the next animation and unchecked can transition to self otherwise it would loop at the start of the animation and would never finish playing. For the left to be true and we want it all other directions to be false since its just walking to the left.



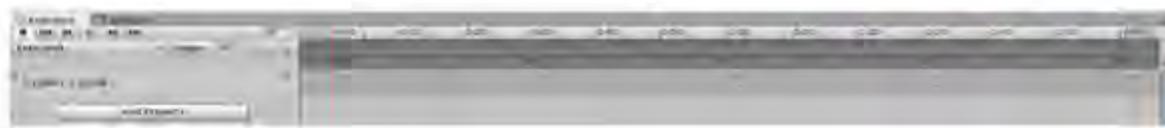
Attack example

For the attack animations there is different animations so they start and end in the direction the player was moving at the start of the animation. In the settings the only thing I have changed is that the interruption source is change to none so the animation can't be interrupted. Of course "Attack" has to be true for all attack animations, but I had to add a second PlayerAttackDown that has allbut "Attack" to false for when the player is not moving and attacks.

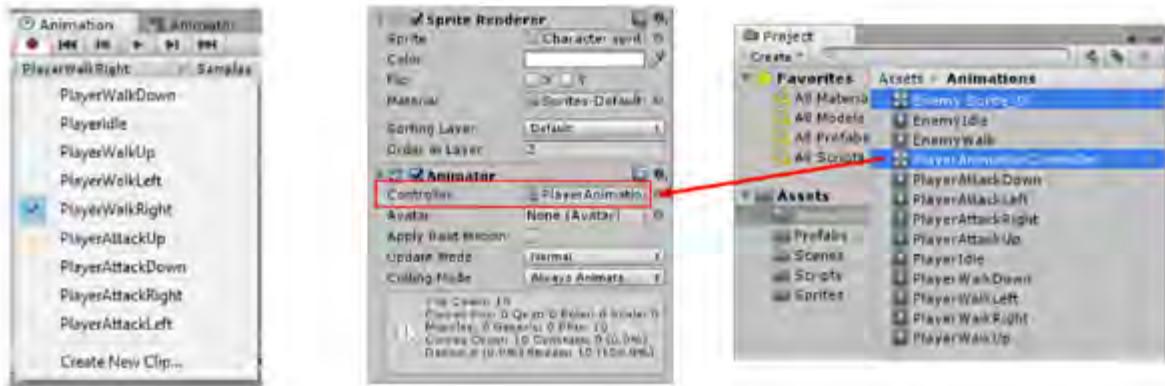


Furthermore PlayerIdle has all conditions to false.

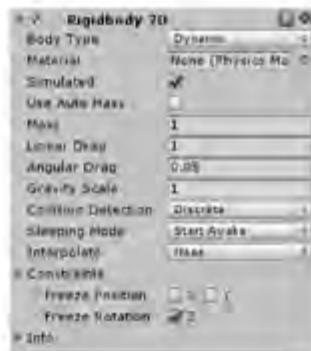
The enemy is always walking in game towards the player, and he deals damage in contact with the player so only one animation was necessary and it will always play when it's instantiated. The direction will be changed by the scale so I can also change the colliders scale when it changes direction.



The "Player" and "Enemy" object have to use these animations so a sprite renderer and animator components are necessary on both objects. When you create a new animation it also creates a controller for that animation and any new clips you create in that group of clips. The controller need to be referenced in the component of the object from the assets folder where you saved it.



For coding the movement scripts for the "Player" object, it needs a Rigidbody2D component to be able to use forces for movement. These are the settings I used, note I disabled gravity in the settings as this is top down so there is no need for it. Body type has to be dynamic so it's affected by other colliders and forces. Also the z rotation is frozen because we don't want our player to rotate anti-clockwise or clockwise.



Next, I'll make the Player scripts that will deal with movement attack and animation.

```

//When dealing with forces we have to use FixedUpdate() as it is designed for dealing with rigidbodies as you need to apply the force every fixed frame
void FixedUpdate()
{
    //Get the current absolute velocity for x and y components
    curSpeedX = Mathf.Abs(rbody2d.velocity.x);
    curSpeedY = Mathf.Abs(rbody2d.velocity.y);

    //reset the force applied in each frame
    float ForceX = 0f;
    float ForceY = 0f;

    //Checking for user input and setting the Force to the speed in the corresponding +ve or -ve direction for X and Y (4 iterations one for each direction)
    if (Input.GetKey("down")) {
        //Check if the current speed is less than the threshold and if so setting force to speed
        if (curSpeedY < SpeedCap) {
            ForceY = -speed;
        }
    }
    if (Input.GetKey("up")) {
        if (curSpeedY < SpeedCap){
            ForceY = speed;
        }
    }
    if (Input.GetKey("right"))
    {
        if (curSpeedX < SpeedCap){
            ForceX = speed;
        }
    }
    if (Input.GetKey("left"))
    {
        if (curSpeedX < SpeedCap){
            ForceX = -speed;
        }
    }
}
//Adding a force to the rigidbody with components ForceX and ForceY
rbody2d.AddForce(new Vector2(ForceX, ForceY));

```

Testing the code, the player moved in each direction according to the input of each arrow. The player is able to move in all four directions but the movement of the player is not connected to the animations yet but will connect them after the movement is completed.

I found that when using forces the player would keep moving even after the force was removed, like rolling a ball on a flat ground and that makes the player very hard to accurately control. I made this code to try to fix this issue:

Although this in theory should work, I realised that ForceX/Y is being reset every frame so realistically it would only run once, meaning that even though the force was applied the velocity didn't change much because the force was applied over a tiny time period (Using F=ma and $v = u + at$). Furthermore, I think this code is very inefficient and I don't really like it in general so I scratched it completely. I realised that if I was going to use a force to stop the player I was going to have to apply the force over a larger time and that didn't seem right.

```

if (rbody2d.velocity.magnitude > 0) {
    float StopX = -ForceX * StopMulti;
    float StopY = -ForceY * StopMulti;
    if (Input.GetKeyUp ("down")) {
        rbody2d.AddForce (new Vector2 (StopX, 0));
    }else if (Input.GetKeyUp ("up")) {
        rbody2d.AddForce (new Vector2 (StopX, 0));
    }else if (Input.GetKeyUp ("right")) {
        rbody2d.AddForce (new Vector2 (StopY, 0));
    }else if (Input.GetKeyUp ("left")) {
        rbody2d.AddForce (new Vector2 (StopY, 0));
    }
}

```

Because I can't really use force I'm going to modify the velocity to slow it down.

```

if (Input.GetKeyUp ("up")) {
    SlowdownY ();
}
if (Input.GetKeyUp ("down")) {
    SlowdownY ();
}
if (Input.GetKeyUp ("left")) {
    SlowdownX ();
}
if (Input.GetKeyUp ("right")) {
    SlowdownX ();
}

void SlowdownX(){
    float SlowX = rbody2d.velocity.x * 0.1f;
    float SameY = rbody2d.velocity.y;
    rbody2d.velocity = new Vector2(SlowX,SameY);
}

void SlowdownY(){
    float SlowY = rbody2d.velocity.y * 0.1f;
    float SameX = rbody2d.velocity.x;
    rbody2d.velocity = new Vector2(SameX,SlowY);
}

```

So, I
had to
implement
this:

What the code does is that when the arrow key is released it will call Slowdown[X or Y] () which will set the velocity of the player to one tenth the current speed, so the player can be properly controlled, instead of having the player completely stop on the direction they were going which might throw off the user a multiplier is used so the stopping motion is more natural and instinctive. With this structure, I can reuse each function for x and y. I first put the code in the FixedUpdate() as the functions dealt with physics but when testing the player movement, 5-10% of the time the slowdown function wouldn't work and leave the player velocity untouched. I found that putting the if statements in the Update() fixed it and I didn't get any more undesired motion, this is because I'm dealing with the velocity directly as supposed to forces.

This code since it doesn't deal with rigid bodies or physics the code will be called in the Update() function (which is called every frame, but some frames might take longer to process than others, therefore might be

```

//Checking which component x and y of the velocity of the player is greater
if (Mathf.Abs(rbody2d.velocity.y) > Mathf.Abs(rbody2d.velocity.x))
{
    //Checking if the velocity of the largest component (in this case y) is positive(up) or negative(down)
    if (rbody2d.velocity.y > 0.1f){
        //Setting the booleans to the appropriate value for the direction of the player to match the sprite
        anim.SetBool ("Up",true);
        anim.SetBool ("Down", false);
        anim.SetBool ("Right", false);
        anim.SetBool ("Left", false);

    }
    else if (rbody2d.velocity.y < -0.1f){
        anim.SetBool ("Up",false);
        anim.SetBool ("Down", true);
        anim.SetBool ("Right", false);
        anim.SetBool ("Left", false);
    }
    else if (Mathf.Abs(rbody2d.velocity.x) > Mathf.Abs(rbody2d.velocity.y)) {
        if (rbody2d.velocity.x > 0.1f) {
            anim.SetBool ("Up",false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", true);
            anim.SetBool ("Left", false);
        }
        else if (rbody2d.velocity.x < -0.1f) {
            anim.SetBool ("Up",false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", true);
        }
    }
}

```

delayed
when
being
called
again,
FixedU
pdate()
in the
other
hand
will
always
take
the
same
time).

Since we have dealt with the walking animations, now I need to make the player go into the idle animation

```
//Checking if the absolute or modulus of the magnitude(sqrt[(x^2)+(y^2)]) is less than the idle speed
if (Mathf.Abs (rbody2d.velocity.magnitude) < idleSpeed) {
    anim.SetBool ("Up", false);
    anim.SetBool ("Down", false);
    anim.SetBool ("Right", false);
    anim.SetBool ("Left", false);
}
```

when the player is doing under certain speed; this code is also called in Update().

I could have written the algorithms myself but using these already available functions make the code shorter and simpler, which takes less space in memory, saves time in coding and also makes the code more suitable for other developer to read the code and follow it.

The death of the player is quite straight forward to implement:

```
//Checking if the player's health is equal or less than 0
if (PlayerHealth <= 0) {
    Die ();
}
```

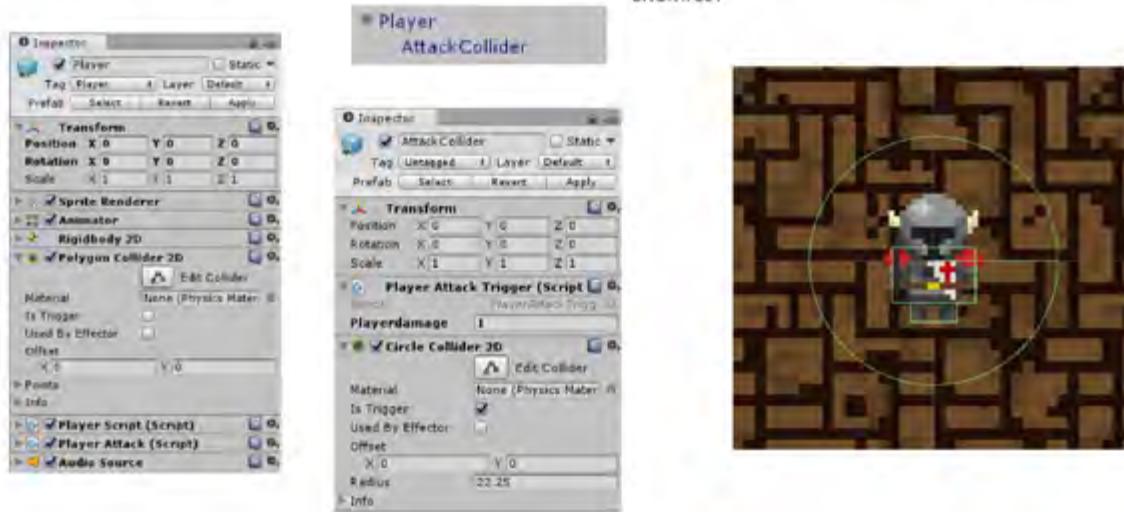
Currently I don't have any other scenes so Die() is empty once I have a score system and game over scenes I'll add it to Die().

For the player to take damage this is the code I used:

```
public void Damage(int Enemydamage){
    PlayerHealth -= Enemydamage;
}
```

This method has to be public since it will be called from the collider of the enemy when the collider of the player enters the trigger collider of the enemy. I will use the same structure to deal damage to either party, both enemy and player damage functions are the same the only difference is the change of variable names from Enemydamage → Playerdamage and PlayerHealth → EnemyHealth.

To deal damage to enemies the player is going to have a child object with a collider with the trigger property which will be attached to it. The player needs to have the "Player" tag and the same for the enemy with the "Enemy" tag. Furthermore, I gave the player object a collider to collide with walls and take damage from enemies.



This is the script that will detect the enemy entering the collider (is trigger). This is attached to the

```
public class PlayerAttackTrigger : MonoBehaviour {
    public int Playerdamage = 1;

    //This function is called when there is a collision between its collider and another collider and those colliders share the same tag
    void OnTriggerEnter2D(Collider2D col)
    {
        //CHECK IF THE THE COLLIDER THAT IS COLLIDING WITH IS NOT A TRIGGER COLLIDER AND MAKES SURE ITS A ENEMY OBJECT
        if (col.isTrigger == false && col.gameObject.tag == "Enemy") {
            //Send a signal to the parent of the collider in this case sends the function "Damage" and the value of the player
            col.SendMessageUpwards("Damage", Playerdamage);
        }
    }
}
```

AttackCollider object. This is where the damage of the player is stored.

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

public class PlayerAttack : MonoBehaviour {

    private bool attacking = false;
    private float attackcooldown = 0.4f;
    private float attacktimer = 0f;
    public Collider2D AttackCollider;
    public Animator anim;
    public AudioClip swing;
    public AudioSource sound;
    public Toggle SFXtoggle;

    void Start () {
        //Saves the components under variables so I can use them later
        anim = gameObject.GetComponent<Animator> ();
        sound = gameObject.GetComponent<AudioSource> ();
        AttackCollider.enabled = false;
    }
    void Update () {
        //Check if space is pressed and the player is not attacking
        if(Input.GetKeyDown(KeyCode.Space) && !attacking){
            attacking = true;
            attacktimer = attackcooldown;
            AttackCollider.enabled = true;
            sound.PlayOneShot (swing, 0.6f);
        }
        //Timer for the collider to be disabled after 0.4 sec
        if(attacking){
            if (attacktimer > 0) {
                attacktimer -= Time.deltaTime;
            } else {
                attacking = false;
                AttackCollider.enabled = false;
            }
        }
        // To make the animation play while is attacking
        anim.SetBool ("Attack", attacking);
    }
    //Function used by the pause settings
    public void SetSFX(){
}
```



This is the timer and activator of the AttackCollider game object. This script is attached to the Player object. The script checks if the space key is pressed and the player is not attacking already, attacking becomes true, attack timer is set to attackcd, the collider is enabled and the sound of the sword is played at 60% volume.

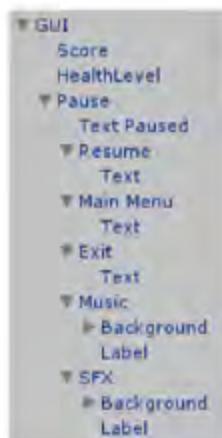
Then each frame is checked if attacking is true, if so, if the attack timer is greater than zero the attack timer will be reduced by the time passed from the last frame. When attack timer is less than zero attacking is set to false and the collider is disabled. Then the Attack variable is set so the animation will play when the player is attacking.

I added the public SFX enable/disable function that will be used in the pause menu here since the SFX sound are played from the player. It is a simple mute/unmute. I need to be using UnityEngine.UI to access the toggle class.

The function work as intended and it activates the collider for 0.4 seconds and then it disables it. The attack animations start and end in the direction the player is moving. When I have the enemy object, ready I will test the ability to deal damage to the enemy.

GUI

The GUI will contain the score counter, health level and pause menu.



```
public class YourScore : MonoBehaviour {
    private Text text;
    void Start () {
        int score = PlayerPrefs.GetInt ("Score", 0);
        text = GetComponent<Text> ();
        text.text = "Score: " + score;
    }
}
```

The score will be outputted from the text component of "Score" and it will increase whenever an "Enemy" is destroyed by the amount the enemy is worth. The script is attached to the Score object,

I want the health level to be simple something that everyone will understand so the health overlay will be displayed by an array of images that change depending on the health. By referencing the player we can access the PlayerHealth variable. The player has 6 health and the array has 7 images 6 plus an empty one. I came into some trouble in which I had the sprites in descending order and it was slightly confusing in the arrays since the numbers were going

```
public Sprite[] Hearts;
public Image HealthLevel;
private PlayerScript player;
void Start () {
    player = GameObject.FindGameObjectWithTag ("Player").GetComponent<PlayerScript> ();
    HealthLevel.sprite = player.PlayerHealth;
}
```



backwards so I decided that it made more sense for the sprites to be in the same order of the array so they are both increasing. So, I flipped the sprites and these are the results.

The pause menu is going to be a canvas overlay inside the GUI canvas which contains the score and the health

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

public class PauseMenu : MonoBehaviour {

    public GameObject Pause;
    public bool paused = false;

    void Start(){
        Pause.SetActive(false);
    }

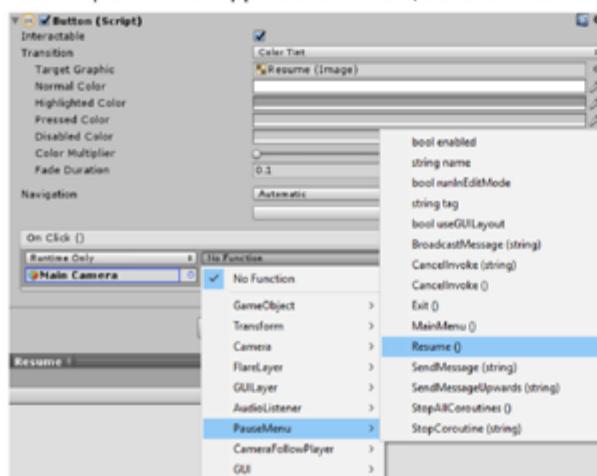
    void Update(){
        if (Input.GetButtonDown ("Pause")) {
            paused = !paused;
        }
        //Check if paused is true, then enables the pause menu canvas, and set the time scale to 0(stopped)
        if (paused) {
            Pause.SetActive (true);
            Time.timeScale = 0;
        }
        //Check if paused is false, then disables the pause menu canvas, and set the time scale to 1(normal speed)
        if (!paused) {
            Pause.SetActive (false);
            Time.timeScale = 1;
        }
    }
    //Function for the resume button, it sets the bool pause to false
    public void Resume(){
        paused = false;
    }
    //Function for the main menu button, it will Load the scene 0 the scenes are set up on the builder
    public void MainMenu(){
        SceneManager.LoadScene (0);
    }
    //Function for the exit button, it will close the application
    public void Exit(){
        Application.Quit ();
    }
}
```

level.

Firstly, will be using UnityEngine.SceneManagement to manipulate the scenes since we want to access the main menu from the pause menu. When the game starts, it starts disabled. If the "Pause" button is pressed it will set pause to the opposite of its value; I set the "Pause" button to "P". When the game is paused all frame

rate dependant functions are paused, so effectively all physics.

The GUI script and the Pause menu are components of the main camera since it's never enabled or disabled. The YourScore script is attached to the Score object since it need to get the text component. To connect the functions to the buttons and toggles I need



to reference the method inside the script in the button component, and do the same with the Music/SFX toggles. I placed the Music component on the Game controller so I can organize and not have each single component in different objects.

Camera

Function for the camera to follow the player.

```
public class CameraFollowPlayer : MonoBehaviour
{
    private Vector2 velocity;
    public float smoothTimeY;
    public float smoothTimeX;
    public GameObject player;

    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    void FixedUpdate ()
    {
        float positionY = Mathf.SmoothDamp(transform.position.y, player.transform.position.y, ref velocity.y, smoothTimeY);
        float positionX = Mathf.SmoothDamp(transform.position.x, player.transform.position.x, ref velocity.x, smoothTimeX);

        transform.position = new Vector3(positionX, positionY, transform.position.z);
    }
}
```

This code makes the camera follow the player and also it puts a delay between the movement of the player and the movement of the camera. The script is attached to the Camera object. Testing the functions, I found that the optimal delay was a multiplier of 0.08 as more delay would leave the player too off centre, and less would make the camera too snappy and make the impression that the player and camera moved at the same time.



Enemy

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyScript : MonoBehaviour {

    public int EnemyHealth;
    public int MaxHealth = 5;
    public float speed = 3f;
    public float check;
    public float oldpos;
    public float newpos;
    public int ScoreValue = 100;

    private Transform player;

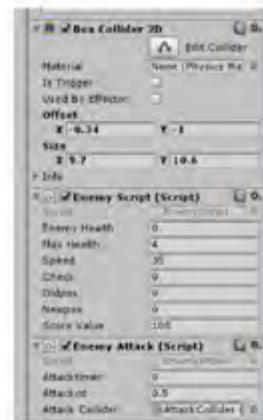
    void Start () {
        //Setting player equal to the transform of the object with the "Player" Tag which stores the position, rotation and scale of an object
        player = GameObject.FindGameObjectWithTag ("Player").transform;
        //Giving the enemy full health
        EnemyHealth = MaxHealth;
    }

    void Update () {
        oldpos = transform.position.x;
        //Moving towards the player at speed times the time passed since last frame
        transform.position = Vector2.MoveTowards(transform.position, player.position, speed * Time.deltaTime);
        newpos = transform.position.x;
        //Check if the enemy is moving towards the right or left and changing the scale by -1 of the object (including the sprite renderer and collider)
        if (oldpos > newpos) {
            transform.localScale = new Vector3 (-1, 1, 1);
        }
        if (oldpos < newpos) {
            transform.localScale = new Vector3 (1, 1, 1);
        }
        //If the enemy has a health of 0 or below the "enemy" object will be destroyed, and will award points equal to its scorevalue
        if (EnemyHealth <= 0) {
            ScoreScript.score += ScoreValue;
            Destroy (gameObject);
        }
    }

    //Dealing damage to the enemy
    public void Damage(int PlayerDamage){
        EnemyHealth -= PlayerDamage;
    }
}

```

The enemy movement is different to the player. I've decided to use variable `MoveTowards()` which deals with movement by translating from point A to point B in steps(`speed`). I also used `Time.deltaTime` to make it frame dependant. For the animation is the walking animation all the time, what I'm doing is changing the scale of the object itself so all components attached to the object are also flipped as I want to take colliders into account. After it dies the score of the enemy is added to the score and the game object is destroyed. Finally, I used the same structure for dealing damage to the enemy that I used with the player.



For attacking I'm going to use the same structure that I used for the player so the enemy is going to have a child with a collider that will turn on and off every 0.5 seconds. The code below does just that it is a simple on and off timer that is 0.5

seconds enabled and 0.5 seconds disabled.

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

public class EnemyAttack : MonoBehaviour {

    public float attacktimer = 0f;
    public float attackcd= 0.5f;
    public Collider2D AttackCollider;

    void Start(){
        AttackCollider.enabled = false;
    }
    void Update (){
        //add time for every frame
        attacktimer += Time.deltaTime;
        //Simple timer for enabling and disabling the AttackCollider
        if ((attacktimer >= attackcd) && !AttackCollider.enabled ) {
            attacktimer = 0;
            AttackCollider.enabled = true;
        }
        if ((attacktimer >= attackcd) && AttackCollider.enabled) {
            attacktimer = 0;
            AttackCollider.enabled= false;
        }
    }
}
```



The code worked wonderfully and turned the collider on and off with no problems whatsoever.

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

public class EnemyAttackTrigger : MonoBehaviour {

    public int Enemydamage = 1;

    //This function is called when there is a collision between colliders
    void OnTriggerEnter2D(Collider2D col)
    {
        //Checks if the the collider that is colliding with is not a trigger collider, and makes sure its a Enemy object
        if (col.isTrigger == false && col.CompareTag ("Player")) {
            col.SendMessageUpwards ("Damage", Enemydamage);
        }
    }
}
```

The code for checking collisions is the same one the player has just with Enemy damage.

Game Controller

```

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

public class GameController : MonoBehaviour {

    public GameObject Enemy;
    public Vector3 spawnValues;
    public int enemyNum;
    private int curenemyNum;
    private GameObject[] count;
    public float initialWait;
    public float waveWait;
    public GameObject Player;
    public AudioSource Music;
    public Toggle Musictoggle;

    void Start(){
        Player = GameObject.FindGameObjectWithTag("Player");
        StartCoroutine(SpawnWave ());
    }

    //since I want the code to wait between waves I need to use a coroutine in order to use the yield WaitForSeconds()
    IEnumerator SpawnWave(){
        //wait for initialWait seconds
        yield return new WaitForSeconds (initialWait);
        //loop that will always run until the game ends
        while(true){
            //Finds all objects in the scene with teh tag enemy and puts them in an array, then I use the lenght of the array
            count = GameObject.FindGameObjectsWithTag("Enemy");
            curenemyNum = count.Length;
            //checks if the number of enemies in the scene is 0
            if (curenemyNum == 0) {
                for (Int i = 0; i < enemyNum; i++) {
                    //Randomizes the spawn positions and an empty rotations since we dont need rotations but we still need to pass teh value to instantiate
                    Vector3 spawnPosition = new Vector3 (Random.Range (-spawnValues.x, spawnValues.x), Random.Range (-spawnValues.y, spawnValues.y), spawnValues.z);
                    Quaternion spawnRotation = new Quaternion ();
                    //Instantiates the enemy with a position and rotation
                    Instantiate (Enemy, spawnPosition, spawnRotation);
                }
                //Every time the for loop finishes another enemy will be added for the next wave
                enemyNum++;
            }
            //wait for waveWait seconds
            yield return new WaitForSeconds (waveWait);
        }
    }
}

```

The game controller script is going to store the spawn function and the script for the music toggle.

The musictoggle is the same as the SFX toggle just a simple change in variable names, and reference it in the toggle.

```

public void SetMusic(){
    if (!Musictoggle.isOn) {
        Music.mute = true;
    } else {
        Music.mute = false;
    }
}

```

Main menu

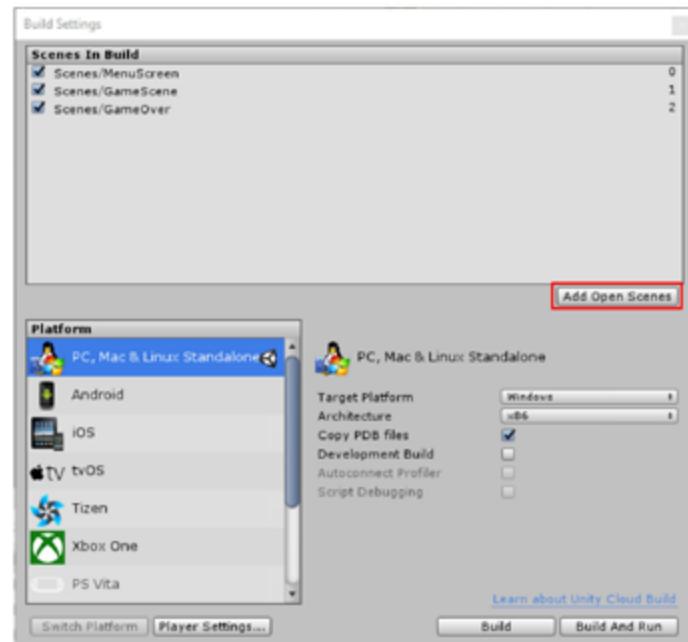
As I already have the design set I solely have to write the scripts for the buttons music and toggles. To the right is the design of the menu in the engine, when the game is running the background will be black instead of having a black picture which would increase the size of the program and is unnecessary. I created an empty scene which later will become the game over screen, then added said scene and the Game screen to the builder. Then I used the builder to make sure that each scene is in the order they should be in. The menu should be 0, since the 0 scene will be the first one loaded.



This is the hierarchy of the MenuScreen.

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

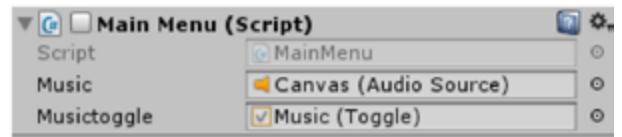
public class MainMenu : MonoBehaviour {
```



```
    public AudioSource Music;
    public Toggle Musictoggle;

    public void Play (){
        SceneManager.LoadScene (1);
    }
    public void Main(){
        SceneManager.LoadScene(0);
    }
    public void Exit(){
        Application.Quit ();
    }
    public void SetMusic(){
        if (!Musictoggle.isOn) {
            Music.mute = true;
        } else {
            Music.mute = false;
        }
    }
}
```

The code for the buttons is quite simple once you have referenced each scene in the builder, just have to be using UnityEngine.SceneManagement. I also have to use UnityEngine.UI to access Toggle. The music component and script which holds the functions is attached to the canvas which should be disabled in the inspector because we don't actually want it to run until we want it to.



Game Over

The game over screen will be a transition screen between the game and the main menu. It will show the score acquired by the user while the duration of the game. It will also show the highest score ever acquired by the user in that copy of the application.

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

public class GameOver : MonoBehaviour {

    void Start () {
    }
    void Update () {
        if (Input.GetKeyDown("return") || Input.GetKeyDown("enter")) {
            LoadMenu ();
        }
    }
    void LoadMenu(){
        SceneManager.LoadScene (0);
    }
}
```

The GameOver script just waits for the enter or return key to load the main menu



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

public class HighScore : MonoBehaviour {

    private Text text;

    void Start () {
        int highscore = PlayerPrefs.GetInt ("HighScore", 0);
        text = GetComponent<Text> ();
        text.text = "Highscore: " + highscore;
    }
}
```

The HighScore script retrieves the "HighScore" from the class PlayerPrefs which is used to store preferences between game sessions, and it updates the text to it.

```
void Die(){
    PlayerPrefs.SetInt ("Score", ScoreScript.score);
    int highscore = PlayerPrefs.GetInt ("HighScore", 0);
    if (ScoreScript.score > highscore) {
        PlayerPrefs.SetInt ("HighScore", ScoreScript.score);
    }
    SceneManager.LoadScene (2);
}
```

I added the code necessary record the users score for it to be accessed from the GameOver scene. I also checked that if the Score is greater than the current HighScore, the Score would become the value of HighScore. Then the game over scene is loaded.

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

public class YourScore : MonoBehaviour {

    private Text text;

    void Start () {
        int score = PlayerPrefs.GetInt ("Score", 0);
        text = GetComponent<Text> ();
        text.text = "Score: " + score;
    }
}
```

The YourScore script retrieves the "Score" from the class PlayerPrefs which is used to store preferences between game sessions, and it updates the text to it.

Validation

I have developed my program's structure so that ten validation process is inherent to the program itself. I have achieved this by limiting extremely the amount of input that the program reads. The only inputs will be the user interacting with the buttons and toggles of the GUI with the use of the mouse, the arrow keys,

spacebar for controlling the player, and lastly the letter "p" to toggle the pause. The structure I have put in place doesn't allow for the user to input erroneous data that would make the application crash or produce an error. Although this is good in this case where simple instructions are taking place, if more features were to be added in the future such as a login screen or a name for the player, some changes would have to be made to make sure the data entered in the program is accurate and valid.

Testing

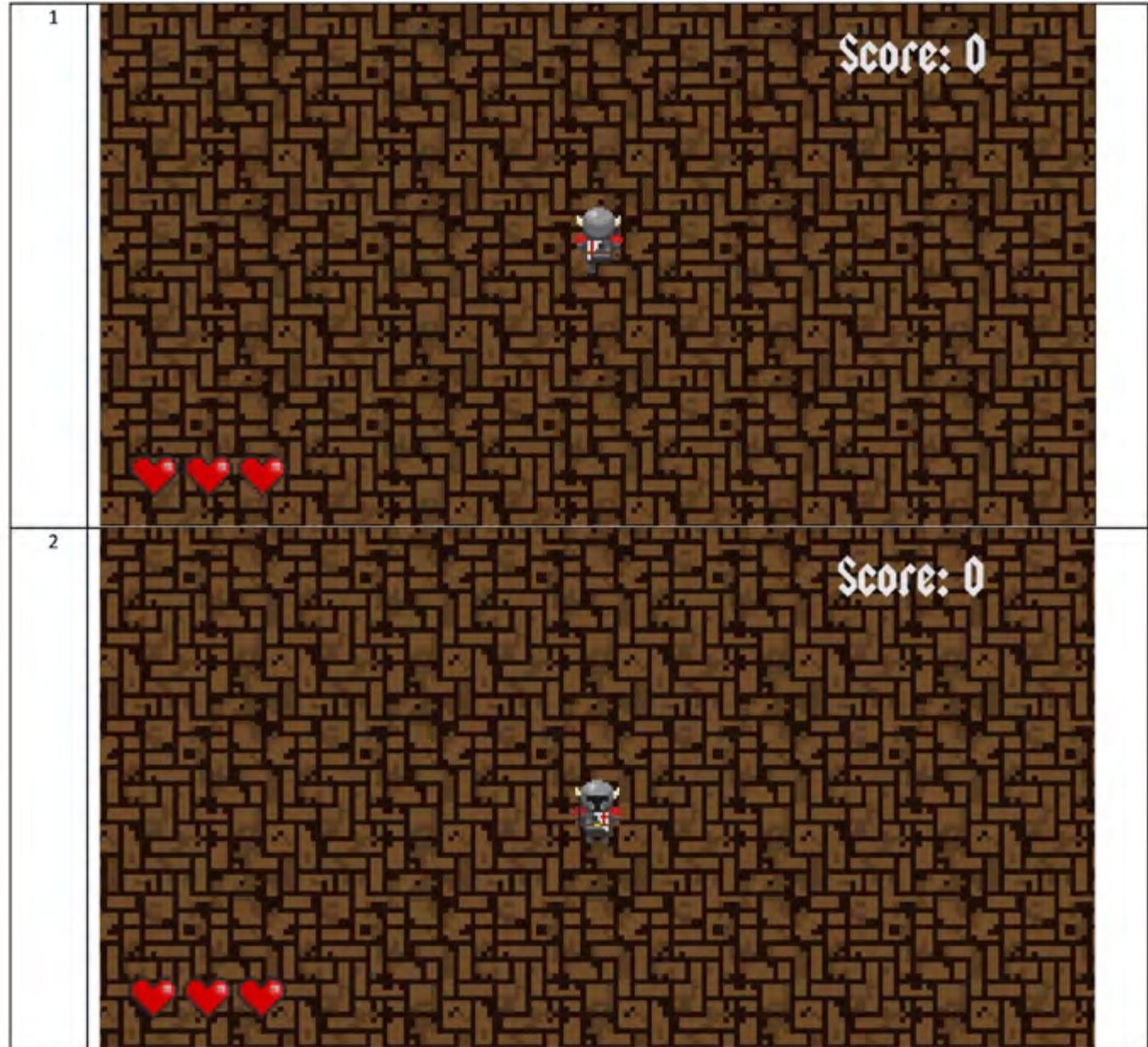
The program is now finished and now I will go into testing each scenario of the program. I will be writing all my tests in the table below. Note component lines will not be visible when the program is built, I test the program in the IDE so I can make sure everything is working in order.

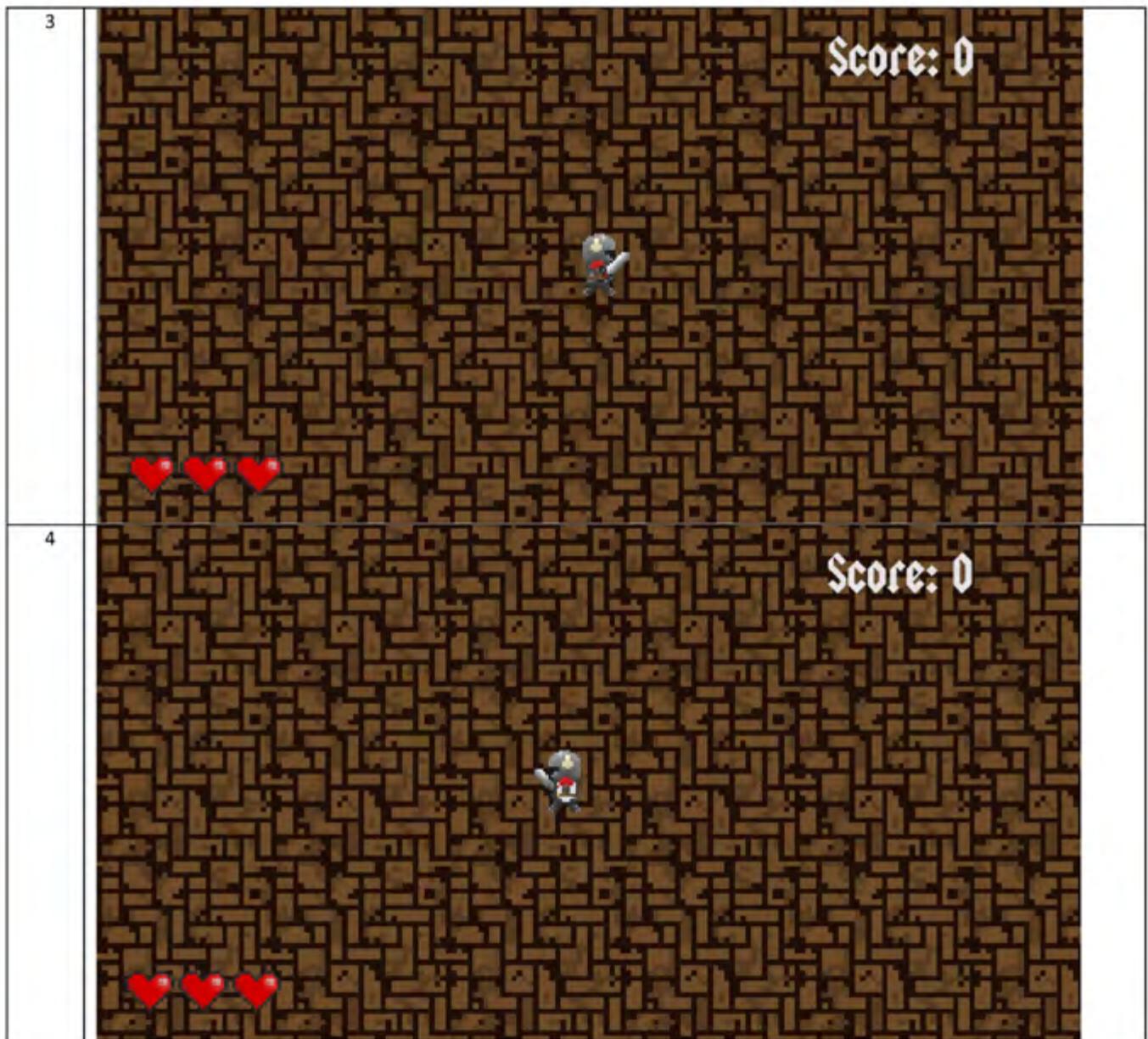
Test N°	Function	Input/condition	Expected result	Actual result	Screen References
1	Player movement (when pressed down)	UP ARROW	The player accelerates in the +y direction until it maxes out at velocity of 100	As expected	1
2		DONW ARROW	The player accelerates in the -y direction until it maxes out at velocity of 100	As expected	2
3		RIGHT ARROW	The player accelerates in the +x direction until it maxes out at velocity of 100	As expected	3
4		LEFT ARROW	The player accelerates in the -x direction until it maxes out at velocity of 100	As expected	4
5	Player movement (when key is released)	UP ARROW	The player reduces its velocity by factor 10	As expected	No visual aid is relevant
6		DONW ARROW	The player reduces its velocity by factor 10	As expected	No visual aid is relevant
7		RIGHT ARROW	The player reduces its velocity by factor 10	As expected	No visual aid is relevant
8		LEFT ARROW	The player reduces its velocity by factor 10	As expected	No visual aid is relevant
9	Player Walk animation(X and Y being the components of the player movement)	MOD(Y)> MOD(X) AND Y > 0.1f	"Up" set to true and all else to false, therefore transitioning to the PlayerWalkUp animation	As expected	1
10		MOD(Y)> MOD(X) AND Y < -0.1f	"Down" set to true and all else to false, therefore transitioning to the PlayerWalkDown animation	As expected	2
11		MOD(Y)< MOD(X) AND X > 0.1f	"Right" set to true and all else to false, therefore transitioning to the PlayerWalkRight animation	As expected	3
12		MOD(Y)< MOD(X) AND X < -0.1f	"Left" set to true and all else to false, therefore transitioning to the PlayerWalkLeft animation	As expected	4
13	Player idle Animation(X and Y being the components of the player movement)	Magnitude of the player is less than idle speed	"Up", "Down", "Right" and "Left" set to false, therefore transition to idle	As expected	No visual aid is relevant
14	Player attack AND PlayerAttack Animation	Space (While "Up" is true and all other directions are false)	The player will play the attack animation(up) and the AttackCollider is enabled	As expected	

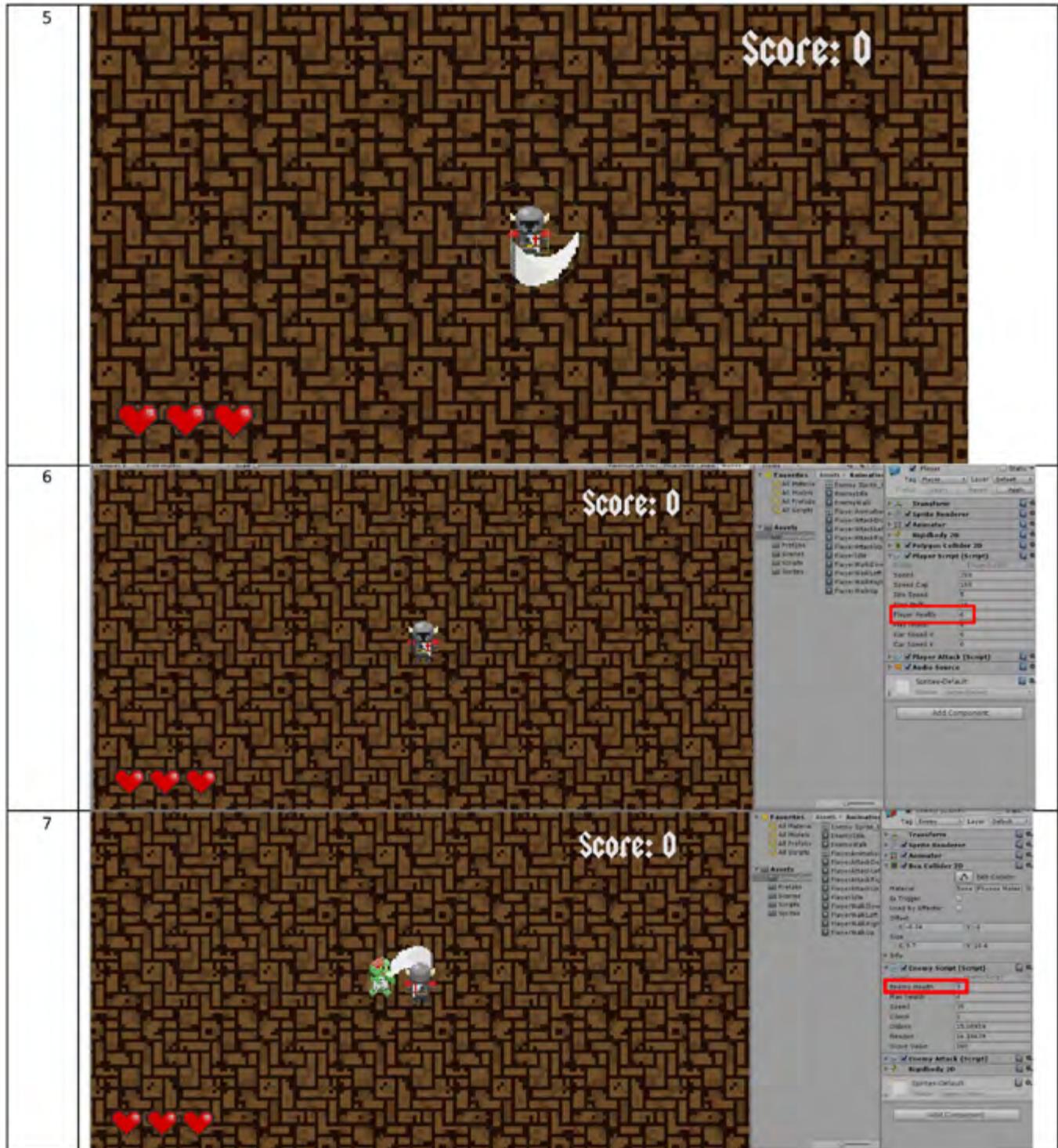
15		Space (While "Down" is true and all other directions are false)	The player will play the attack animation(down) and the AttackCollider is enabled	As expected	5 The four animations are the same apart from the start and end positions to follow movement.
16		Space (While "Right" is true and all other directions are false)	The player will play the attack animation(right) and the AttackCollider is enabled	As expected	
17		Space (While "Left" is true and all other directions are false)	The player will play the attack animation(left) and the AttackCollider is enabled	As expected	
18	Player starts with max health	-	The player will start the game with max health	As expected	6
19	Player [dealt] damage and Enemy damage [taken]	When an enemy enters the AttackCollider	The enemy's health will reduce by 1	As expected	7
20	Player damage [taken] and Enemy [dealt] damage	When the player enters the collider of the enemy	The player's health is reduced by 1	As expected	8
21	Enemy Starts with max health	When instantiated	The enemy will start with max health when it's instantiated	As expected	9
22	Enemy movement	After its instantiated	The enemy will always move towards the player until the enemy is destroyed	As expected	No visual aid is relevant
23	Enemy will always play the EnemyWalk animation	After its instantiated	The enemy will always play the EnemyWalk animation until the enemy is destroyed	As expected	No visual aid is relevant
24	Enemy scale will change with direction (positive, in terms of x)	When the new position is greater than the old position	The local scale of the Enemy will be set to (1,1,1), meaning the colliders, sprite renderer, animator and all other component's will also be set to that scale	As expected	10
25	Enemy scale will change with direction (negative, in terms of x)	When the old position is greater than the new position	The local scale of the Enemy will be set to (1,1,1), meaning the colliders, sprite renderer, animator and all other component's will also be set to that scale	As expected	11
26	Player dies	Health <= 0	The score will be saved in PlayerPrefs("YourScore",0), the game over screen will load	As expected	12
27	Check highscore if its beaten	When the player dies	Save user score as highscore if it is higher than the current highscore	As expected	12,13
28	Player collisions between walls	-	The walls will stop the player from leaving the game map	As expected	14
29	Camera following the player	The player moves	The camera will always have the player in the middle of the screen space	As expected	15,16
30	Camera smooth movement	-	The camera will have a small delay between the player moving and the camera following the player	As expected	15,16

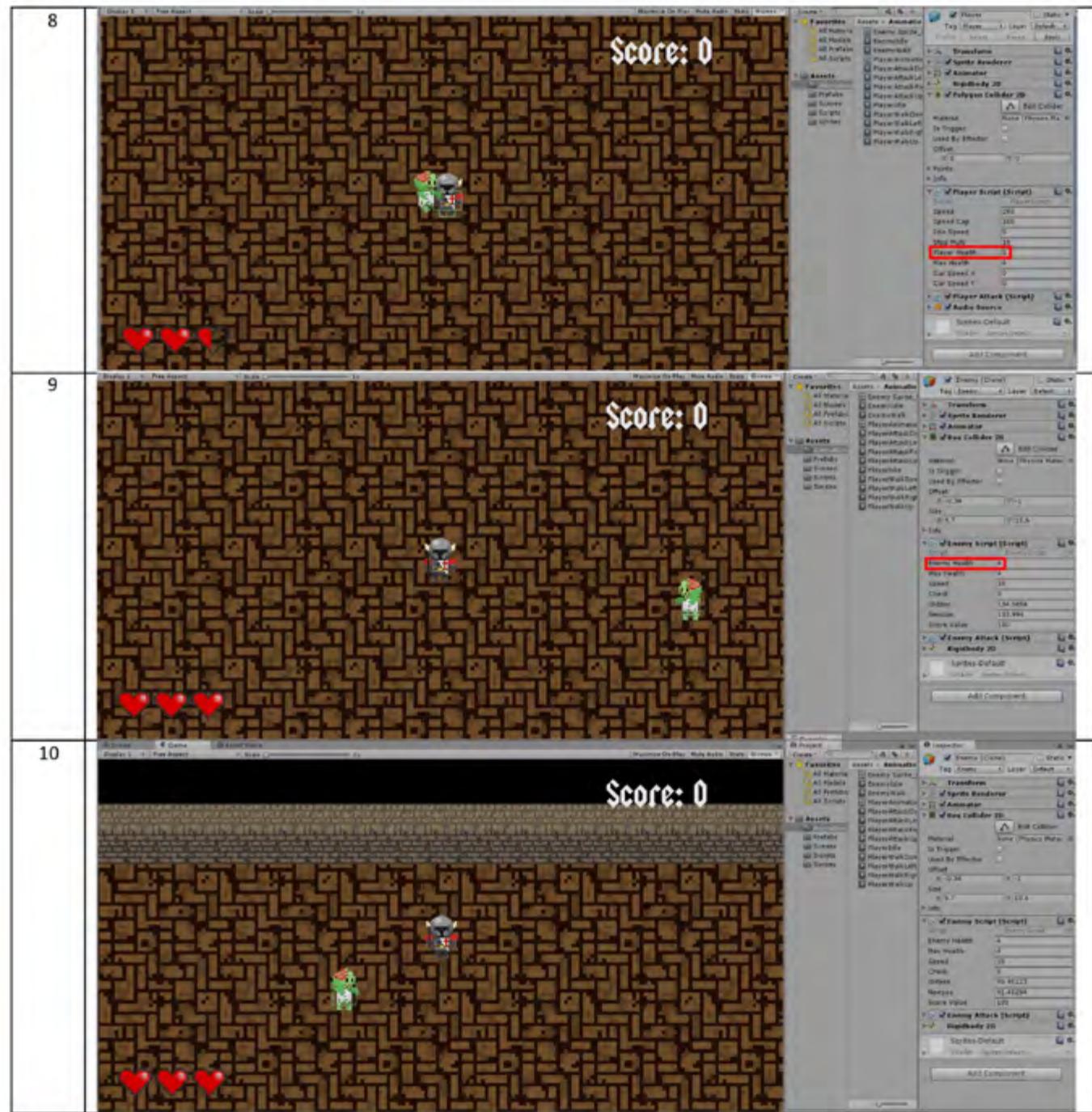
31	Health level image update	The health of the player changes	The amount of health of the player will be shown by the amount	As expected	17
32	Score counter increases when an enemy is defeated	An enemy is destroyed	The amount of points the enemy is worth will be added to the score counter	As expected	18
33	GameScreen – Music	The game screen loads	The background music will play automatically when the game starts	As expected	19
34	GameScreen – SFX	The player attacks	The sound from the attack animation of the player will be played	As expected	20
35	Spawn enemies in random position	-	The enemies will spawn in random locations within the map	As expected but changes are necessary	21
36	Spawn enemies in waves of X+1 enemies	All enemies are destroyed	After all enemies are dead the number of enemies spawned must be the number of enemies in the last wave plus one	As expected	21,22
37	Main Menu – Music	The Main	When the main menu loads the music will start playing	As expected	23
38	Main Menu – Play	Button pressed	The Game Screen will load	As expected	24
39	Main Menu – Exit	Button pressed	The program will close	As expected	No visual aid is relevant
40	Main Menu – Music toggle	Toggle On/Off	The toggle will mute/unmute the music	As expected	25
41	Pause game	"P" key pressed when pause menu is inactive	The game will pause when the pause menu is inactive	As expected	26
42	Un-pause game	"P" key is pressed when the pause menu is active	The game will un-pause when the pause menu is inactive	As expected	27
43	Paused menu – Resume button	The button is pressed	The game will resume	As expected	27
44	Paused menu – Main Menu button	The button is pressed	The current game will be exited and the main menu will load	As expected	28
45	Pause menu – Exit	The button is pressed	The current game will exit and the application will close	As expected	No visual aid is relevant
46	Paused menu – Music toggle	The toggle value is changed (true being unmuted)	The toggle will mute/unmute the music	As expected	29
47	Paused menu – SFX toggle	The toggle value is changed (true being unmuted)	The toggle will mute/unmute the player SFX	As expected	30
48	Game over display user score	-	The score from the last game will be displayed	As expected	31
49	Game over display highscore	-	The highest score ever achieved in the game will be displayed	As expected	31
50	Game over press enter key to continue to main menu	Press "Enter" or "Return" Key	The Main menu scene will be loaded	As expected	32

Screenshots

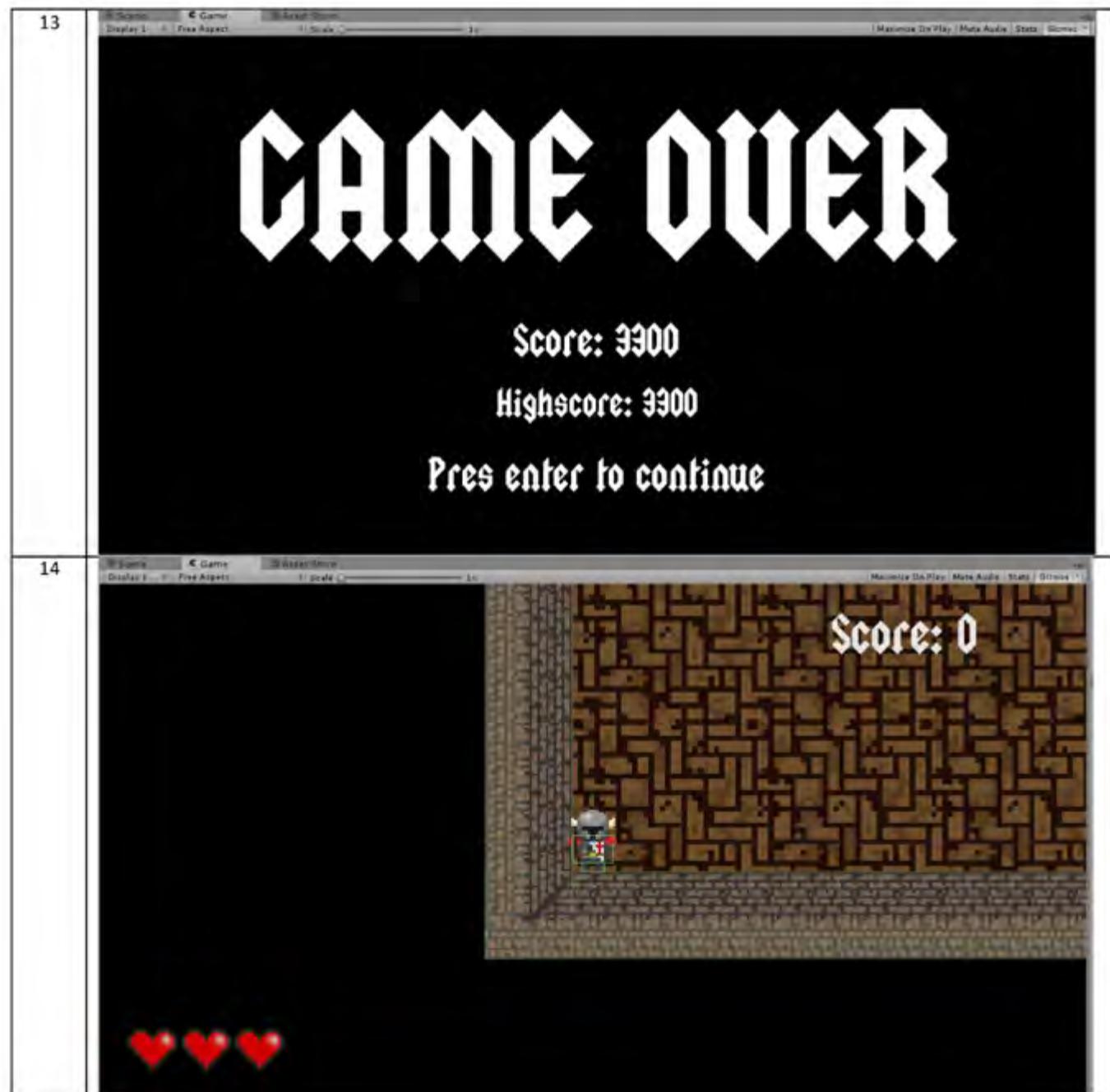










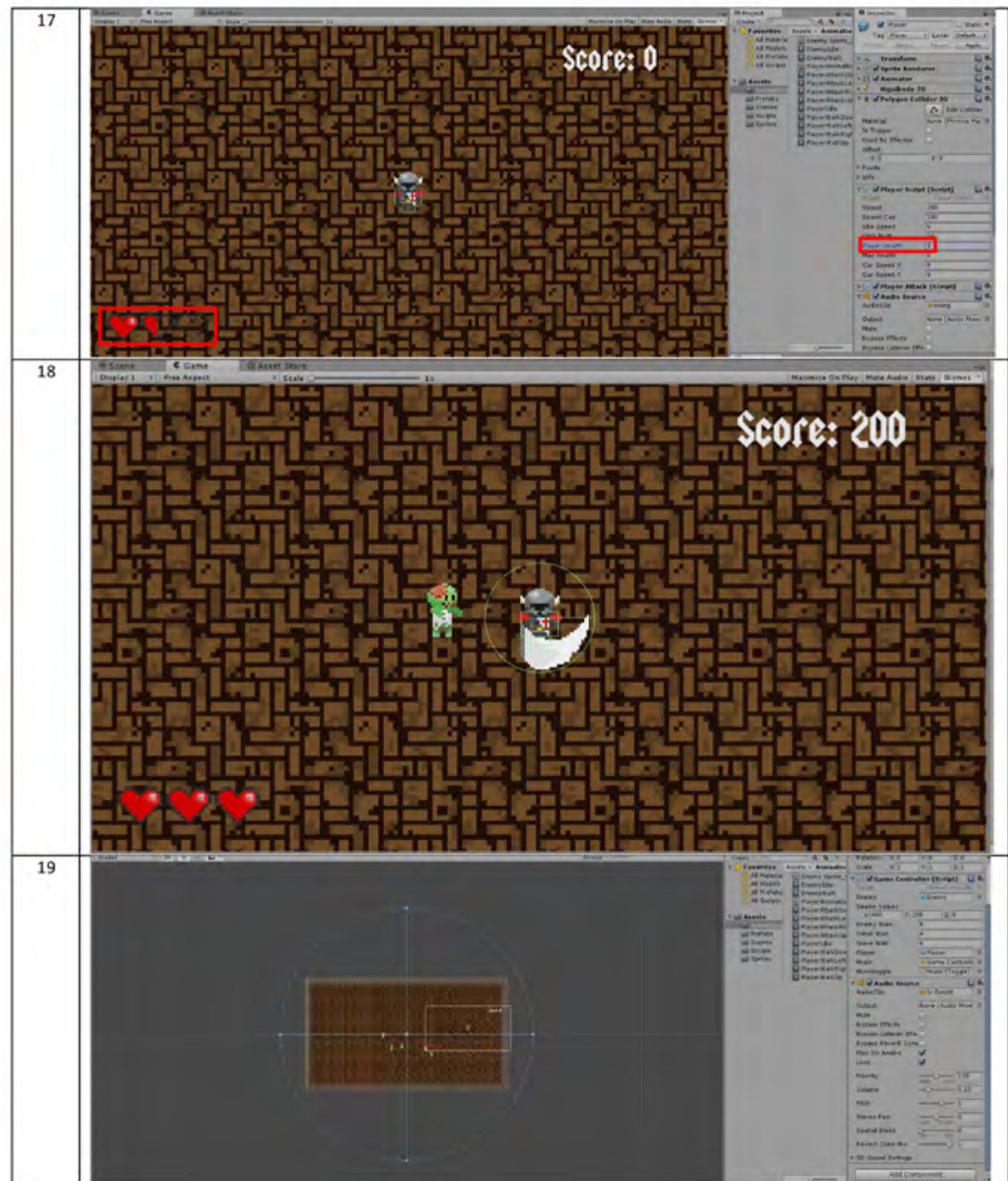


15

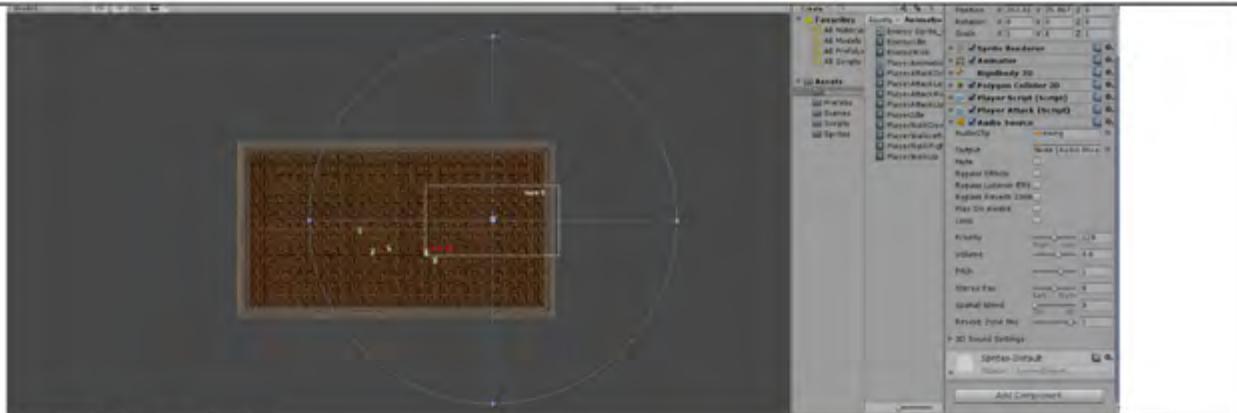


16





20

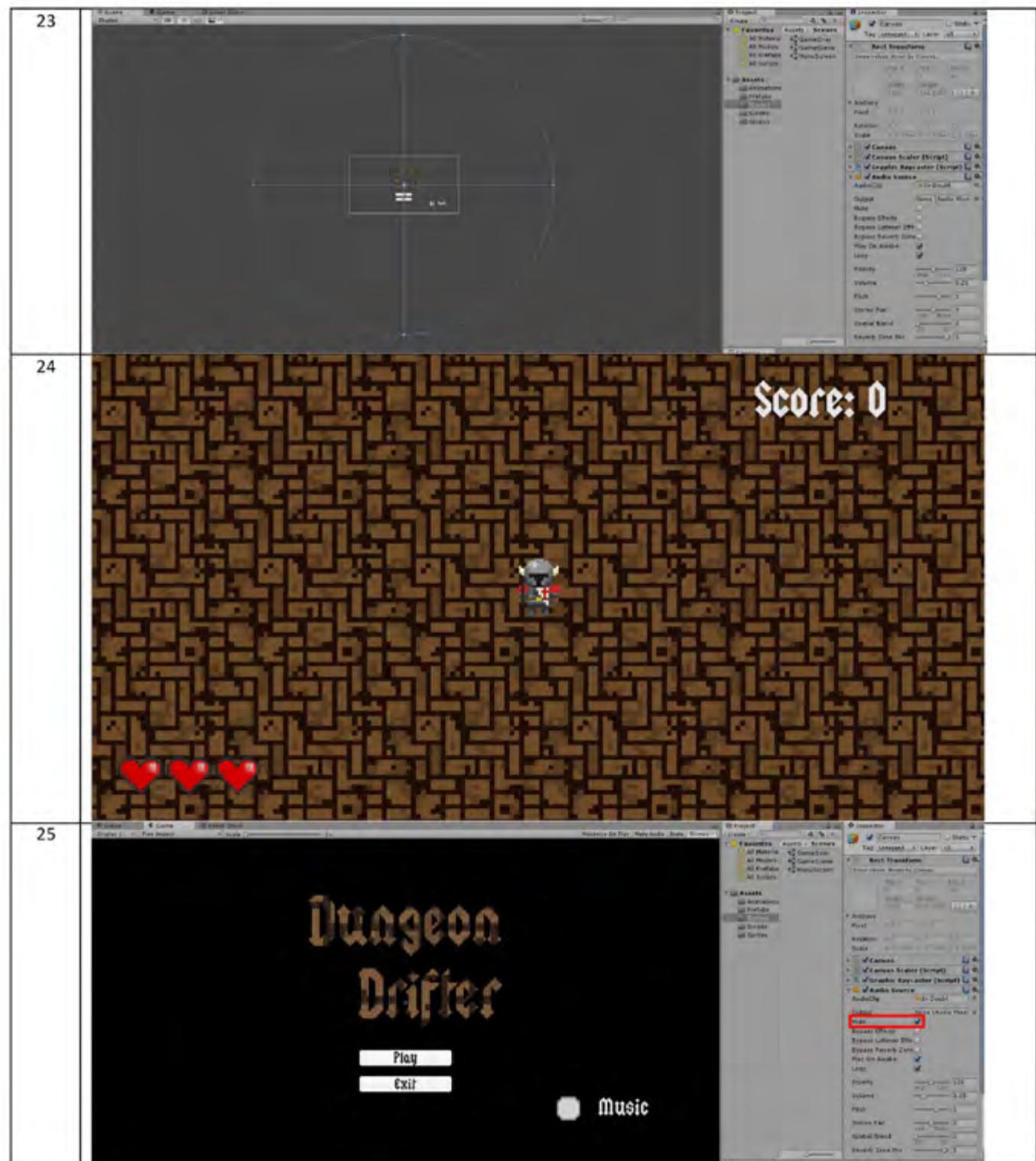


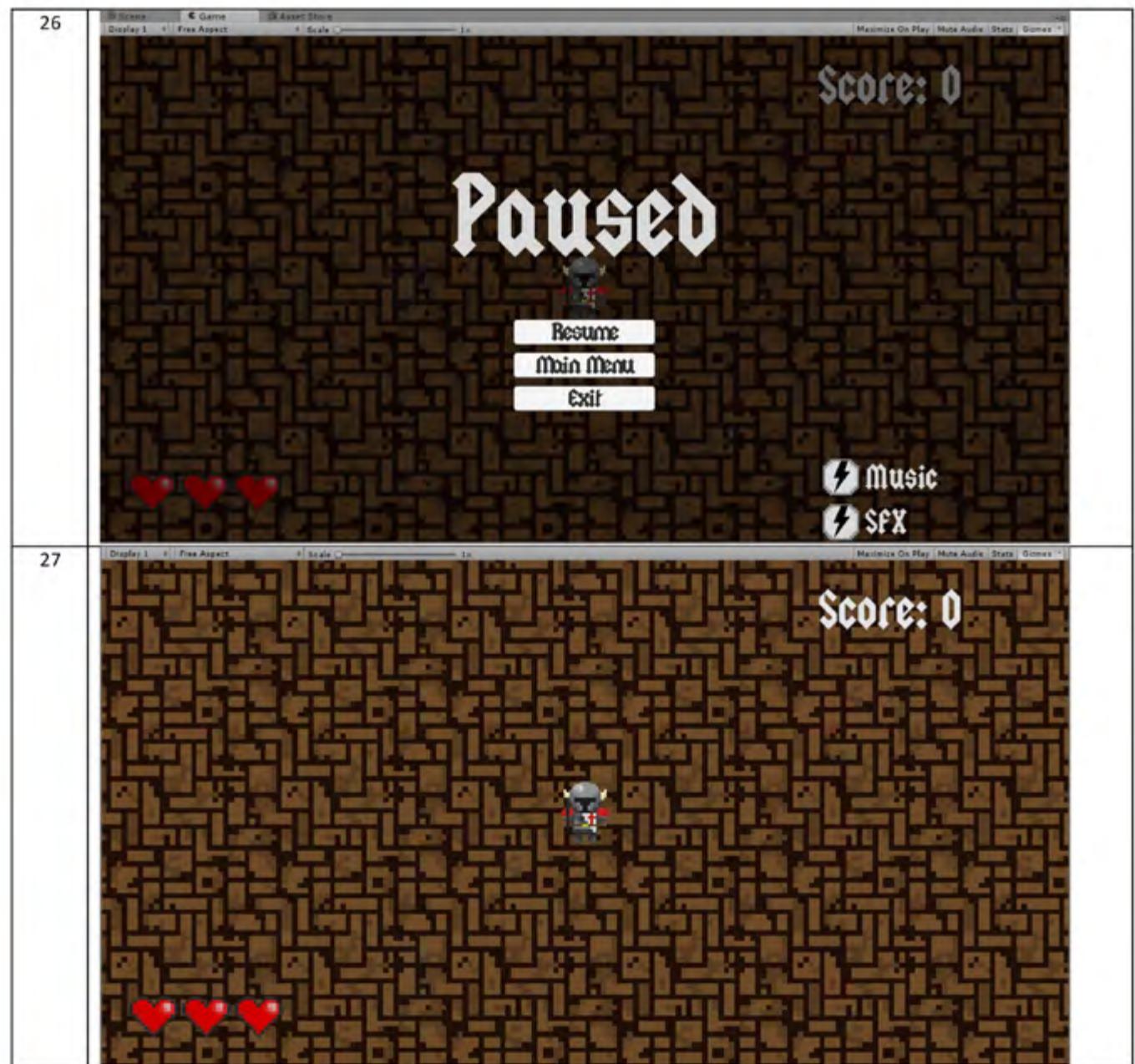
21

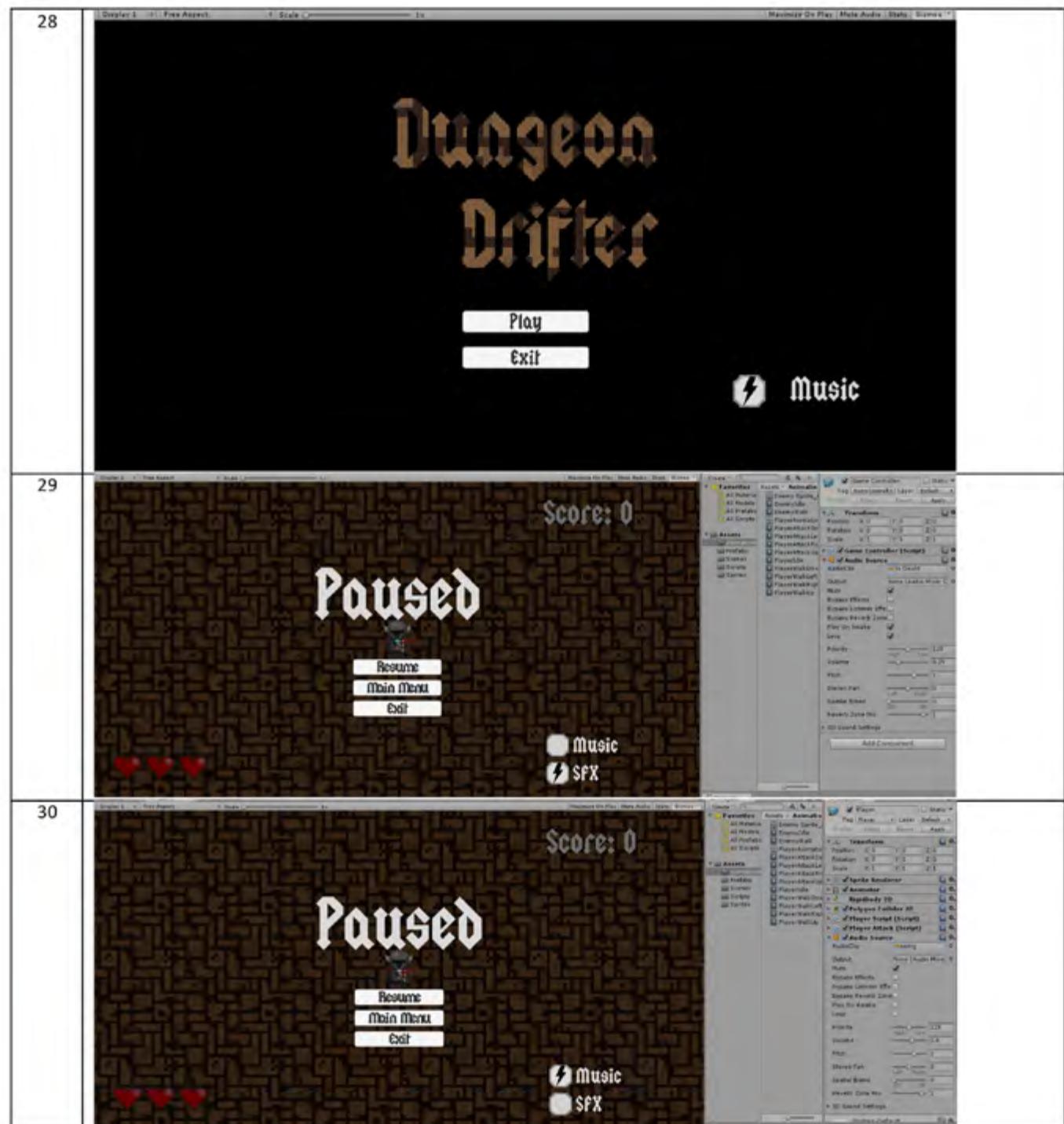


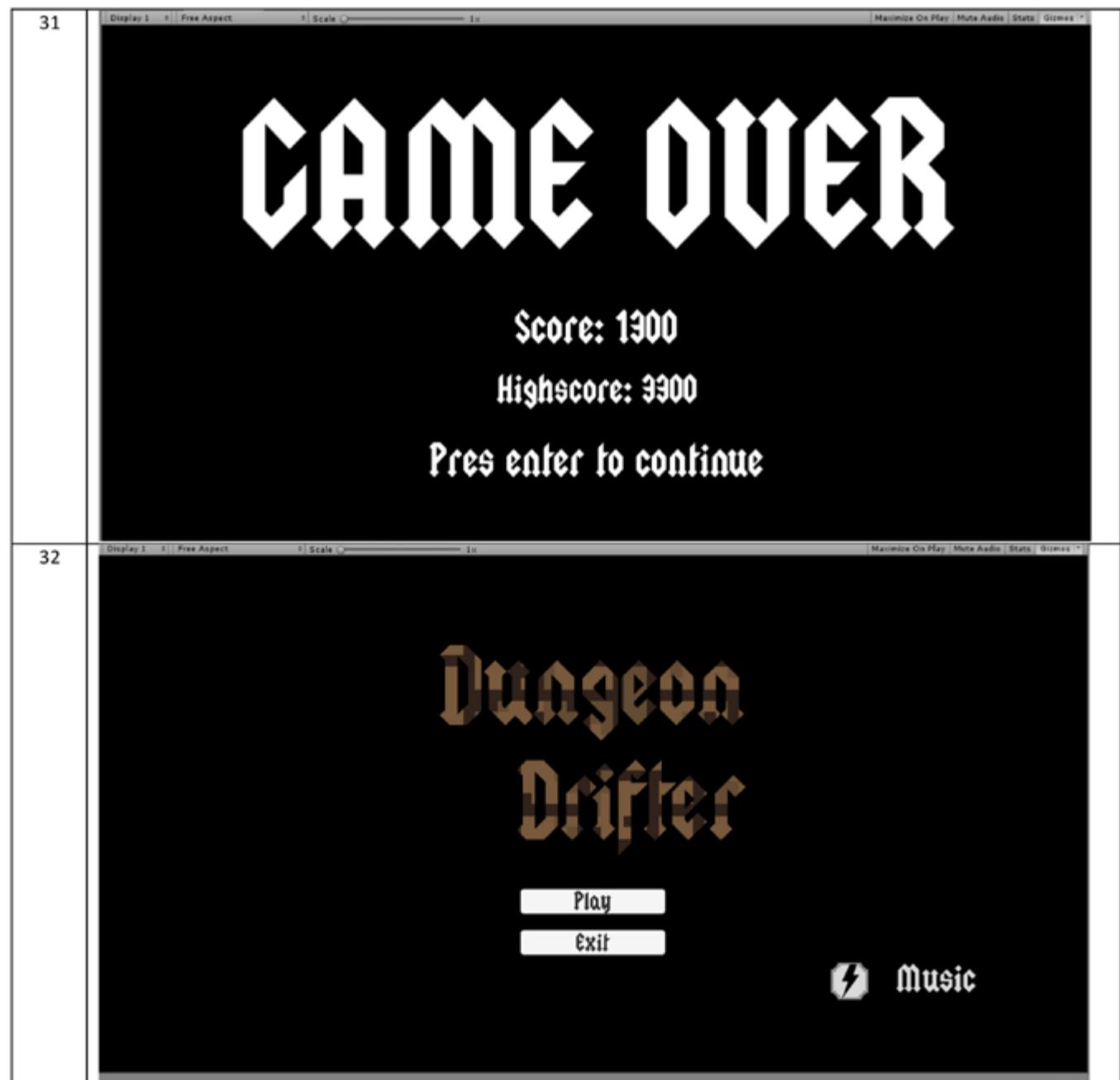
22











Result of Testing

All of the features in the test plan were tested and all of them were successful but I did see an unexpected problem. No changes are necessary for the proper operation of the code, still I have made a few changes which I felt the program needed to improve.

While testing the spawn function I found that the enemies would spawn in the proximity or on rare cases it would spawn on top of the player. Below is an instance where this happened. If the enemy spawned on top of

the player, the player would instantly take damage and that is not something that any user would want. Since this is rather important I have to make sure that enemy is spawned away from the player. Firstly, I didn't know how to do this so I had to look in the physics section of the unity documentation for information and help. I found the Physics2D.OverlapCircleAll() static function which returns a list of colliders that overlap with the generated circle of the function. I used this to make sure the random values created by the randomizer are not in the proximity of the player.



```
//Randomizes the spawn positions and an empty rotations since we dont need rotations but we still need to pass the value to instantiate
Vector3 spawnPosition = new Vector3 (Random.Range (-spawnValues.x, spawnValues.x), Random.Range (-spawnValues.y, spawnValues.y), spawnValues.z);
Quaternion spawnRotation = new Quaternion ();
//Instantiates the enemy with a position and rotation
Instantiate (Enemy, spawnPosition, spawnRotation);
```

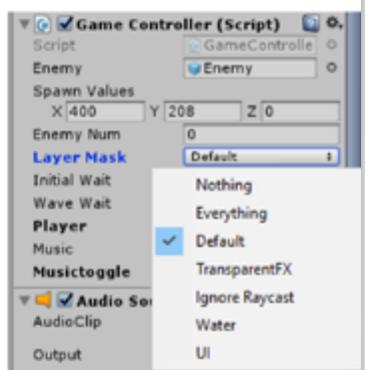
Old code:

New code:

So, before the enemy is instantiated, a circle is of radius 185 with the generated centre returns all the

```
//Randomizes the spawn positions and an empty rotations since we dont need rotations but we still need to pass the value to instantiate
Vector3 spawnPosition = new Vector3 (Random.Range (-spawnValues.x, spawnValues.x), Random.Range (-spawnValues.y, spawnValues.y), spawnValues.z);
Quaternion spawnRotation = new Quaternion ();
//Checks if there is a collider by creating a square with dimensions r= 120 and checking if any colliders overlap
colliders = Physics2D.OverlapCircleAll(spawnPosition , 185,layerMask);
int checkForColliders = colliders.Length;
if (checkForColliders == 0) {
    //Instantiates the enemy with a position and rotation
    Instantiate (Enemy, spawnPosition, spawnRotation);
} else {
    //If the Enemy is not instantiated we need to go through the loop once more
    i--;
}
```

overlapping colliders that are on the default layer (selected from the inspector). Then I check if the length of the array is 0, which if it is, it instantiates the enemy because it means that the player is not in that area. If a collider is detected then "i" is increased by one since will need it to run the loop once more because it didn't instantiate. Now all enemies spawn far away from the player so the user can have time to react, and also not in the view of the camera where it feels unnatural for enemies to just appear.





I did some extensive iterative testing to make sure the enemies would spawn outside of the camera view and of course far from the player and it was successful.

In the "PlayerScript" script, Update() method was a bit cluttered and messy so I decided to call the code that

```
// Update is called once per frame
void Update ()
{
    //Checking if the player's health is equal or Less than 0
    if (PlayerHealth <= 0) {
        Die ();
    }
    if (Input.GetKeyUp ("up")) {
        SlowdownY ();
    }
    if (Input.GetKeyUp ("down")) {
        SlowdownY ();
    }
    if (Input.GetKeyUp ("left")) {
        SlowdownX ();
    }
    if (Input.GetKeyUp ("right")) {
        SlowdownX ();
    }
    PlayerAnimation ();
}

void PlayerAnimation(){
    //Checking which component + magnitude of the velocity of the player is greater
    if (Mathf.Abs(rbody2d.velocity.x) > Mathf.Abs(rbody2d.velocity.y)) {
        //Checking if the velocity of the largest component (in this case x) is positive or negative
        if (rbody2d.velocity.x > 0.1f){
            //Setting the booleans to the appropriate value for the direction of the player to match the sprite
            anim.SetBool ("Up", false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", true);
            anim.SetBool ("Left", false);
        }
        else if (rbody2d.velocity.x < -0.1f){
            anim.SetBool ("Up", false);
            anim.SetBool ("Down", true);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", true);
        }
        else if (Mathf.Abs(rbody2d.velocity.x) > Mathf.Abs(rbody2d.velocity.y)) {
            if (rbody2d.velocity.y > 0.1f) {
                anim.SetBool ("Up", true);
                anim.SetBool ("Down", false);
                anim.SetBool ("Right", false);
                anim.SetBool ("Left", false);
            }
            else if (rbody2d.velocity.y < -0.1f) {
                anim.SetBool ("Up", false);
                anim.SetBool ("Down", true);
                anim.SetBool ("Right", false);
                anim.SetBool ("Left", true);
            }
        }
        //Checking if the absolute or modulus of the magnitude(sqrt((x^2)+(y^2))) is less than the idle speed
        if (Mathf.Abs(rbody2d.velocity.magnitude) < IdleSpeed) {
            anim.SetBool ("Up", false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", false);
        }
    }
}
```

dealt with the SetBoolean for the animation transitions by putting it in its own function:

This makes the code easier to follow so a third-party developer could understand the code better.

Lastly, I added the instructions of how to play the game in the Main Menu scene.



User Testing

Now that I have a working solution and all big issues with the program are dealt with, I will give a built copy to my clients to test and review. They will test the game just like a normal user would use the program, and I have generated this set of simplified questions to make the testing more friendly and understandable.

Test N°	Question	Answer (Yes/No)	Comment
1	Does the application open?		
2	Does music start playing when in main menu?		
3	Do all the buttons/toggles in the main menu work?		
4	Are the buttons/toggles placed appropriately?		
5	Do all the graphics follow the theme?		
6	Are the colour scheme appropriate with the graphics		
7	Is the music appropriate to the style and theme of the game?		
8	Are the instructions clear?		
9	Are the controls appropriate?		
10	Does the player respond appropriately to input?		
11	Does the movement of the player feel responsive?		
12	Are the points displayed and updated accurately?		
13	Does the health level change when it's hit?		
14	Do the enemies die after 4 hits?		
15	Do the enemy objects disappear after they are defeated?		
16	Are the enemies too difficult to defeat?		
17	Do the walls stop you from leaving the game space?		
18	Is the map big enough?		
19	Does the game over scene load when the player's health is 0?		
20	Is the score shown in the game over scene?		
21	Does the main menu load when enter is pressed in the game over scene?		
Extra information relevant to the system that was not covered by the questions above or any other comments regarding the program.			

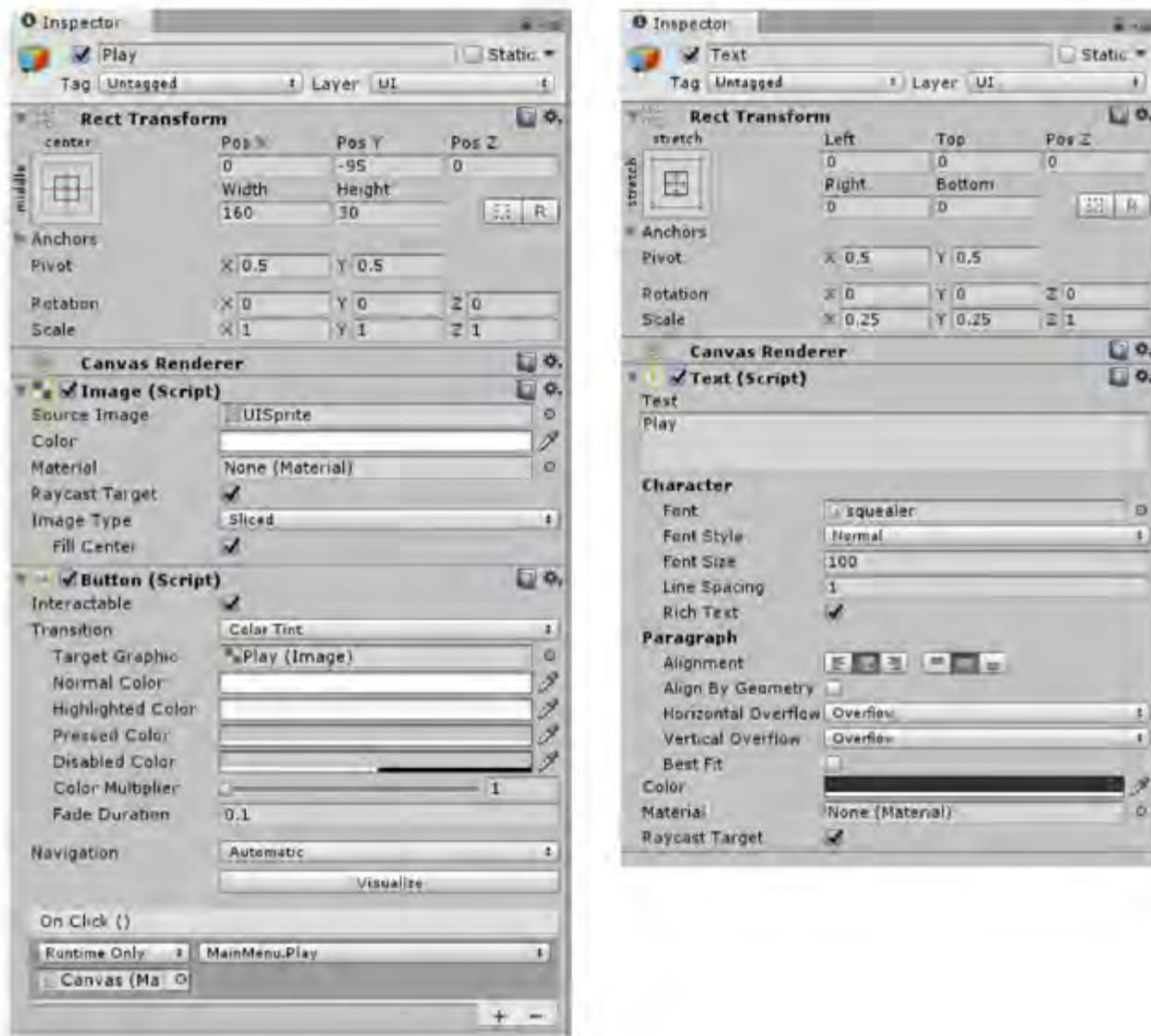
These are the results of the user test

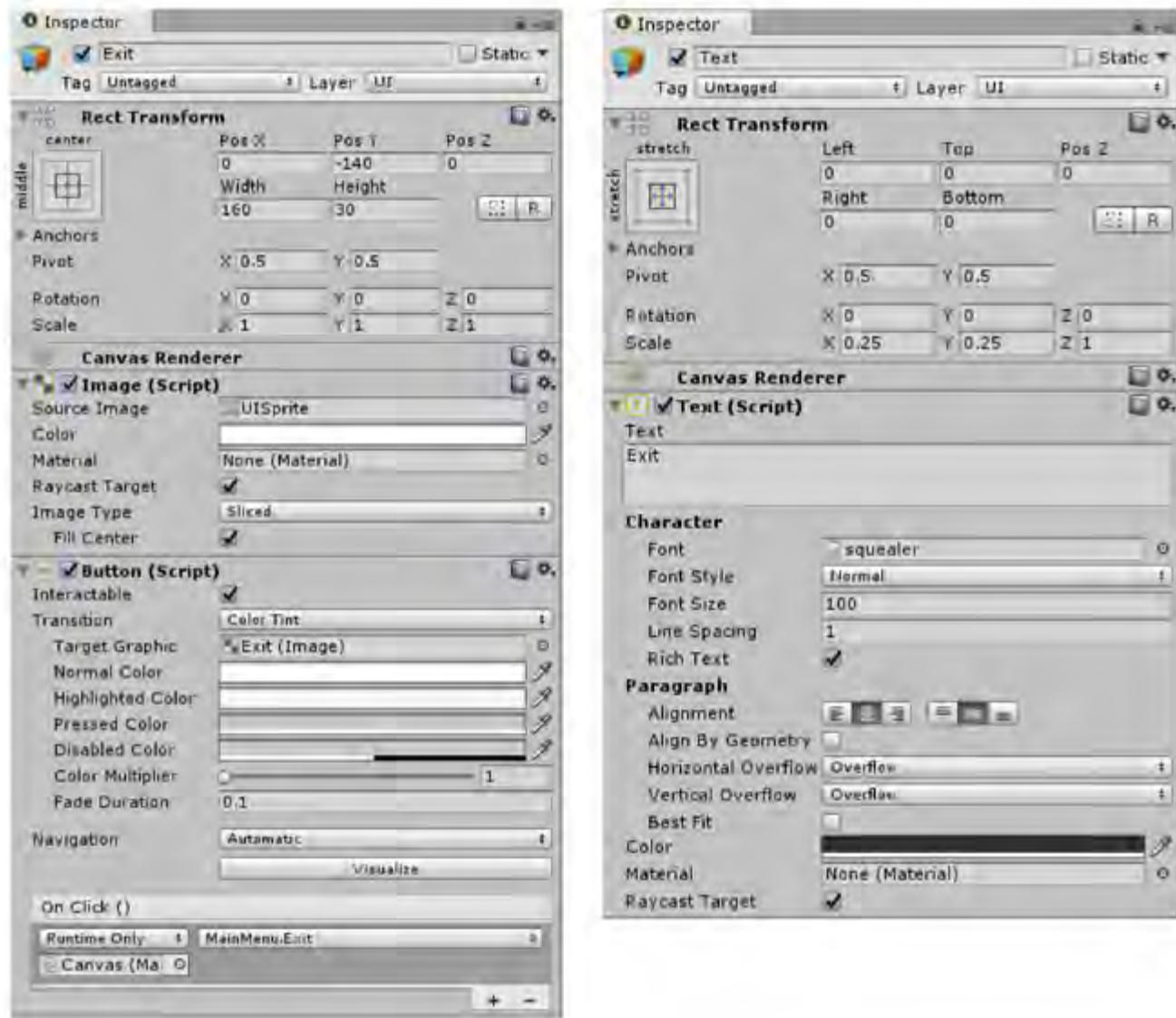
Test N°	Question	Answer (Yes/No)	Comments
1	Does the application open?	Yes[10] No[0]	-
2	Does music start playing when in main menu?	Yes[10] No[0]	-
3	Do all the buttons/toggles in the main menu work?	Yes[10] No[0]	-
4	Are the buttons/toggles placed appropriately?	Yes[10] No[0]	-
5	Do all the graphics follow the theme?	Yes[10] No[0]	-
6	Are the colour scheme appropriate with the graphics	Yes[10] No[0]	-
7	Is the music appropriate to the style and theme of the game?	Yes[10] No[0]	-
8	Are the instructions clear?	Yes[10] No[0]	-
9	Are the controls appropriate?	Yes[10] No[0]	-
10	Does the player respond appropriately to input?	Yes[10] No[0]	-
11	Does the movement of the player feel responsive?	Yes[10] No[0]	-
12	Are the points displayed and updated accurately?	Yes[10] No[0]	-
13	Does the health level change when its hit?	Yes[10] No[0]	-
14	Do the enemies die after 4 hits?	Yes[10] No[0]	-
15	Do the enemy objects disappear after they are defeated?	Yes[10] No[0]	-
16	Are the enemies too difficult to defeat?	Yes[10] No[0]	-
17	Do the walls stop you from leaving the game space?	Yes[10] No[0]	-
18	Is the map big enough?	Yes[10] No[0]	-
19	Does the game over scene load when the player's health is 0?	Yes[10] No[0]	-
20	Is the score shown in the game over scene?	Yes[10] No[0]	-
21	Does the main menu load when enter is pressed in the game over scene?	Yes[10] No[0]	-
Extra information relevant to the system that was not covered by the questions above or any other comments regarding the program.			

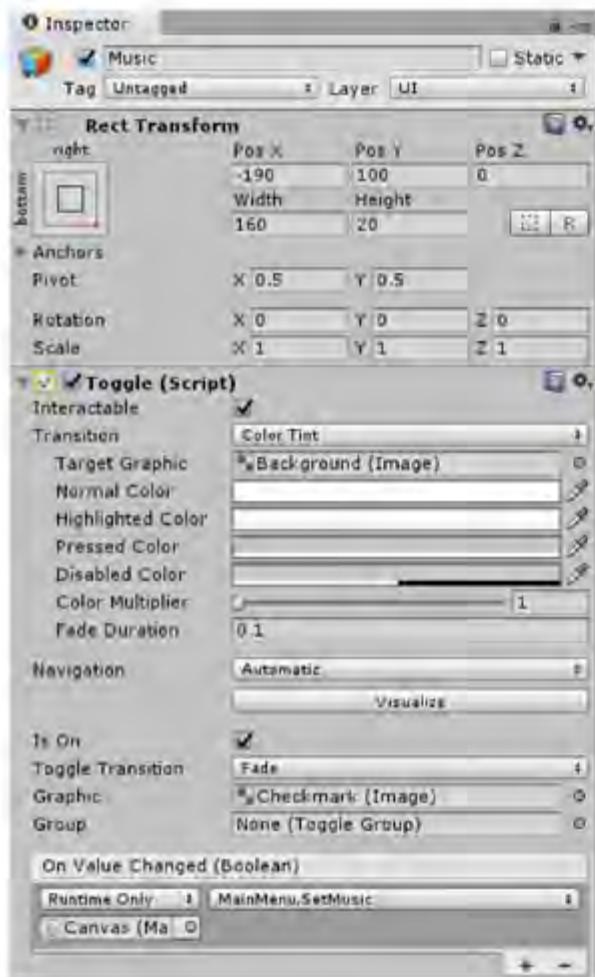
Final Settings and Code

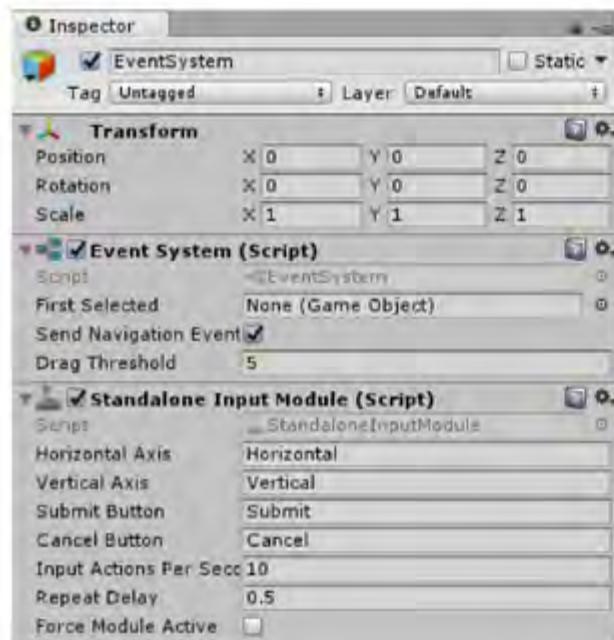
MenuScreen



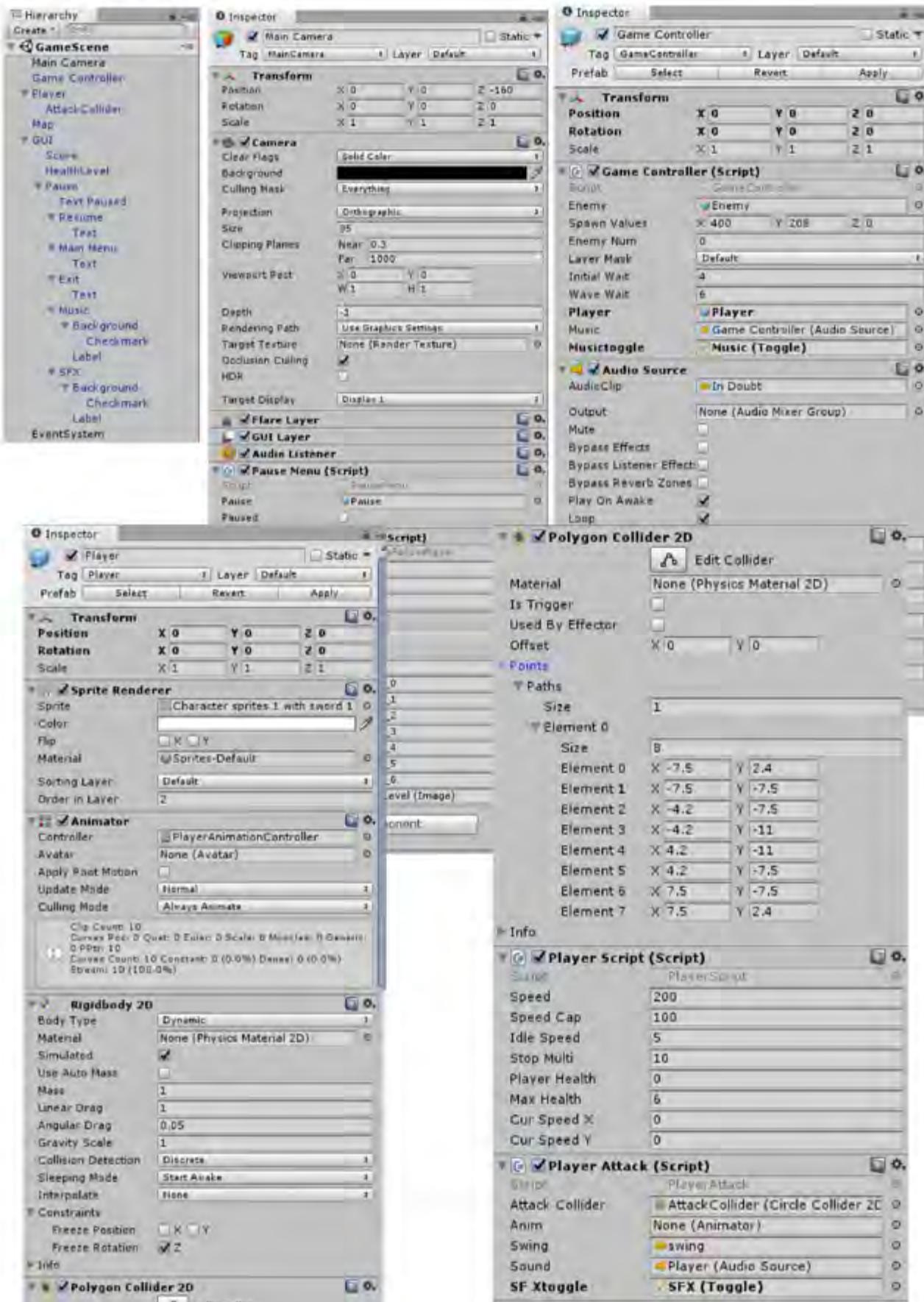


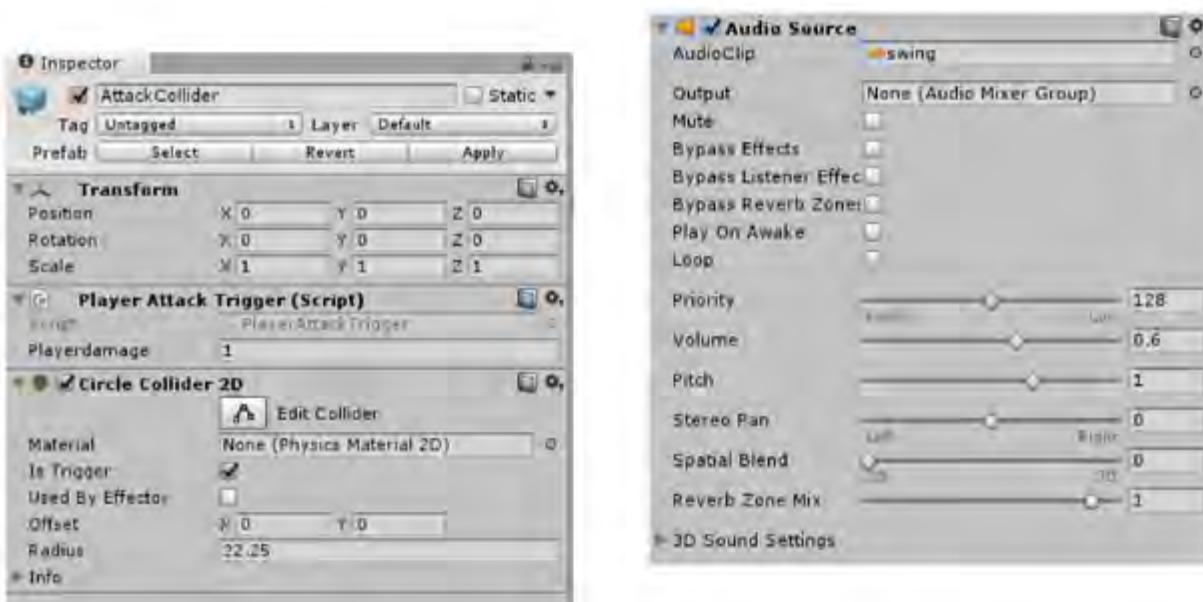


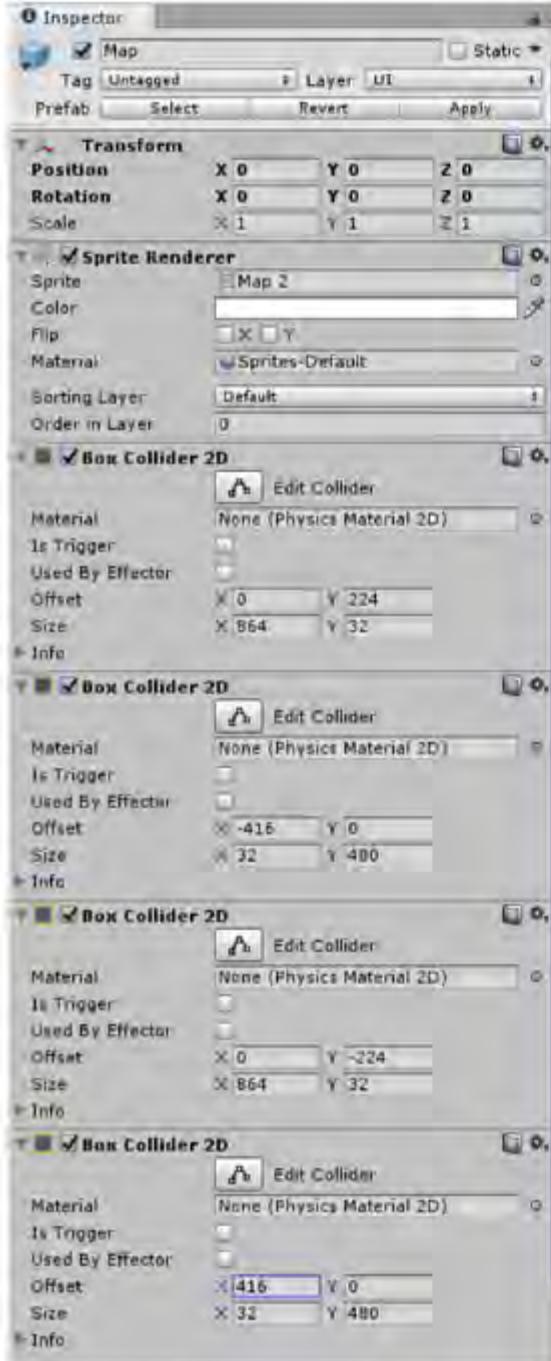


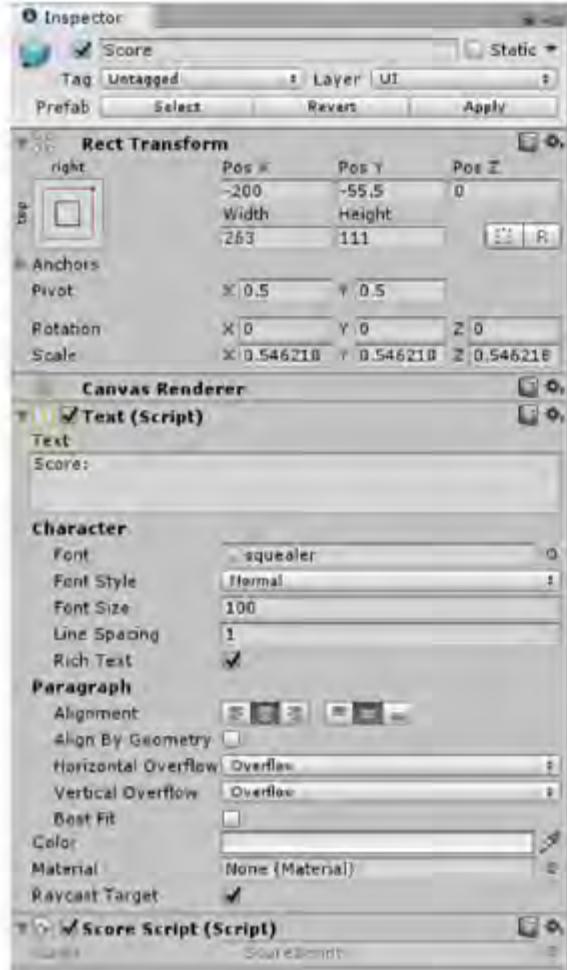


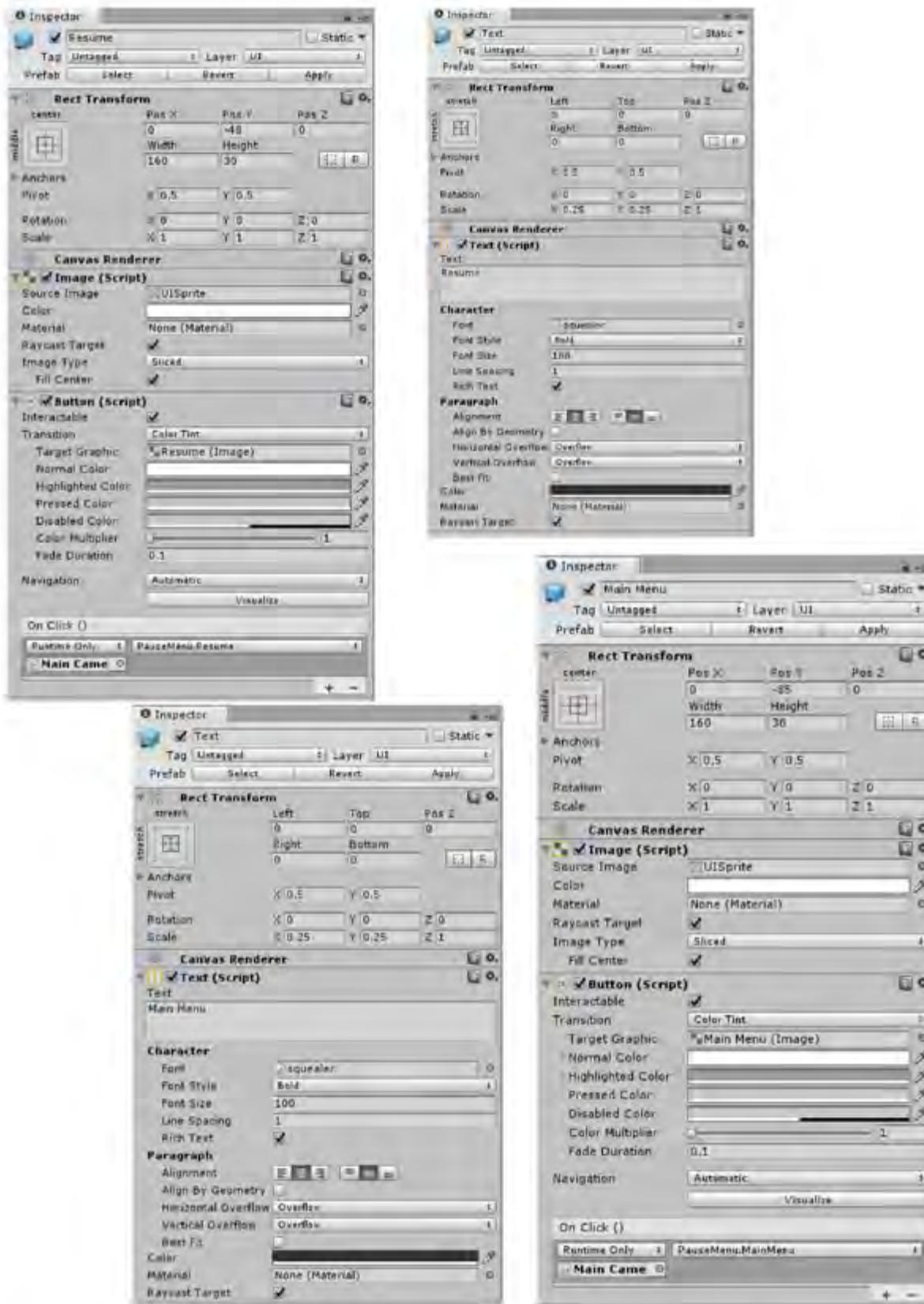
GameScene

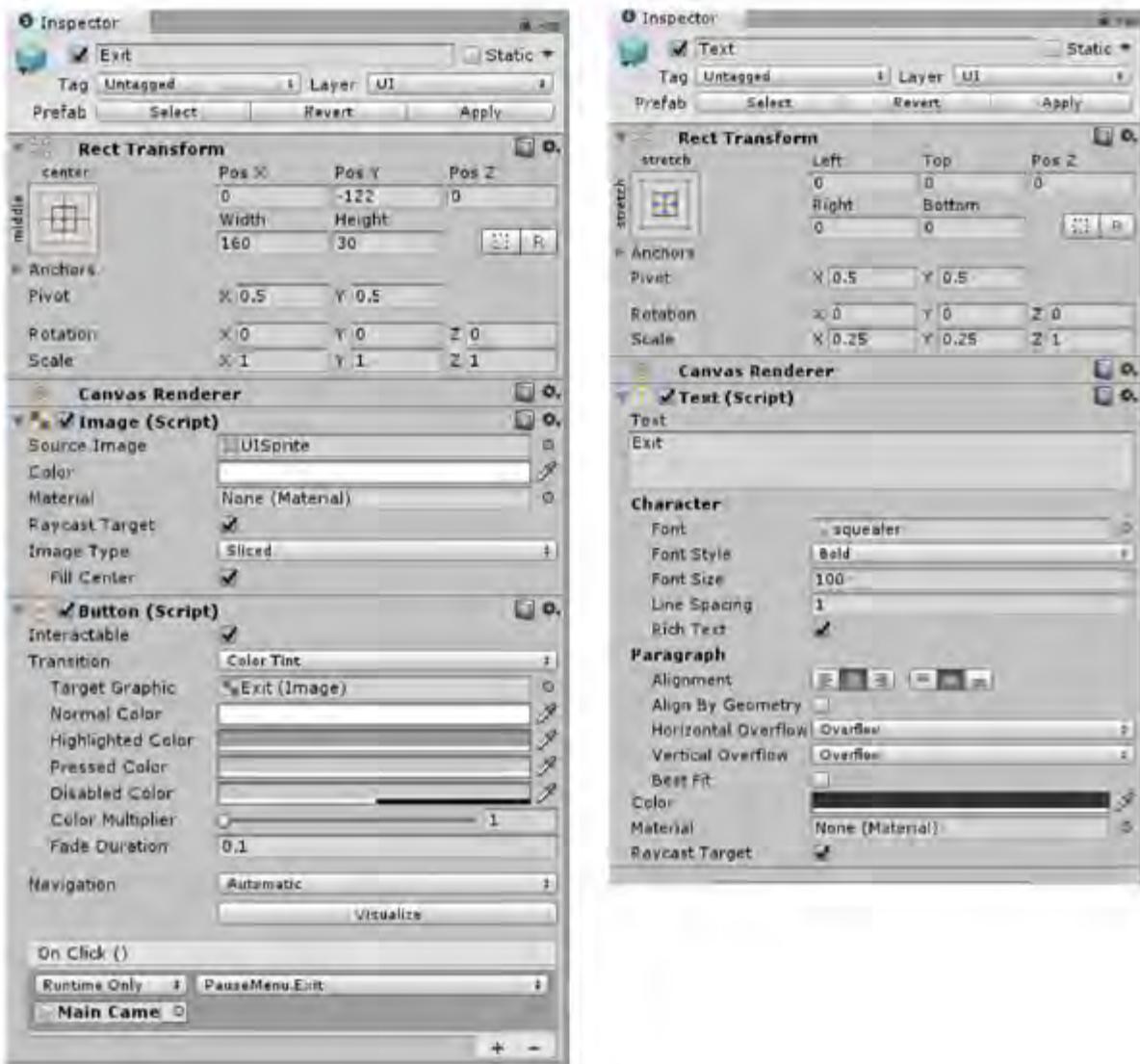


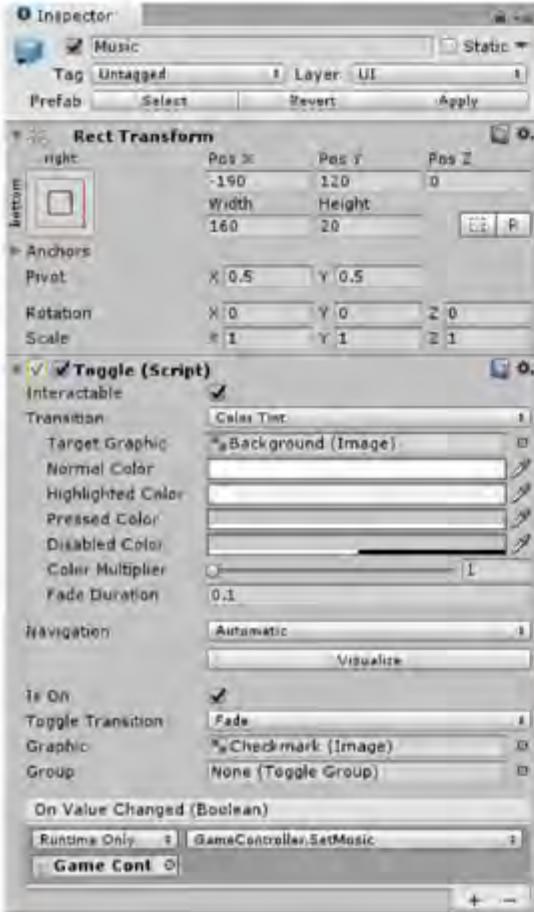


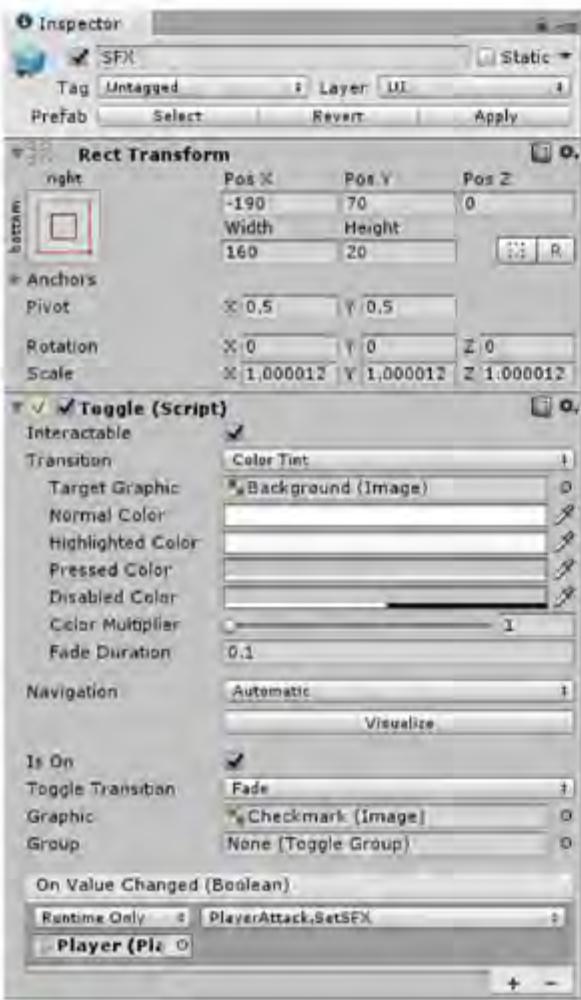


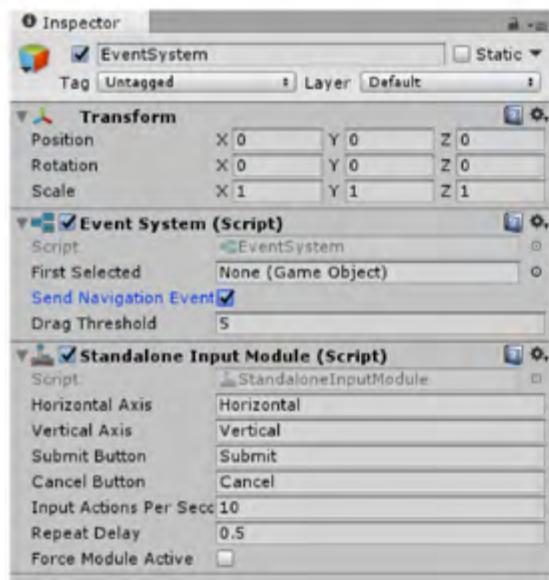




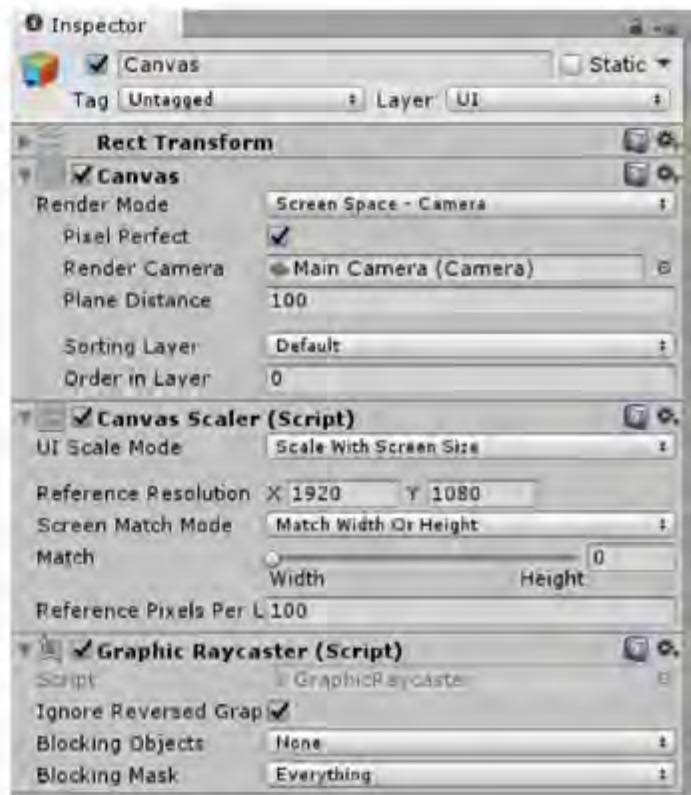


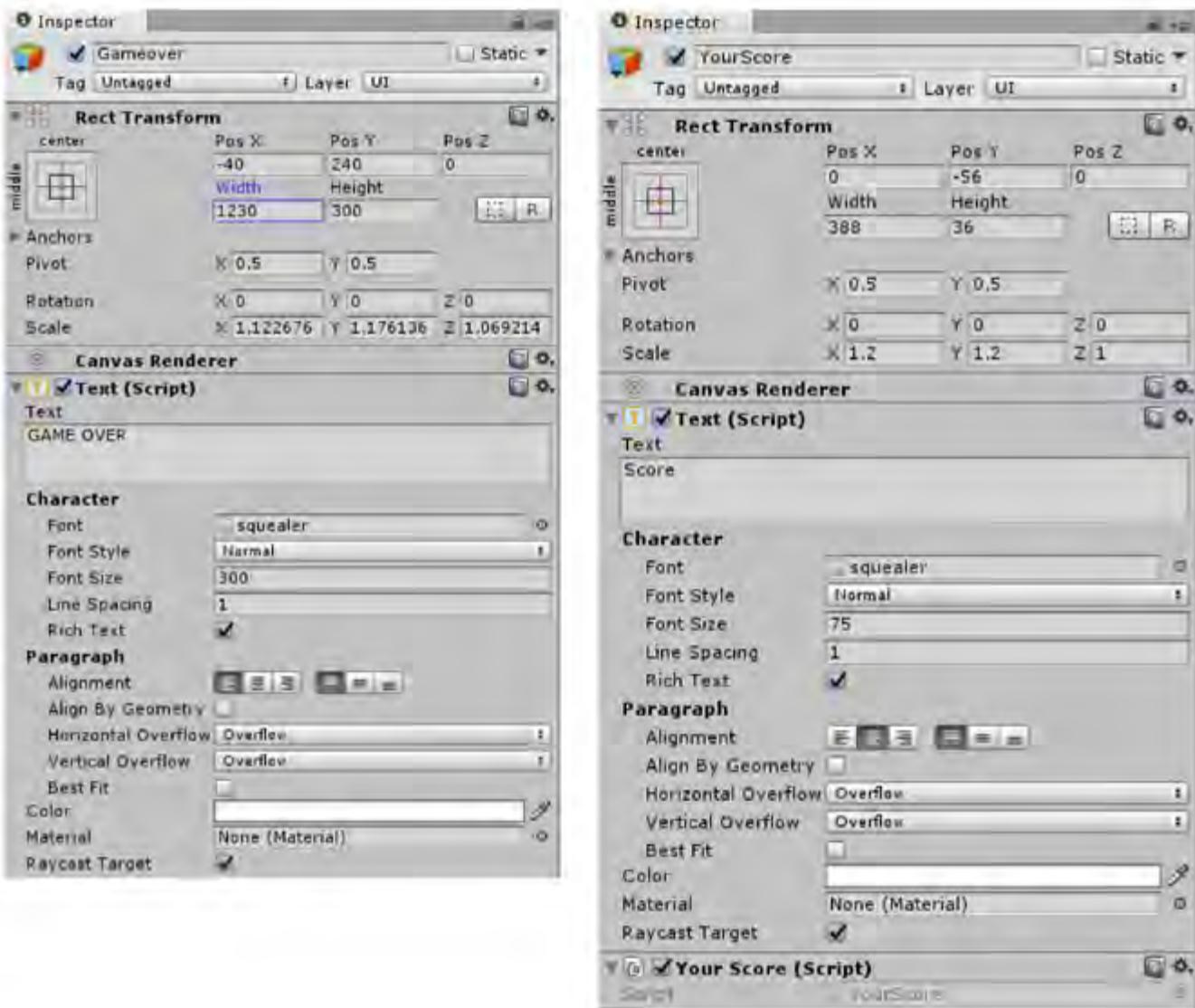




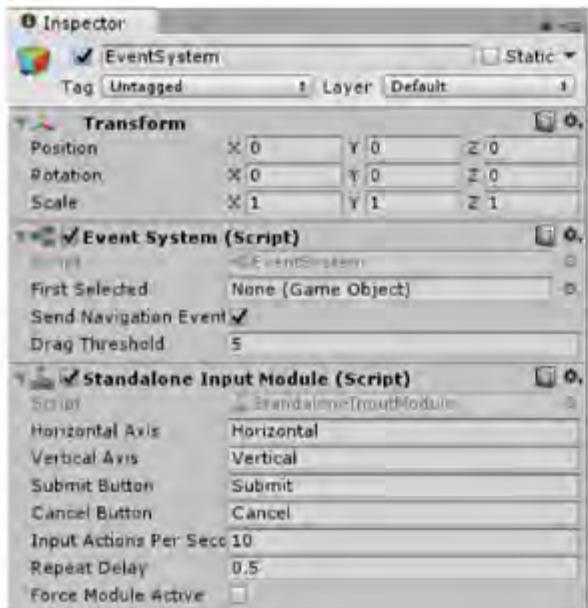


GameOver









Scripts

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraFollowPlayer : MonoBehaviour {

    private Vector2 velocity;
    public float smoothTimeY;
    public float smoothTimeX;
    public GameObject player;

    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    void FixedUpdate ()
    {
        float positionY = Mathf.SmoothDamp(transform.position.y, player.transform.position.y, ref velocity.y, smoothTimeY);
        float positionX = Mathf.SmoothDamp(transform.position.x, player.transform.position.x, ref velocity.x, smoothTimeX);

        transform.position = new Vector3(positionX, positionY, transform.position.z);
    }
}

```

CameraFollowPlayer

GUI

```

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

public class GUI : MonoBehaviour {

    public Sprite[] Hearts;
    public Image HealthLevel;
    private PlayerScript player;

    void Start(){
        player = GameObject.FindGameObjectWithTag ("Player").GetComponent<PlayerScript> ();
    }
    void Update(){
        HealthLevel.sprite = Hearts[player.PlayerHealth];
    }
}

```

GameController

```

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

public class GameController : MonoBehaviour {

    public GameObject Enemy;
    public Vector3 spawnValues;
    public int enemyNum;
    private int cureneNum;
    private GameObject[] count;
    private Collider2D[] colliders;
    public LayerMask layerMask;
    public float initialWait;
    public float waveWait;
    public GameObject Player;
    public AudioSource Music;
    public Toggle MusicToggle;

    void Start(){
        Player = GameObject.FindGameObjectWithTag("Player");
        StartCoroutine (SpawnWave (layerMask));
    }

    //Since I want the code to wait between waves I need to use a coroutine in order to use the field WaitForSeconds();
    IEnumerator SpawnWave(LayerMask layerMask){
        //Wait for initialWait seconds
        yield return new WaitForSeconds (initialWait);
        //Loop that will always run until the game ends
        while(true){
            //Finds all objects in the scene with teh tag enemy and puts them in an array, then I use the lenght of the array
            count = GameObject.FindGameObjectsWithTag("Enemy");
            cureneNum = count.Length;
            //checks if the number of enemies in the scene is 0
            if (cureneNum == 0) {
                for (int i = 0; i < enemyNum; i++) {
                    //Randomizes the spawn positions and an empty rotations since we dont need rotations but we still need to pass teh value to Instantiate
                    Vector3 spawnPosition = new Vector3 (Random.Range (-spawnValues.x, spawnValues.x), Random.Range (-spawnValues.y, spawnValues.y), spawnValues.z);
                    Quaternion spawnRotation = new Quaternion ();
                    //Checks if there is a collider by creating a square with dimensions ~120 and checking if any colliders overlap
                    colliders = Physics2D.OverlapCircleAll(spawnPosition , 120,layerMask);
                    int checkForColliders = colliders.Length;
                    if (checkForColliders == 0) {
                        //Instantiates the enemy with a position and rotation
                        Instantiate (Enemy, spawnPosition, spawnRotation);
                    } else {
                        //If the Enemy is not instantiated we need to go through the loop once more
                        i--;
                    }
                }
                //Every time the for loop finishes another enemy will be added for the next wave
                enemyNum++;
            }
            //Wait for waveWait seconds
            yield return new WaitForSeconds (waveWait);
        }
    }

    public void SetMusic(){
        if (!MusicToggle.isOn) {
            Music.mute = true;
        } else {
            Music.mute = false;
        }
    }
}

```

PlayerScript

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerScript : MonoBehaviour {

    public float speed = 200f;
    public float SpeedCap = 200f;
    public float idleSpeed = 5f;
    public float StopMulti = 10f;
    public int PlayerHealth;
    public int maxHealth = 6;
    public float curSpeedX;
    public float curSpeedY;
    private Rigidbody2D rbody2d;
    private Animator anim;

    // Only runs once
    void Start () {
        rbody2d = gameObject.GetComponent<Rigidbody2D> ();
        anim = gameObject.GetComponent<Animator> ();

        PlayerHealth = maxHealth;
    }
    // Update is called once per frame
    void Update () {
        //Checking if the player's health is equal or less than 0
        if (PlayerHealth <= 0) {
            Die ();
        }
        if (Input.GetKeyUp ("up")) {
            SlowdownY ();
        }
        if (Input.GetKeyUp ("down")) {
            SlowdownY ();
        }
        if (Input.GetKeyUp ("left")) {
            SlowdownX ();
        }
        if (Input.GetKeyUp ("right")) {
            SlowdownX ();
        }
        PlayerAnimation ();
    }

    //When dealing with a forces we have to use FixedUpdate() as it is designed for dealing with rigidbodies as you need to apply the force every fixed frame
    void FixedUpdate()
    {
        //Get the current absolute velocity for x and y components
        curSpeedX = Mathf.Abs(rbody2d.velocity.x);
        curSpeedY = Mathf.Abs(rbody2d.velocity.y);

        //reset the force applied in each frame
        float ForceX = 0f;
        float ForceY = 0f;

        //Cheking for user input and setting the force to the speed in the corresponding +ve or -ve direction for X and Y (4 iterations are for each direction)
        if (Input.GetKeyDown("down")) {
            //Check if the current speed is less than the threshold and if so setting force to speed
            if (curSpeedY < SpeedCap) {
                ForceY = -speed;
            }
        }
        if (Input.GetKey("up"))
        {
            if (curSpeedY < SpeedCap){
                ForceY = speed;
            }
        }
        if (Input.GetKeyDown("right"))
        {
            if (curSpeedX < SpeedCap){
                ForceX = speed;
            }
        }
        if (Input.GetKey("left"))
        {
            if (curSpeedX < SpeedCap){
                ForceX = -speed;
            }
        }
        //Adding a force to the rigidBody with components ForceX and ForceY
        rbody2d.AddForce(new Vector2(ForceX, ForceY));
    }
}

```

```

void PlayerAnimation(){
    //Checking which component x and y of the velocity of the player is greater
    if (Mathf.Abs(rbody2d.velocity.y) > Mathf.Abs(rbody2d.velocity.x)) {
        //Checking if the velocity of the largest component (in this case y) is positive(up) or negative(down)
        if (rbody2d.velocity.y > 0.1f){
            //Setting the booleans to the appropriate value for the direction of the player to match the sprite
            anim.SetBool("Up",true);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", false);
        }
        else if (rbody2d.velocity.y < -0.1f){
            anim.SetBool("Up",false);
            anim.SetBool ("Down", true);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", false);
        }
    }else if (Mathf.Abs(rbody2d.velocity.x) > Mathf.Abs(rbody2d.velocity.y)) {
        if (rbody2d.velocity.x > 0.1f) {
            anim.SetBool("Up",false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", true);
            anim.SetBool ("Left", false);
        }
        else if (rbody2d.velocity.x < -0.1f) {
            anim.SetBool("Up",false);
            anim.SetBool ("Down", false);
            anim.SetBool ("Right", false);
            anim.SetBool ("Left", true);
        }
    }
    //Checking if the absolute or modulus of the magnitude(sqrt((x^2)+(y^2))) is less than the idle speed
    if (Mathf.Abs (rbody2d.velocity.magnitude) < idlespeed) {
        anim.SetBool("Up",false);
        anim.SetBool ("Down", false);
        anim.SetBool ("Right", false);
        anim.SetBool ("Left", false);
    }
}
void SlowdownX(){
    float SlowX = rbody2d.velocity.x * 0.1f;
    float SameY = rbody2d.velocity.y;
    rbody2d.velocity = new Vector2(SlowX,SameY);
}
void SlowdownY(){
    float SlowY = rbody2d.velocity.y * 0.1f;
    float SameX = rbody2d.velocity.x;
    rbody2d.velocity = new Vector2(SameX,SlowY);
}
public void Damage(int Enemydamage){
    PlayerHealth -= Enemydamage;
}
void Die(){
    PlayerPrefs.SetInt ("Score", ScoreScript.score);
    int highscore = PlayerPrefs.GetInt ("highScore", 0);
    if (ScoreScript.score > highscore) {
        PlayerPrefs.SetInt ("HighScore", ScoreScript.score);
    }
    SceneManager.LoadScene (2);
}

```

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

public class PlayerAttackTrigger : MonoBehaviour {

    public int Playerdamage = 1;

    //This function is called when there is a collision between its collider and another collider, and stores that collider under "col"
    void OnTriggerEnter2D(Collider2D col)
    {
        //Checks if the the collider that is colliding with is not a trigger collider, and makes sure its a Enemy object
        if (col.isTrigger == false && col.CompareTag ("Enemy")) {
            //Send a signal to the parent of the collider in this case sends the function "Damage" and the value of the player damage
            col.SendMessageUpwards ("Damage", Playerdamage);
        }
    }
}

PlayerAttackTrigger
```

PlayerAttack

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

public class PlayerAttack : MonoBehaviour {

    private bool attacking = false;
    private float attackcooldown = 0.4f;
    private float attacktimer = 0f;
    public Collider2D AttackCollider;
    public Animator anim;
    public AudioClip swing;
    public AudioSource sound;
    public Toggle SFXtoggle;

    void Start () {
        //Saves the components under variables so I can use them later
        anim = gameObject.GetComponent<Animator> ();
        sound = gameObject.GetComponent<AudioSource> ();
        AttackCollider.enabled = false;
    }

    void Update () {
        //Check if space is pressed and the player is not attacking
        if(Input.GetKeyDown(KeyCode.Space) && !attacking){
            attacking = true;
            attacktimer = attackcooldown;
            AttackCollider.enabled = true;
            sound.PlayOneShot (swing, 0.6f);
        }
        //Timer for the collider to be disabled after 0.4 sec
        if(attacking){
            if (attacktimer > 0) {
                attacktimer -= Time.deltaTime;
            } else {
                attacking = false;
                AttackCollider.enabled = false;
            }
        }
        // To make the animation play while is attacking
        anim.SetBool ("Attack", attacking);
    }

    //Function used by the pause settings
    public void SetSFX(){
        if (!SFXtoggle.isOn) {
            sound.mute = true;
        } else {
            sound.mute = false;
        }
    }
}
```

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

public class YourScore : MonoBehaviour {

    private Text text;

    void Start () {
        int score = PlayerPrefs.GetInt ("Score", 0);
        text = GetComponent<Text> ();
        text.text = "Score: " + score;
    }
}

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

public class HighScore : MonoBehaviour {

    private Text text;

    void Start () {
        int highscore = PlayerPrefs.GetInt ("HighScore", 0);
        text = GetComponent<Text> ();
        text.text = "Highscore: " + highscore;
    }
}

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

public class ScoreScript : MonoBehaviour {

    public static int score;
    private Text text;

    void Start () {
        text = GetComponent<Text> ();
        score = 0;
    }

    void Update () {
        text.text = "Score: " + score;
    }
}
```

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour {

    public GameObject Pause;

    public bool paused = false;

    void Start (){
        Pause.SetActive(false);
    }

    void Update(){
        if (Input.GetButtonDown ("Pause")) {
            paused = !paused;
        }
        //Check if paused is true, then enables the pause menu canvas, and set the time scale to 0(stoped)
        if (paused) {
            Pause.SetActive (true);
            Time.timeScale = 0;
        }
        //Check if paused is false, then disables the pause menu canvas, and set the time scale to 1(normal speed)
        if (!paused) {
            Pause.SetActive (false);
            Time.timeScale = 1;
        }
    }
    //Function for the resume button, it sets the bool pause to false
    public void Resume(){
        paused = false;
    }
    //Function for the main menu button, it will load the scene 0 the scenes are set up on the builder
    public void MainMenu(){
        SceneManager.LoadScene (0);
    }
    //Function for the exit button, it will close the application
    public void Exit(){
        Application.Quit ();
    }
}

```

PauseMenu

GameOver

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

public class GameOver : MonoBehaviour {

    void Start () {
    }
    void Update () {
        if (Input.GetKeyDown("return") || Input.GetKeyDown("enter")) {
            LoadMenu ();
        }
    }
    void LoadMenu(){
        SceneManager.LoadScene (0);
    }
}

```

MainMenu

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

public class MainMenu : MonoBehaviour {

    public AudioSource Music;
    public Toggle Musictoggle;

    public void Play(){
        SceneManager.LoadScene (1);
    }
    public void Main(){
        SceneManager.LoadScene(0);
    }
    public void Exit(){
        Application.Quit ();
    }
    public void Setmusic(){
        if (!Musictoggle.isOn) {
            Music.mute = true;
        } else {
            Music.mute = false;
        }
    }
}
```

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyScript : MonoBehaviour {

    public int EnemyHealth;
    public int MaxHealth = 5;
    public float speed = 35f;
    public float check;
    public float oldpos;
    public float newpos;
    public int ScoreValue = 100;

    private Transform player;

    void Start () {
        //letting player equal to the transform of the object with the "Player" tag which stores the position, rotation and scale of an object
        player = GameObject.FindGameObjectWithTag ("Player").transform;
        //Giving the enemy full health
        EnemyHealth = MaxHealth;
    }

    void Update () {
        oldpos = transform.position.x;
        //Moving towards the player at speed times the time passed since last frame
        transform.position = Vector2.MoveTowards(transform.position, player.position, speed * Time.deltaTime);
        newpos = transform.position.x;
        //Check if the enemy is moving towards the right or left and changing the scale by -1 of the object including the sprite renderer and colliders
        if (oldpos > newpos) {
            transform.localScale = new Vector3 (-1, 1, 1);
        }
        if (oldpos < newpos) {
            transform.localScale = new Vector3 (1, 1, 1);
        }
        //If the enemy has a health of 0 or below the "enemy" object will be destroyed, and will award points equal to its ScoreValue
        if (EnemyHealth <= 0) {
            ScoreScript.score += ScoreValue;
            Destroy (gameObject);
        }
    }

    //Dealing damage to the Enemy
    public void Damage(int Playerdamage){
        EnemyHealth -= Playerdamage;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyAttack : MonoBehaviour {

    public float attacktimer = 0f;
    public float attackcd= 0.5f;
    public Collider2D AttackCollider;

    void Start(){
        AttackCollider.enabled = false;
    }
    void Update (){
        //add time for every frame
        attacktimer += Time.deltaTime;
        //Simple timer for enabling and disabling the AttackCollider
        if ((attacktimer >= attackcd) && !AttackCollider.enabled ) {
            attacktimer = 0;
            AttackCollider.enabled = true;
        }
        if ((attacktimer >= attackcd) && AttackCollider.enabled) {
            attacktimer = 0;
            AttackCollider.enabled= false;
        }
    }
}

```

EnemyScript
EnemyAttack

EnemyAttackTrigger

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

public class EnemyAttackTrigger : MonoBehaviour {

    public int Enemydamage = 1;

    //This function is called when there is a collision between colliders
    void OnTriggerEnter2D(Collider2D col)
    {
        //Checks if the the collider that is colliding with is not a trigger collider, and makes sure its a Enemy object
        if (col.isTrigger == false && col.CompareTag ("Player")) {
            col.SendMessageUpwards ("Damage", Enemydamage);
        }
    }
}
```

All the settings and code were reviewed by my clients all of which were happy with them. I requested a signature to prove they complied with the solution.

Asha John Simeon Andrew
John Svennhyzen

Section 4 - Evaluation

The final version of my program had most of the features and specifications that my clients required, but due to time constraints I wasn't able to fulfil all of them. I will go into each of the requirements and explain why I did or did not achieve them and why they were fulfilled.

1. Game window will be full screen.

I know there is no real merit in my part in this since the IDE I used already provides that feature, but I do believe I fully provided the feature to my clients by choosing the IDE. It allows for the resolution to be changed before the application is launched in a windows form. This setting can actually be changed in window before the application launches.

2. The inputs will be through the keyboard and mouse.

I fully met this simple requirement. This is proven in tests 1 to 8, 14, 38 to 47, and 50. The user is able to control the player movement effectively in terms of movement and attack. All the buttons transition between scenes properly, the toggles mute and unmute the music and SFX correctly and the key "P" pauses and un-pauses the game in the GameScene.

3. User inputs control the actions of the character.

This is proven to be true by the testing done on the player on tests 1 through 8, and test 14. The user is able to control all the actions of the player through the arrow keys and space bar.

4. The character can move in four directions: up, down, left and right.

The requirement is fully met. Something I would have liked to add to the requirements list is the animation on requirements and this goes for all animations. This is proven in tests 1 through 4. The player moves in the four arrow directions according to input.

5. The character will be able to attack.

This is fully met, proven in the test 14. The player initiates starts the attack animation in the direction of movement and the attack collider is enabled to check for collisions.

6. The character view is oblique.

I designed all of the art, graphics and sprites of the game to fit this requirement so it is indeed fully met. This can be backed up by all the screenshots in the GameScene from page 51 to 63.

7. The camera follows the player.

This has proven to be met by the test 29 and 30. When the player moves in a direction the camera moves in the direction of the player with a delay.

8. The map will be a closed area where the player can move in.

This is shown to be met by the graphics and the test 28. The walls in the map delimit the space of the game area, when an enemy or the player tries to move in the direction outside of the wall, they collide and they stop and cannot move in that direction.

9. There are different starting characters with different base attack and health points.

This was one of the requirements I wasn't able to meet due to time constraints.

10. When the character collides with enemy, the player's health reduces.

Fully met, shown by the test 20. When the enemies health changes, the array of the health level images updates to the right one.

11. When the character hits the enemy with a weapon, the enemy's health reduces.

Fully met, shown by the test 19. When the enemy's collider enters the player's AttackCollider the value of the health of the enemy that entered said collider.

12. When an enemy is defeated, points are awarded to the player.

This is shown in the test 32, proven to be fully met. When the enemy is defeated, the health is less or equal to 0, the points that the enemy is worth are added to the score of the player.

13. The enemy will follow the player until the player is dead

This is shown in the test 22 and 23, proven to be fully met. The enemy moves in the direction of the player once it is instantiated and will always do so.

14. There will be different enemy types.

This the second of the requirements I wasn't able to meet due to time constraints.

15. The game progresses in waves: a number of enemies will appear after they are all defeated another wave of increased number of enemies will appear.

Fully met, shown in the tests 35 and 36. The enemies spawn in batches or waves, starting with one, after it is killed, the next wave is queued up to spawn two enemies and so on.

16. Game ends when character's health is depleted.

Fully met, shown in the test 26. When the health of the player is less or equal to 0, the GameOver scene is loaded to tell the user that the game is ended.

17. Game over will display the user total score.

Fully met, shown in the tests 26 and 48. Before the Game over screen is loaded the score of the user is saved in PlayerPrefs and it's called in the GameOver scene. Then the values is displayed as text.

18. Points are and stored under highscore if the score was higher at the end of the game.

Fully met, shown in the tests 26, 27 and 49. Before the value of the player score is saved it is compared to the highscore if it's higher then the user score is also saved as highscore in the PlayerPrefs. After the GameOver scene is loaded the highscore is called and displayed as text.

Addressing unmet Specification requirements

The two requirements that I wasn't able to meet are quite complex, in all actuality they are features that only some of my clients commented about and I put them down because I really did want to fulfil my client's needs. But simply put I did not have enough time to develop them. I had to learn how to deal with unity and its extensive library of classes, functions and methods; which wasn't the worst. It took me a lot of time learning and getting use to the user interface of unity which made me waste time by referencing the wrong objects or classes. In the further developments I will make sure I take into account time for learning the IDE/SDK that I will be using. After I wasn't able to fulfil these requirements I reported it to my clients by email, I only got one partial negative response of one of my clients:

Re: Changes to the Specification Requirements 

 para mí 

Hey Carlos,

Oh well, that is a shame I was really looking forward to the different types of characters, but I'm fine with the changes.

Limitations

The program is limited by the programs coding and the inheritance of it being a game. The program will be used only by people that have a computer, play videogames and like a 2D retro arena game; which cut the amount of potential clients by a many orders of magnitude. To potentially deal with this the game could be made more accessible such as making it available for smartphones and consoles, since the market for mobile applications is currently exploding with potential customers. It's also limited in its playability, by creating more playable characters it would attract customers that like to replay the game in a different style. Furthermore having more enemies it would make the game play more exciting than defeating the same enemy over and over.

Improvements

- Add different characters with different statistics or character customization – this would make the game more of a personal experience and make it feel less repetitive. It would also allow the user to replay the game many times and get a different experience every time.
- Enemy diversity – defeating the same enemy over and over is fun at the start then it will get repetitive very quickly, enemy diversity would mean that a different approach would have to be taken to defeat the different types of enemies, which would make for a more challenging and fun experience.
- Enemy AI – By having more complex movement and attack patterns the game would be hugely improved.
- Enemy knockback – to put it simply when the enemy is damaged it would “flinch” backwards the same as when someone would get hit and would step backward after the fact. And also a visual cue/animation to indicate the user when the enemy was hit.
- Player/Enemy animations – If the game was going to go into production I would like the movement animation to have an extra three or four more frames to make the visuals more appealing.
- Improvements on the code – I feel that the code is not the most optimal and that it could be optimised for faster load times; less space required on memory; better understanding of the code; and to make it easier/faster to compute each frame.

References

Credits –

Music: <http://www.purple-planet.com>

Font License: Typodermic Fonts Inc. End User License Agreement (02-2014)

Examiner commentary

Question/Part: AO 2.2 Analysis

Marks: 7/10

The problem definition is well-written, introduces the idea well and leads to a description of who the stakeholders will be and how they may make use of the solution.

There is no evidence of any discussion of computational methods or approaches to be used.

There number of investigation strategies are used to collect data about potential features and approaches. Each of these are analysed and lead to a list of requirements for the solution, including hardware and software. These requirements are simply described and measurable, however they lack the explanation required for higher marks.

Several limitations of the solution are discussed. These are general limitations of software for impaired users rather than specific to the student's solution but they have attempted to explain how they will address these.

The teacher's mark can be agreed based on a best-fit approach; the evidence provided does meet most of the descriptors for level 3.

Question/Part: AO 3.1 Design

Marks: 10/15

The design objective bullet points describe the breakdown of the system. There are additional modular diagrams also explain some decomposition of the solution, although most elements could be broken down further.

The pseudocode algorithms are well presented and easy to follow; however although several at the start of this work have explanations, the rest are standalone and the purpose and connection between them becomes less clear.

There are a number of GUI designs and listings of buttons/layouts for each screen; however the link to usability is not always made explicit. An explanation of why the colours/fonts/layout etc are 'more pleasing to the eyes' would strengthen this work.

Key variables are described in the data dictionaries. No validation is identified, however this is a game with no data input except movement/ clicks therefore validation is not necessarily appropriate.

An outline of some testing of the solution is planned and shown using test tables – this is simplistic and how to carry out the tests is not always clear.

There is no plan for any post-development testing.

The teacher's mark of 10 for this section is a little lenient. The work overall lacks explanation expected to be awarded marks from the third level. Alongside this, not all of the descriptors for the second level are met and therefore best-fit would indicate a mark at the top of the second level.

Question/Part: AO 3.2 Developing the coded solution

Marks: 12/15

The description of development is reasonably good. Each section/ stage of work implemented is explained well with accompanying code, settings and interface evidence. The evidence is a little hard to see in places and the conversion to PDF has caused some elements to layer.

At some stages of development there are reviews of the changes needed to code and explanations of why plans were changed. There are some before/after screenshots to accompany these. Although the testing and reasons for these changes are described, no evidence is provided of these tests being carried out during the development.

Validation (or the lack of need for it) is discussed on page 50.

It is clear that the variables and structures are well named.

Code listings are provided at the end of the project which show most subroutines have a reasonable amount of annotation/ comments.

Question/Part: AO 3.2 Testing to inform development

Marks: 10/10

The teacher's mark for this section is generous and cannot be agreed.

Although the students describes tests done to check function during development, there is no evidence of this testing provided in the write-up of the development section of the work.

The student clearly states at the top of their Testing section that this testing was carried out after the program was finished. This technically could be described as end stage alpha testing prior to beta, and a few small updates to the work are shown here.

There is only one test where the result is listed as requiring changes, which are shown on pages 68-69. This work should not be fully credited in this section as on the whole, it is not informing development.

The work has also been used to award the students marks in the next section, which should be marked independently.

Taking all of these points into account, the best fit for this work falls into Mark Band 2; there is some explained evidence of developmental testing and a little evidence of remedial action.

Question/Part: AO 3.3 Testing to inform evaluation

Marks: 5/5

The testing tables show post-development testing of functionality. However it is not made clear that the solution is stress-tested for robustness over and above normal testing. Evidence of the tests is provided in the form of static screen shots.

The end-user test table is a sequence of Yes/No questions. The answers are summarised as 10 marks awarded as Yes for each 'does this work' type question. No comments were given and there is no additional annotation for the user testing.

This work does not meet the descriptor for level 4, and does not fully meet the descriptors for level 3; it is generously marked.

Question/Part: AO 3.3 Evaluation of solution

Marks: 12/15

Each user requirement is evaluated and the comments cross-referenced nicely to the tests which support the statements made. This is good practice.

There are two unmet criteria which are discussed and brought through to an improvements section, although detail about how these could be addressed is lacking.

No real comment is made on the usability features – this could be in part due to the fact that they were not clearly identified in the Design stage.

The improvements discussed and in some cases reasons for these changes are explained.

Several limitations are identified, although one is a generic platform issue. No maintenance issues are discussed.

On the whole the evaluation section lacks depth. The line of reasoning is clear and supported with some evidence, however not all aspects of the descriptors for level 3 are fully met. The teacher's mark for this section is a little lenient.



We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

Whether you already offer OCR qualifications, are new to OCR, or are considering switching from your current provider/awarding organisation, you can request more information by completing the Expression of Interest form which can be found here:

www.ocr.org.uk/expression-of-interest

OCR Resources: the small print

OCR's resources are provided to support the delivery of OCR qualifications, but in no way constitute an endorsed teaching method that is required by OCR. Whilst every effort is made to ensure the accuracy of the content, OCR cannot be held responsible for any errors or omissions within these resources. We update our resources on a regular basis, so please check the OCR website to ensure you have the most up to date version.

This resource may be freely copied and distributed, as long as the OCR logo and this small print remain intact and OCR is acknowledged as the originator of this work.

OCR acknowledges the use of the following content:
Square down and Square up: alexwhite/Shutterstock.com

Please get in touch if you want to discuss the accessibility of resources we offer to support delivery of our qualifications:
resources.feedback@ocr.org.uk

Looking for a resource?

There is now a quick and easy search tool to help find **free** resources for your qualification:

www.ocr.org.uk/i-want-to/find-resources/

www.ocr.org.uk/alevelreform

OCR Customer Contact Centre

General qualifications

Telephone 01223 553998

Facsimile 01223 552627

Email general.qualifications@ocr.org.uk

OCR is part of Cambridge Assessment, a department of the University of Cambridge. *For staff training purposes and as part of our quality assurance programme your call may be recorded or monitored.*

© **OCR 2017** Oxford Cambridge and RSA Examinations is a Company Limited by Guarantee. Registered in England. Registered office 1 Hills Road, Cambridge CB1 2EU. Registered company number 3484466.
OCR is an exempt charity.

