OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Bishes Limbu

5395

61813

Wave Survival Game

H446-03 – PROJECT CONTENTS

CONTENTS

A. Analysis 3

B. Design 34

C. Developing the coded solution ("The development story") 95

D. Evaluation 163

Project Appendixes 192

A. ANALYSIS

DESCRIPTION OF THE GAME

My game will be an 8-bit platformer and a horde survival game. The player will control a Spartan that must survive waves of incoming creatures. Each wave will have an increasing number of enemy soldiers the player must fend off. To aid the player there will be power-ups available, which are dropped from the sky, which allows the player to kill enemies more easily. Additionally, the enemies will have a chance to drop ambrosia, the food of the gods, which heals the player. Each time a player kills or attacks an enemy he is awarded points that will accumulate to his score. When attacks are used in combos it will apply multipliers on the score to allow the player to earn more points. Subsequently, when enemies hit the player, points will be deducted from his score as well as losing Health Points, which will determine if the player will die. As the round number increases so will the difficulty, enemies will become more abundant and the ferocity/strength of the enemies will increase. The player must survive 10 rounds to win the game.

The game will put the player in an intense situation where the player is under heavy pressure. The player must fight through and kill many enemies and survive the assault. Due to the game being surrounded by fighting, violence, and death, the game will be targeted at teenagers who love action, survival and fighting games. The game will be playable with just a PC or laptop, which most teens own.

SUTIABLE STAKEHOLDERS AND WHY

A stakeholder for my game would be any teenager or young adult who loves games. However, due to the game being very violent, the stakeholders will most likely be teenagers or young adults. My game is a retro 8-bit and a horde survival game, it would be ideal to find teenagers who are interested in 8-bit style games and wave survival games.

STAKEHOLDER 1: BENJAMIN CHARLTON HAMMOND

One of my stakeholders will be Benjamin Charlton Hammond, an enthusiastic 17-year-old gamer and a veteran fan of the Call of Duty: Nazi Zombie game-mode. He has been playing since the original release in 2008 and is still playing today, over a decade later. However, due to the Nazi Zombies game-mode becoming more complex over time the changes have detoured Ben from new releases. He has started to dislike the Nazi Zombie game-mode and has stated he would like a simple game that focuses mainly on the core survival aspect.

My game would appeal to him as it is his favourite genre, wave survival. Furthermore, my game will approach his needs for simplicity in wave survival games. He will have lots of fun playing the game.

STAKEHOLDER 2: ISAAC MARK WOODWARD

My second stakeholder Isaac Mark Woodward, a passionate gamer, his love for retro style games is immeasurable. Most of the games he owns are in the retro genre. However, he thinks the genre is oversaturated with RPG games. He says RPG games have taken over and he would love to play something different.

My game would appeal to Isaac as it has a retro graphic style, with 8-bit graphics which he loves. Furthermore, it would satisfy his need for a new genre of retro style games. As my game is not an RPG game. He will also have lots of fun playing the game.

SOLVING THE PROBLEM BY COMPUTATIONAL METHODS

THINKING ABSTRACTLY AND VISULIZATION

Abstraction is the process of separating and filtering out ideas and specific details that are not needed in order to concentrate on those that are needed.
Here are some areas where abstraction would help solve the problem.

- When attacked by enemies' fatal injuries happen in real life that could injure you very badly and disable you from fighting on, some injuries can take months to heal. However, in the game, the attacks from the enemies will not disable your character, additionally, ambrosia will heal you instantly.

Why is this done?

This is done to make the game fun and playable for hours. The stakeholders would not like a game where people die quickly after a fatal wound or are incapacitated. This would only infuriate stakeholders and players as they are helpless to win or play the game.

- In real life, people run out of stamina when fighting and moving. In the game stamina will not exist, to make the game easier and simpler.

Why is this done?

It would make stakeholders happy as they can play the game for hours, having fun, not restricted by real-life conventions.

- My game will have a combo meter and the amount of Health Points displayed at the top of the screen. This is a visualization as these are not displayed in real life. This is to allow the player to adapt and understand the situation they are in.

Why is this done?
It allows the players to strategize against enemies when they can see their Health Points. For example, when they are low on Health Points they can retreat and try to fight defensively. This makes the game more intense and fun for the stakeholders and players.

- In real life battles last for very long and the soldier must endure and survive till the end. However, in the game, the player can click the pause button and pause the game and take a break to attend to some other needs.

- Additionally, there will be HUDs displayed to say if you have died or survived, this would not happen in real life.

Why is this done?
It gives signals to the player to what to do next and an incentive to keep trying and playing the game.

THINKING AHEAD

Thinking ahead is identifying the inputs and outputs before the solution is coded.

- The movement system must be prepared so the player can move the character. The right arrow key will move the character to the right and vice versa with the left arrow key. The spacebar key will make the character jump.

Why is this done?
Movement is needed for the player to play the game. Planning this gives an early overview of the movement system.

- The combat system must also be prepared. The player will perform attacks with the “a” and “s” key. The “a” key for fast attacks and the “s” key for heavy attacks, while the “d” key will be used to block incoming attacks from the enemy.

Why is this done?
Planning this gives an overview and future design insights of what the player can do to avoid and use to fight against the creatures.

- The game will also save the scores of each player, so the game must also make a database to store the records and sort them from highest to lowest, forming a leader-board table. This database must be written into a file so that it saves the data of all the players' scores. Therefore, it gives us ideas of what we need to make for score system in our game.

Why is this done?

It gives base requirements for the system to work. For example, it shows us that we need a database and sorting method/function to order the scores.

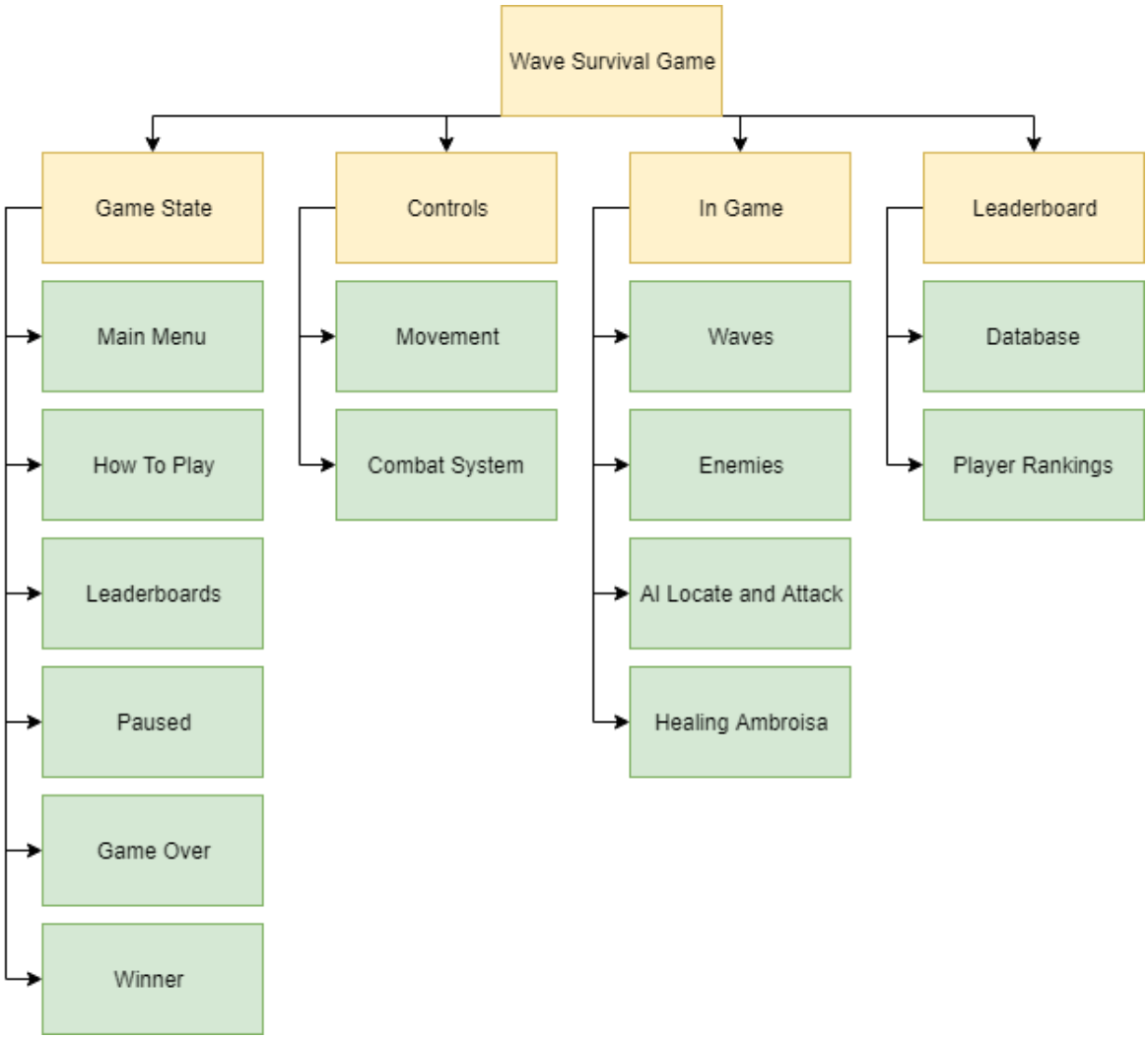
THINKING PROCEDURALLY AND DECOMPOSITION

Thinking procedurally and decomposition is breaking down your problem into sections.

Why is this done?

Thinking procedurally allows me to break the problem into systems that can be done one at a time. It makes it easier to understand and grasp the ideas and systems needed for the solution. It makes it easier for me to accomplish and build the solution.

Here is an early breakdown of the problem into 4 main sections with systems within them.



• Section 1 is the game state. The game will be in different states. For example, the menu screen will have buttons to go to other screens such as “how to play” and “leaderboards” screen. To reduce production time, you can create a subroutine that switches pages when a button is clicked, this code can be duplicated and used for each screen change by just replacing what button is clicked and what screen it changes into. The winner screen and game over the screen can be coded with If statements. For example, when the final enemy in the final wave has died the code will detect this and run the winners screen

• Section 2 is the controls, this is largely also based on if statements. For example, when “a” for fast attack is pressed, the code will detect the input and run the action. This will be added to all movement commands.

- Section 3 is the game. The wave system must more enemies each increasing round. The enemies must locate the player and attack the player till he falls. The enemies must move on their own and there is an AI element to this. Multiple scenarios must be coded using if statements and loops so that the enemies are always moving and trying to attack the player.

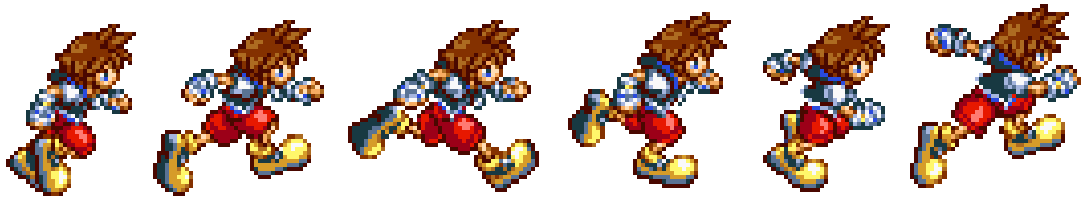
- Section 4 is data management. The game will provide a leaderboard table. A database should be created in the game files where the top 10 scores/players are stored. The game should be able to read the database and display the highest-ranking players/score in the leaderboards screen. Also, the game must be able to write to the database. If a new player scores higher than someone on the leaderboard the game should edit the leaderboard and delete the lowest record in the database.

THINKING LOGICALLY

- The menu is a large decision point. The game will allow you to select between 4 options. “Play”, “Controls”, “About”, “Leaderboards”. A decision must be made by the user to which part he would like to navigate too. While the game must loop the menu screen until a decision has been made.

- The game will also feature loops. The game must continue until the player has died or till he wins at wave number 10.

- The game will feature animations of the sprite when he is moving, standing idle and attacking. For example, when the sprite is moving the game will loop through the different version of the sprite to show movement until he stops moving



Here is an example of an animation loop.

- The game will feature AI enemies; Each AI must respond to the actions/location of the player and must locate the player and attack till he has fallen. For this loops can be used. You can loop the actions of the AI by telling it to keep looking for the player and chase him till it is in attacking distance. Afterwards, exit the loop and go into another loop that keeps attacking the player till he has died.

THINKING CONCURRENTLY

- Character animations would be solved at the same time. The program would run the player's character and the enemies character animations at the same time during the game.

- Button methods and functions can be reused instead of coding new ones by replacing the page it links too.

OVERALL SUMMARY OF COMPATIBLITY

The solution is solvable by computational methods. The solution can be separated into different sections. Each part of a game can be a different problem that stems into many problems. Problem recognition is used.

Additionally, after learning of these problems the problem can be decomposed into many other sub-problems that need to be addressed before the problem can be completed. Program decomposition is used.

In addition, problems must be broken down into their simplest form and worked from the ground up. Abstraction needs to be used to achieve this.

Many of the problems can also be solved by coding methods such as:

- Iteration
- Selection
- Arrays

Furthermore, assets of the game can come under coding methods such as:

- Classes
- Objects
- Global Variables

For this problem a machine is needed to preform lots of calculations very fast while still being responsive to inputs from the user while providing outputs. Therefore, this is a very compatible problem as computers can do all the 3 mentioned. Additionally, an approach with computational thinking also allows problem to be deconstructed into sizable chunks for solving.

RESEARCH

CALL OF DUTY BLACK OPS 2: NAZI ZOMBIES

A game that can be played across consoles and PCs. Nazi Zombies is a 3D Horde Survival game that can be played solo or with up to 4 players. In the game you must defend yourself from infinite waves of Nazi Zombies.

FEATURES

- Graphics

It has strong realistic HD and 3D graphics which immerse the player in a stunning 3D rendered world.

- Sounds

Wide variety of sounds. There are lots of environmental sounds such as the zombies, the ground, shooting weapons and the bus. Additionally, there are dialogues from each of the 4 characters for different situations. Furthermore, there are soundtracks for different moments in the game such as losing, when the next round starts and special rounds. These sounds immerse the player into the virtual world of the game and make the player feel more exposed to the game's environment.

- Multiplayer

The game can be played with other players, locally through split screen or online through the internet. This accessibility provides convivence for the player and allows them to team up with friends to fight against the horde of zombies.

- Gamemodes

The game provides different gamemodes to help when things get stale and repetitive. There are 3 gamemodes you can play.

1. Classic

Classic, is the original horde mode. When you are locked in 1 part of the map and must survive against the hordes.

2. Transit

Transit allows you to traverse the zombie area and visit different parts of the whole map through a bus.

3. Greif

Greif is zombies with 2 separate squads. The squads must outlive each other to win.

- Maps

The game has a selection of different maps to give players a new environment to survive in, it forces them to adapt and learn their new environment to survive for longer. Additionally, it makes things less repetitive, as fighting on the same map can become boring after a while.

- Changing Environment

The game allows you to access the environment and change it. It does this by allowing you to build barricades across different points of entry. Additionally, with the points you accumulate you can unlock new sections of the map to traverse.

- Points System

You can earn points by killing and damaging zombies. You can spend these points on new weapons and powerups that are accessible throughout the map or unlock sections of the map.

- Passive and Active Power Ups

There are a variety of powerups to enhance your character and weapon passively.
There are 5 power ups for your character and 1 for your weapon.

- Juggernaut
- Quick Revive
- Speed Cola
- Stamina Up
- Tomb Stone

And for weapons:

- Pack – A - Punch

Furthermore, there are other active powerups that affect the game real time. Such as

- Nuke Ka Boom
- Carpenter
- Insta-kill
- Max Ammo
- Double Points

These make survival easier and can relieve tension at later waves when the game becomes harder.

CONCLUSION

What I like about the game	Would I incorporate it in your game?	Reason
Passive and Active Power-Ups	Yes	As the game gets harder players need to have more powerful tools to fight off the stronger enemies through powering up characters/weapons passively or affecting the game in real time.
Points System	No	There are no unlockable areas in my game.
Changing Environment	No	Do not have the time/skill to incorporate this idea.
Multiple Maps	No	Not enough time to create multiple maps.
Variety of sounds	Yes	Allows for better immersion in the game for the player.
Game-modes	No	Not enough time to create multiple gamemodes
Multiplatform Game	No	Do not have the skills or the time to port it onto different platforms.
Online Multiplayer	No	Do not have the skills or the time to implement this feature.

What I don't like	Why
HD 3D Realistic Graphics	Not everyone's system will have the power to run HD 3D Realistic Graphics on their system.

FORTNITE SAVE THE WORLD

A horde survival game with cartoonish graphics and a unique building mechanic. Playable with multiple people, only through online play.

FEATURES

- Graphics

Unique and quirky graphics style that allows for even weaker systems to run the game. However, sometimes this graphics style is very appealing and allows the game to stand out in a market oversaturated with realistic graphics.

- Sounds

Variety of sounds to help immerse the player into the game. Sounds include:

-Gunshots
-Storm
-Building
-Dialogue
-Traps
-Explosions

As mentioned before, it brings the virtual world to life and exposes the player into this grand environment.

- Variety of weaponry

The player can choose a weapon of their liking. This gives the player choice and allows the player to personalize their kit for killing the hordes of enemies. Furthermore, surviving longer allows you to unlock better weapons for future survival.

- Unique Building Mechanic

Players can use Fornite's unique building mechanics to change the shape of the terrain to their favor in a fight against the hordes of enemies. Gives the player freedom and a choice of playstyle.

- Base building

Players can use the building mechanic to build their base. Essentially a safe-zone where they can choose what missions and where to venture out to next.

- Story and Missions

The player has the choice to go on different missions that tell stories. The stories allow the player to stay interested into the narrative of the game and will persuade the player to continue playing.

CONCLUSION

What I like about the game	Will you incorporate this in your game	Why
Cartoonish Graphic Style	Yes	Not everyone's system will be able to handle a HD realistic game. A cartoonish 8bit graphic style can be run by most computers today.
Story	No	The story is a vital part about games. However due to my game being a smaller project it will not include stories.
Dialogue	Yes	Increases the immersion for the player.
Variety of Weapons	Yes	Allows the player freedom and personalization of their kit used in combat.

What I don't like	Why
Building Mechanic	Some people find this mechanic very unique and interesting. Although it allows freedom for the player, it can introduce more complex controls for the player.
Bases	A horde survival game should be intense and continuous. There should not be a safe zone where the players can relax and think of strategies.

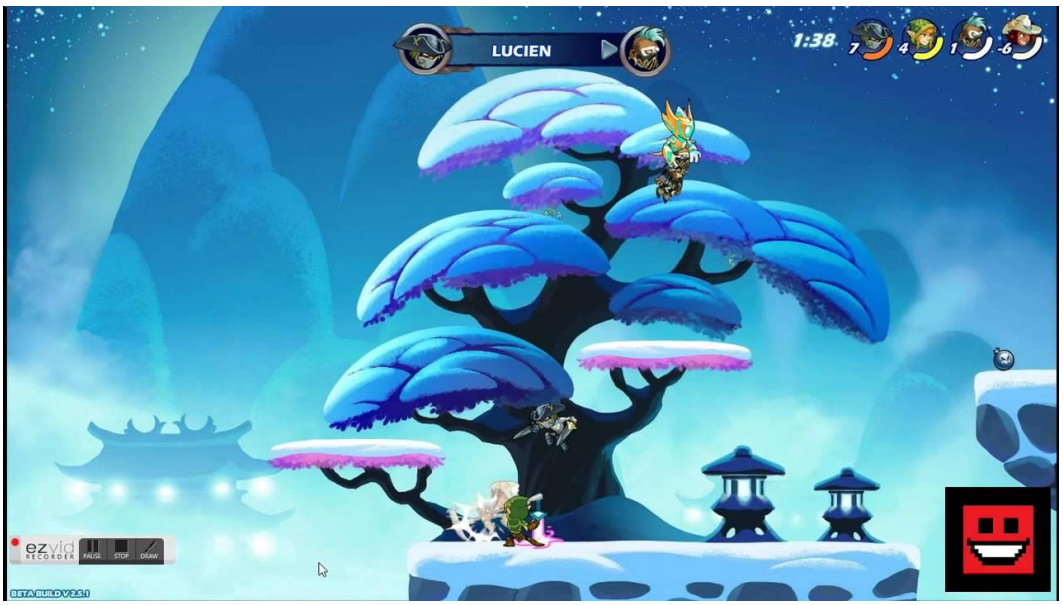
BRWALHALLA

Brawlhalla is a 2D platformer and fighting game with old 8 bit style graphics. It's a free game available on the platform Steam.

FEATURES

- Graphics

Retro graphics that my stakeholder wants, makes the game look simple but clean and pleasant. This game is playable on almost any PC. The game is not graphically demanding and can run on even system with integrated graphics cards.



- Simple control scheme

The game has a simple and easy to learn control scheme.
Using WASD for movement, C to light attack, X to heavy attack and Z to dodge.
This control scheme allows new players to join and learn quickly, while allowing skilled players to show their skill in-game by providing ways to outplay the opponent.

- Multiplayer, CO OP and Online

Allows players to fight against each other or team up with friends to beat opposing teams.

- Different Gamemodes

Allows the players not to be bored of the game. It provides different gamemode such as dodge bomb and football which are also very fun.

- Variety of characters.

There are multiple characters in the game. This allows the player to find a character with a fighting style that fits their needs. It also allows experimentation to master other fighting styles and characters.

- Competitive Ranked

The players can separate themselves from the noobs to the pros. The game offers a ranked system, to test your skills against other players in a highly competitive match. The ranks include:

- Tin
- Bronze
- Silver
- Gold
- Platinum
- Diamond

This allows players to be rewarded and adds a sense of accomplishment as the player gets better at the game.

CONCLUSION

What I like about the game?	Will I incorporate it into the game?	Why?
Retro Graphics	Yes	Allows weaker systems to run the game and make it playable on a large percentage of market devices.
Easy control scheme	Yes	Allows new players to pick up the game easily and join in on players who have played before.
Online and Local Multiplayer	No	Instead of just playing against AI the player can fight against other players. Do not have the skills and time to create this.
Ranked System	No	A system that allows competitive play against other players, it separates the good from the bad. It encourages people to play longer and rewards them with ranks. Do not have the skill to implement this feature and is unnecessary for a small scale offline project.

INTERVIEW WITH STAKEHOLDERS

PROPOSED QUESTIONS

- Would you play a hyper realistic game at a lower performance level or a cartoonish game at a higher performance level?
- How fond are you of 8-bit graphics on a scale of 1 to 10?
- What is your ideal game's graphic type?
- Is graphics the most important aspect of a game?
- How simple should the game's controls be?
- What keys would you prefer for movement?
- Should combat system be easy or hard?
- Should my game be easy or hard?
- Is sound important for a game?
- Should there be environmental sound?
- Should there be dialogue included in a game?
- Should there be a soundtrack included in the game?
- Would you like different weapons to choose from?
- What other features would you like for the game to have?

INTERVIEW WITH ISAAC WOODWARD

Would you play a hyper realistic game at a lower performance level or a cartoonish game at a higher performance level?

I would rather play a cartoonish game at a higher performance level. More FPS means that the game runs more smoothly and is easier on the eye. Less FPS can cause issues and result in a negative experience.

How fond are you of 8-bit graphics on a scale of 1 to 10?

10 I love 8-bit graphics.

What is your ideal game's graphic type?

8 bit obviously.

Is graphics the most important aspect of a game?

No, I would rather have a good story and good mechanics over graphics. Graphics are important but not a true fundamental aspect for a game.

How simple should the game's controls be?
Simple enough to understand easily but should allow for better combat after experience.

What keys would you prefer for movement?
WASD come on dude. The industry standard.

Should combat system be easy or hard?
Easy, it should be easy to control to allow players to jump into the game straight away. But it should also allow for more complex combat through more experience and exposure.

Should my game be easy or hard?
Your game is aimed at teenagers it should be pretty hard and challenging.

Is sound important for a game?
Yes, you cannot have a dull and muted game.

Should there be environmental sound?
Yes, it helps make the game feel more immersive.

Should there be dialogue included in a game?
Only for the important bits.

Should there be a soundtrack included in the game?
Yes, it helps convey the emotions and feelings of certain levels of games.

Would you like different weapons to choose from?
Yes, it helps keep the game interesting and it allows different playstyles. Some people like quick fighters while others like slower fighters that do more damage.

What other features would you like for the game to have?
Since the game is supposed to be challenging, there should be a way to separate the skill of players, to see who is better. A leader-board table to show which player is the best.

INTERVIEW WITH BENJAMIN CHARTLON HAMMOND

Would you play a hyper realistic game at a lower performance level or a cartoonish game at a higher performance level?

I would play the cartoonish game with higher performance level. It hurts my eyes to play games at lower frame rates.

How fond are you of 8-bit graphics on a scale of 1 to 10?

6, not a big fan but I don't mind them.

What is your ideal game's graphic type?

Realistic, I play a lot of games on PC and I love realistic games.

Is graphics the most important aspect of a game?

It's somewhat important but it's not the most important for me it would be the mechanics and combat system.

How simple should the game's controls be?

Simple

What keys would you prefer for movement?

WASD like every game ever.

Should combat system be easy or hard?

Easy. A hard control scheme would just make me angry.

Should my game be easy or hard?

The game should be challenging. So yea hard.

Is sound important for a game?

Yes, who would want a silent game? It's horrible.

Should there be environmental sound?

Yes. Makes the game feel more real and immerses me into the world.

Should there be dialogue included in a game?

Yes, to see the feelings portrayed by the character.

Should there be a soundtrack included in the game?
Yes. Adds to the emotion of the game.

Would you like different weapons to choose from?
Yes, it stops the game being boring and adds more choice to the game.

What other features would you like for the game to have?
Some sword sounds, I don't want a fighting game without sword sounds, it would be weird seeing my character fighting with swords but not hear the swords

SUTIABLE APPROACHES BASED ON RESEARCH AND INTERVIEW

Sutiable Approach/Features	Why
Passive and Active Power Ups	As the game gets harder players need to have more powerful tools to fight off the stronger enemies through powering up characters/weapons passively or affecting the game in real time. This will make the solution more fun and appealing for stakeholders and players. It would create situations where the player will have to strategize depending on the situation, increase the difficulty of the game and meet the needs for high difficulty of the stakeholders.
Variety of sounds	Allows for better immersion of the game and allows the stakeholders to have more fun and feel excited to play the game.
Cartoonish Graphic Style	Not everyone's system will be able to handle a HD realistic game. A cartoonish 8bit graphic style can be run by most computers today. Stakeholders also would like a game that does not revolve around graphics and this would meet their needs.
Dialogue	Increases the immersion for the player/stakeholder. Makes the game more enjoyable.
Variety of Weapons	Allows the player freedom and personalization of their kit used in combat. Therefore, saving the stakeholders and players from boredom after playing the same weapon.

Retro Graphics	Allows weaker systems to run the game and make it playable on a large percentage of market devices.
Easy control scheme	Allows new players to pick up the game easily and join in on players who have played before.

FEATURES OF THE PROPOSED SOLUTION

Menu Screen.

The menu screen will have 4 options.

1. Play
2. How to Play
3. Leaderboards

Why?

- The menu should be there to allow players to navigate through information before the game. For example, to look at the controls of the game before they play, or to look at current records in the leaderboard section.

Simple control scheme

The control scheme will be WASD for movement. J for fast attacks, K for heavy attacks and L for blocking. I have gone with WASD for movement as my interviewees have said that nearly all to most games have WASD as their movement controls as it has become an *'industry standard'*. I have gone with a simple combat controls as interviewees have said that complex controls can frustrate and irritate people. It is better for it to be easy so new players can jump right in to the game and familiarize themselves of the controls.

Why?

- Isaac and Ben both want something that is easy to control and play, as the controls can define is the game is fun or not. As apparently according to Ben harder control scheme will cause players to feel angry. While Isaac wants a simple control scheme because it is easier to pick up and play.
- Complex controls can frustrate and irritate people. It is better for it to be easy so new players can jump right in to the game and familiarize themselves of the controls.
- This has been implemented and has been very successful in completed solution: Brawlhalla and is well received by the players.

1 Map

Why?

The game will be set on 1 map. There will be no additional maps due to the game being a small-scale project.

Challenging and Hard

Why?

Both stakeholders have wished for the game to be harder and challenging. Therefore to meet their needs, instead of making an endless number of waves which are normally easy to fend off against. There will be 10 waves with extremely high difficulty spikes. The game will be challenging and hard, every wave will be a fight till the death. The game will be challenging and hard as its target audience are teenagers who have previously played games before. Additionally, the increased difficulty will create an intense situation. It will generate intense emotions to be displayed by the player and will create a sense of reward, pride and satisfaction for completing the game.

8-Bit Retro Graphics

The game's graphic style will be 8-bit cartoonish graphics.

Why?

This is because it will appeal to a stakeholder and will be easier to run on most systems owned by teens. As not all teens own a device capable of running a 3D game at high frame rates. Also, another main factor is because the game is a small-scale project. I do not have the time or skills to create and implement a 3D models, 3D map and game.

Passive and Active Power-Ups

The game will have multiple passive and active power-ups to help the player beat levels.

Why?

- This is included to help players beat levels and add more variety into the combat of the game. Since the game is going to be challenging and hard players require additional tools to survive.

Point System

As the player plays the game he will be awarded points based on his performance of the game. For example, successful hits and blocks will awards the player points, while getting hit will minus points from his total score. Additionally, more points will be awarded through combo meters, where longer and successful combos are rewarded more than single hits. This will be added to separate the good players from the bad. Players can earn recognition by earning more points to secure a place in the Top 10 leaderboards of the game.

Why?

- Isaac has opted to have a point system, to differentiate between the good and the bad players. To meet his wish I will add a point and leaderboard system.

- Players can earn recognition by earning more points to secure a place in the Top 10 leaderboards of the game.

AI enemies/Waves

The game will need AI enemies, enemies must be coded to attack and kill the player.
There will be 10 waves, each wave the enemies will increase in strength, speed and defense.
Why?

- Without the AI the solution will not work and would not be functional. The player will have nothing to fight.
- They will increase in strength, defense, speed and numbers to meet the needs for a high difficulty game.

Variety of weapons. (3 Weapons)

Player can pick from 3 weapons to use and fight enemies with.
Why?

- This will be added to give the player freedom and personalization of his kit for combat. This has also been already successfully implemented in other successful finished solutions like Fornite Save The World.
- It will keep the players and stakeholders interested and they will continue using the solution.

Soundtrack

Some parts of the game will feature soundtracks, such as playing the game and menu screen.

Why?
Interviewees have claimed that a game without sound can be very dull. Soundtracks will be included to convey emotions and increase immersion of the game. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World and Brawlhalla.

Environmental Sound

Environmental sound will be added. These sounds will play when a player attacks an enemy. Sounds such as sword clashing and stabbing will be added.

Why?
This will be added to increase the immersion the player feels. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World and Brawlhalla

Dialogue

Dialogue will be featured in the game. The players' character should talk and scream/shout during fights, while the enemies let out cries/shouts of their own when they die or attack.

Why?

- Dialogue will be added to increase immersion for the player and make the player feel like it's a real battle.

Leaderboards

Why?
As Isaac has requested, the leader board will be the place where good players are separated from the bad. Player will be able to see where they rank against other players who have also played the game. Top 10 players will be saved in a database and shown in a separate menu screen.
The database must write, read and save the data, so that all top 10 records are scored. It must also delete records when new and better records are set by other players.

LIMITATIONS OF THE SOLUTION

Limitation	Why?
No multiplayer feature	Other complete and successful solutions have multiplayer included in the game therefore the game would not be attracting than the complete solutions.
No cross/platform.	Not everyone owns a laptop or a PC to play on. People will own different devices such as Xboxes and PS4s.
Copyrighted content being used in the game.	Do not have the time or skills to create all the assets in the game such as art and sounds.
Old graphics style	Not everyone will enjoy 8-bit graphics with all the advancements in graphical technology and realistic games.

No story	The solution may not be attractive as the other complete ones as it has no story. It is just a hack n slash game.
Stale and continuous environment.	There is only 1 map, therefore the background could get boring. There is also no environmental changes to the map during the game. Will cause players to be bored of the map.
Simple game.	It is a simple game. There are not as many features as other complete solutions and is missing any unique feature that makes the solution appealing.

SYSTEM REQUIREMENTS

Recommended
Operating System: Windows Vista, 7, 8 or 10
Processor: Clock speed 2GHz or more
Memory: 3GB Ram
Graphics Card: Card with 512MB of Video Ram
Storage: 500MB

Minimum
Operating System: Windows Vista, 7, 8 or 10
Processor: Clock speed 2GHz or more
Memory: 2GB Ram
Graphics Card: Card with 128MB of Video Ram
Storage: 500MB

JUSTIFICATION OF SYSTEM REQUIREMENTS

The system requirements to run the game is very low. This is because the solution/game is very simple and does not have calculations on that computer will struggle on to render or computer. The game is designed to run on a large percentage of the market PCs.

JUSTIFICATION FOR USING THE SOFTWARE: UNITY GAME ENGINE

The game engine that will be used, will be Unity. I will be using Unity is because it is more prevalent across the web and already has been used to create a multiple of games from 2D games such as Dead Cells to 3D giants, such as PUBG and Fortnite. Additionally, it is easy to use. If there is trouble there are lots of tutorials and guides to help learn how to use Unity. Furthermore, it allows an easy port to other platforms and operating systems, if I were to transfer the game to another platform such as Apple's Macintosh.

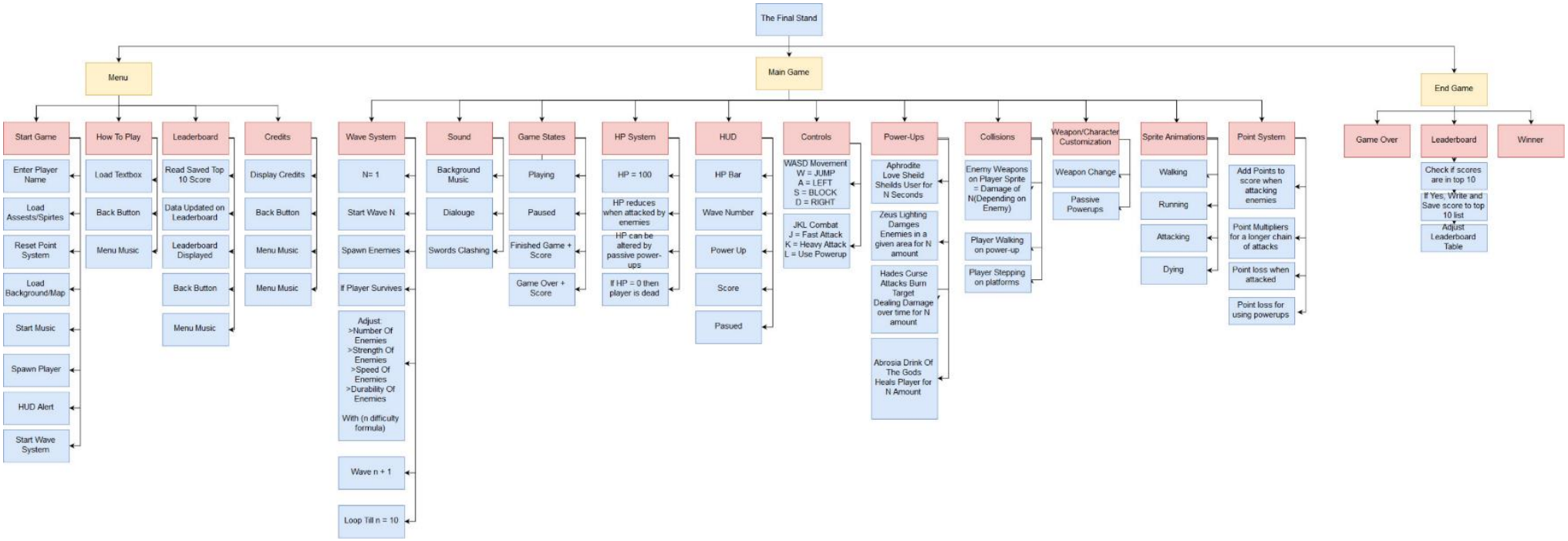
MEASURABLE SUCCESS CRITERA

Success Criteria	Justification
Simple control scheme. • WASD for movment. • J for fast attacks. • K for heavy attacks. • D for blocks.	Interviewees have said that complex controls can frustrate and irritate people. It is better for it to be easy so new players can jump right in to the game and familiarize themselves of the controls.
Menu with 3 options 1. Play 2. How to Play 3. Leaderboards	To allow players to navigate through information before the game. For example, to look at the controls of the game before they play, or to look at current records in the leaderboard section.
Challenging and Hard • Only a some players should get scores above 100.	The game will be challenging and hard as its target audience are teenagers who have previously played games before. Additionally, the increased difficulty will create an intense situation. It will generate intense emotions to be displayed by the player and will create a sense of reward, pride and satisfaction for completing the game.
8-Bit Cartoonish Graphics	Allows low end system to run the game.
2 Passive and Active Powerups	Allows the player new tools to fight the enemies. Makes the game interesting for players.

Point System/Combo Meters • Only some players should manage to get over 100 points	Give a chance for players to evaluate their performance in game to allow room for improvement. Also included to create a leader boards table.
2 Different type of enemies. • Melee enemies • Ranged enemies	To introduce variation of the enemies the player fights.
Variety of weapons • 3 weapons to select from before beginning the game.	This will be added to give the player freedom and personalization of his kit for combat. This has also been already successfully implemented in other successful finished solutions like Fornite Save The World.
Soundtrack 2 sound track pieces. • 1st track for the menu screen. • 2nd When the player is playing the game.	Interviewees have claimed that a game without sound can be very dull. Soundtracks will be included to convey emotions and increase immersion of the game. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World.
Environmental Sound • 2 tracks of dialogue. • 2 tracks of swords clashing. • 1 track of arrow firing • 1 track played when the player dies.	This will be added to increase the immersion the player feels. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World.
AI enemies that follow the player around the map and a changeable speed.	It is a fighting game; the player must be attacked by AI controlled enemies. Speed must be changeable to increase the difficulty of the game.
Leaderboard showing the Top 10 players and scores. Stored outside the game so it can be reloaded when the game is closed and reopened.	Isaac wants the game to show who the best is and split people into different skill levels. It increases the competitive nature of the game and motivates people to try beat the top scores on the leaderboards.
Wave System	System that spawns waves of enemies and stops after 10.

B. DESIGN

SYSTEMS DIAGRAM



MOVEMENT

WHY?

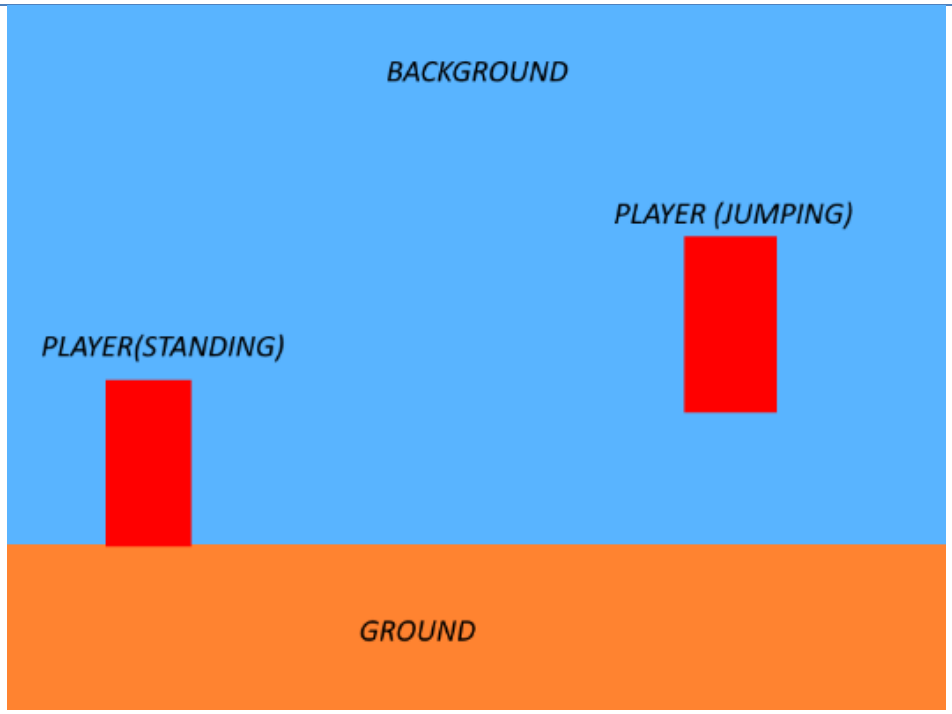
Coding the players functions will be first as it is the core of the game. Without the player movement, the player will be unable to move and fight, essentially unable to play the game. This was first over the menu screen because the menu screen is easily codable compared to the player functions and movement.

JUMPING AND GRAVITY

WHY?

Jumping is needed in this game because if provides a method of traversing the game and evasion maneuverers from enemies. It will be designed first as it the most difficult to code out of all the player functions.

UI



Justification of UI

- There will be a background to fill the space and make the game feel more immersive, as a blank screen can make the game look dull. Makes the game look more appealing to stakeholders and players attracting them to play the game.
- The ground is needed to hold the characters and events of the game. Otherwise the game/solution would be unplayable and incomplete.

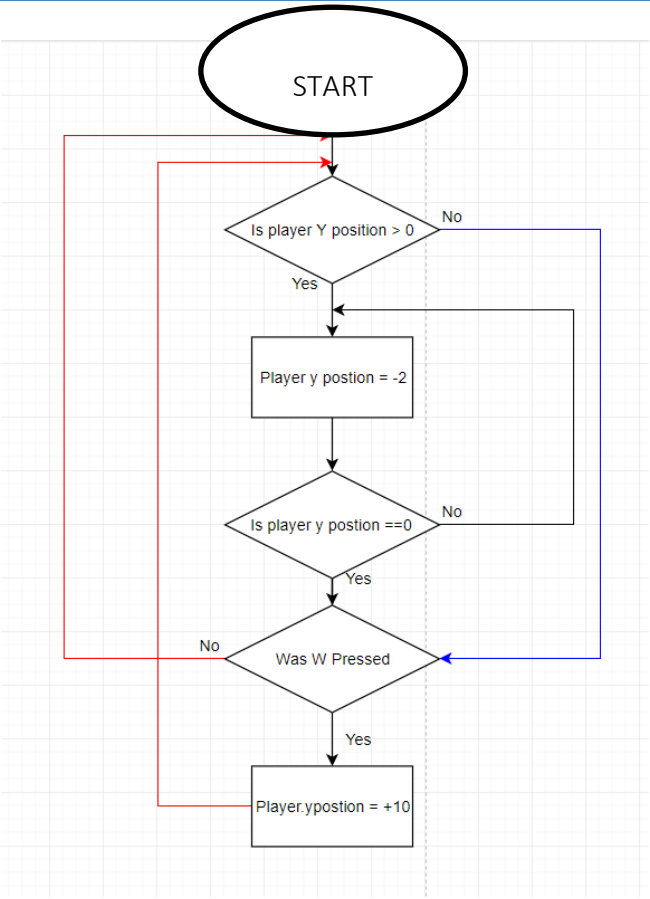
PSEUDOCODE

```
If player y position > 0
 {
    Do until (y==0)
  {
      Player.y position = -2
  }
}
If W key is pressed
{
   Player.y.position = +10
}
```

PSEUDOCODE EXPLAINED

If the player is already in the air (y position > 0) then bring the player down at a constant rate (gravity) (player.y postion =-2). If the player has pressed the W key, make him jump by moving the player up the y axis by 10.

FLOWCHART



KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	YPosition	Integer	The position of where the player is on the y axis	Variable is accessed to allow the player to jump by moving them up the y axis and bring them down by moving them down the y axis	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

Only W will be tested as it is the only input required to test the code. Pressing the button will test if the jump script as worked. The player should move up the y axis and fall back down as 'gravity' brings it down.

TEST DATA FOR BETA TESTING

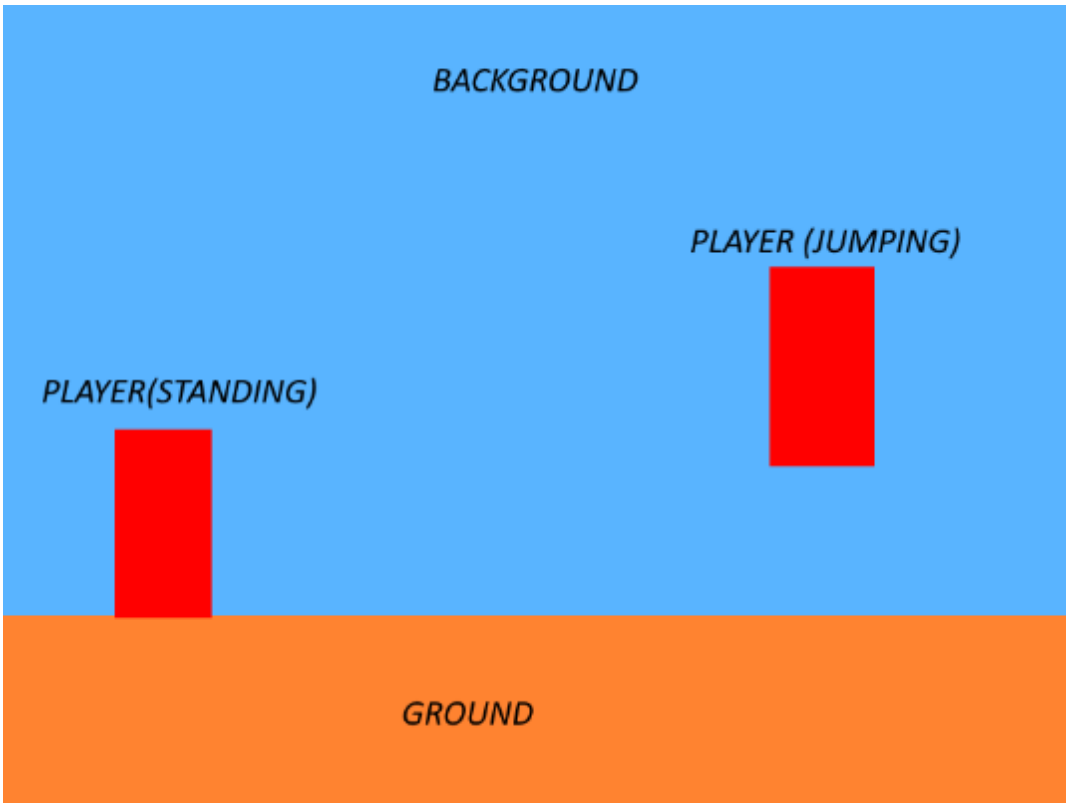
Test Identification	Test	Input	Expected Outcome
Ju1	Gravity	W to jump, to allow for gravity to work.	The character should complete its jump and fall down back to the stage.
Ju2	Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.

JUMPING REDESIGNED

WHY?

Due to the problems mentioned in the development. I have had to redesign the jumping and gravity section. This was due to me designing the game in the syntax of C# Visual Basic, which does not match the syntax of C# Unity. Additionally, gravity does not need to be coded as it is a default asset included in Unity.

UI



Justification of UI

- There will be a background to fill the space and make the game feel more immersive, as a blank screen can make the game look dull. Makes the game look more appealing to stakeholders and players attracting them to play the game.
- The ground is needed to hold the characters and events of the game. Otherwise the game/solution would be unplayable and incomplete.

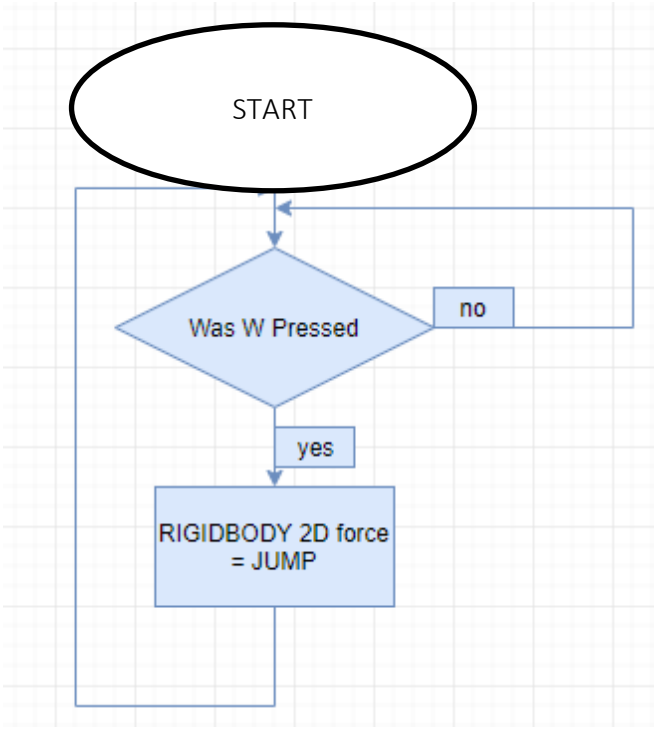
PSEUDOCODE

```
int Jump = 10;

Get Rigidbody2D
If Input = W
{
  Rigidbody2D force Y axis = jump
}
```

PSEUDOCODE EXPLAINED

In Unity it is compulsory to call the physics component, ***RIGIDBODY 2D***, to make any game-object do any actions and movement in Unity. So, the code must call the ***RIGIDBODY2D***, after it calls the ***RIGIDBODY2D*** is tells the object to jump by using the force of the variable ***Jump*** (10) along the Y axis. Therefore, making the player jump.

FLOWCHART



KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	JumpForce	Integer	The force of the jump, the higher the force the higher the player will jump.	Variable that will determine how high the player can jump. It is needed to test the code for this section	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Valid
S	Valid
D	Valid
J	Valid
K	Valid
L	Valid
P	Valid

Justification

Test data has been changed since the previous design. All buttons are now valid in testing the code as there is a test that requires multiple inputs from the player.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
Ju1	Jumping	W to jump, to see if the player can actually jump.	The character should jump when W is pressed.
Ju2	Falling down	W to jump to see if the player falls down after jumping.	The character should fall back down after jumping.
Ju3	Sensible Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.
Ju4	Pressing other buttons	W to jump and any other buttons.	Jump should still occur regardless the fact that another button was pressed.

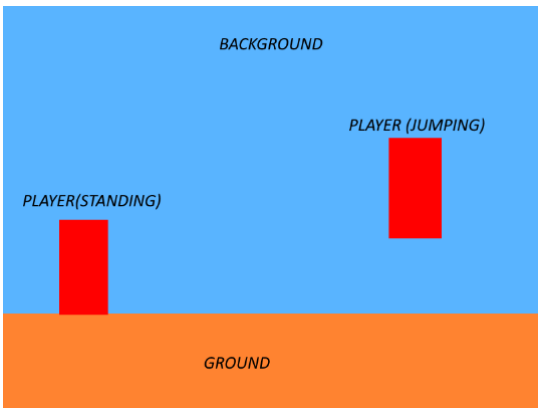
RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

DOUBLE JUMP (ACCOMMODATING STAKEHOLDER FEEDBACK)

WHY?

After review of the jumping system this was recommended to improve the existing solution.

UI



Justification of UI

- The ground is now used for more than just holding the player and enemies. It acts as a signal to tell the game if double jump is available for the player.

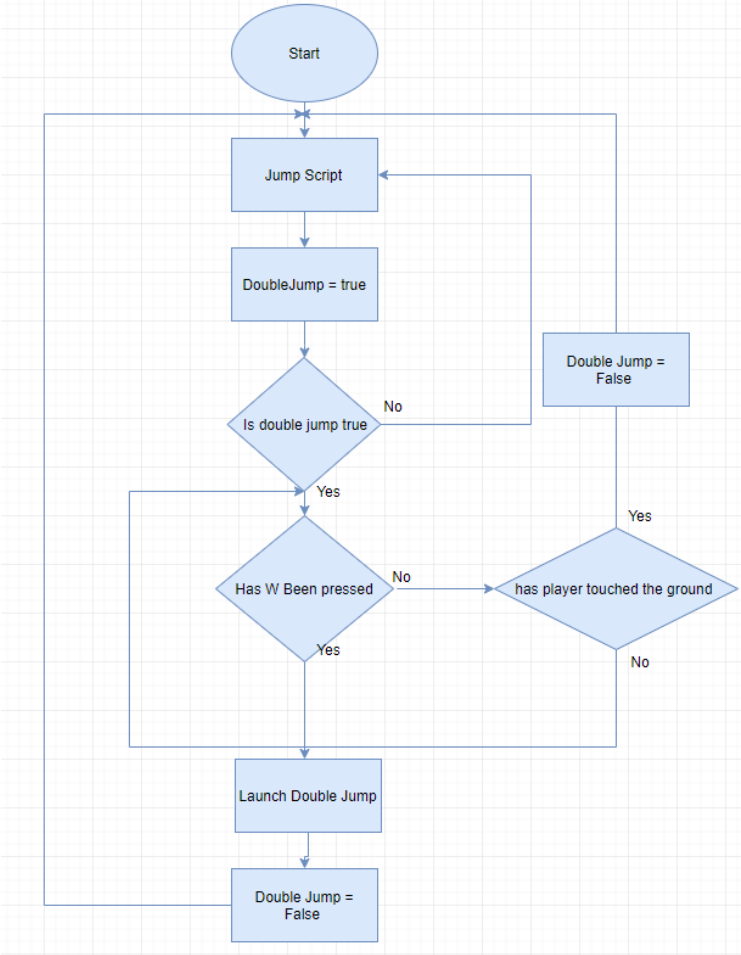
PSEUDOCODE

```
Doublejump bool = false
Jump script
  Double jump = true

If double jump = true
        If W is pressed
                Jump
                Double jump = false
        End If
If touching ground
        Doublejump = false
End If
End
```

PSEUDOCODE EXPLAINED

There will be a Boolean that shows if double jump is available. When the player jumps once, the double jump will be available. If he presses W when double jump is available, he can jump again. After he has double jumped the Boolean will be false. Therefore, allowing the player to jump only twice and fall back down.

FLOWCHART


Start
Jump Script
DoubleJump = true
Is double jump true
No
Yes
Has W Been pressed
No
Yes
has player touched the ground
Yes
No
Double Jump = False
Launch Double Jump
Double Jump = False

KEY VAIRBALES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	JumpForce	Integer	The force of the jump, the higher the force the higher the player will jump.	Variable that will determine how high the player can jump. It is needed to test the code for this section	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	DoubleJump	Boolean	Boolean that will determine if the player can doublejump or not.	To improve my solution stakeholders suggested that I implement double jump. This boolean is needed to signal the game so that it can allow the player to double jump.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- W needs to be pressed twice. It is needed to test the double jump code and make sure the player can use it.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump.	Player should be able to jump once then jump again while still in the air.
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being tested as this was an issue with the first original jump design.	Player should only be allowed to double jump, there should be no additional jumps until he falls back on the ground.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

DOUBLE JUMP REDESIGNED (ACCOMODATING STAKEHOLDER NEEDS)

WHY?

I was unable to find a solution with the previous design plan and therefore scrapped the idea and restarted.

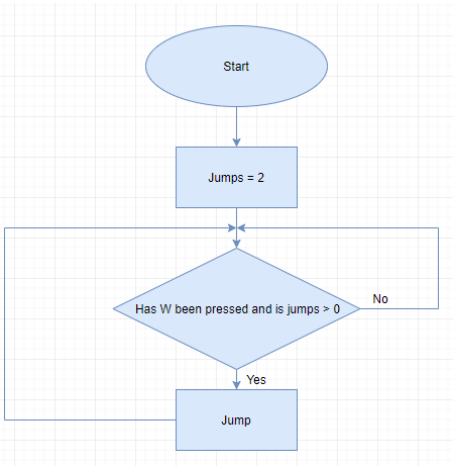
PSEUDOCODE

```
Int jumps = 2
if W is pressed & jumps > 0
        Jump script
        jumps = jumps – 1
end if

 If player collides with ground
        Jumps = 2
End if
```

PSEUDOCODE EXPLAINED

The player will have a set number of jumps. ***Jumps*** variable is set to 2. So, the player can jump 2 times, a double jump. Every time the player jumps, the number of jumps he has is decremented. After he runs out of ***Jumps*** he cannot jump anymore, he has to fall back onto the ground where his number of ***Jumps*** are reset. 0

FLOWCHART



KEY VARIABLES, JUSTIFIACTION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	JumpForce	Integer	The force of the jump, the higher the force the higher the player will jump.	Variable that will determine how high the player can jump. It is needed to test the code for this section	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	Jumps	Integer	This variable determines how many extra jumps the player has. If the jumps reaches 0 then the player cannot jump anymore.	To improve my solution stakeholders suggested that I implement double jump. It is needed to set up the double jump. The value of the variable will be 2. So, the player can jump 2 times, creating a double jump.	Input W will only work If the value of the jump variable is not equal to 0. If it is 0 the W input will not work, and the player will not jump.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- Again, only W will be a valid input to test the code, as the only input the code accepts is W. It is needed to see if the player can double jump. Also needed for beta testing, where W is pressed multiple times.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump.	Player should be able to jump once then jump again while still in the air.
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being tested as this was an issue with the first original jump design.	Player should only be allowed to double jump, there should be no additional jumps until he falls back on the ground.

MOVING LEFT AND RIGHT

WHY?

Again, we are still focusing on the players core movement first. Logically the next step after jumping would be movement of the character. Without the ability to move left and right the solution would be incomplete, and the game would be unplayable.

We are only coding moving left and right as they are the only directions. The game is a 2D platformer, therefore the only direction you can move towards is left and right.

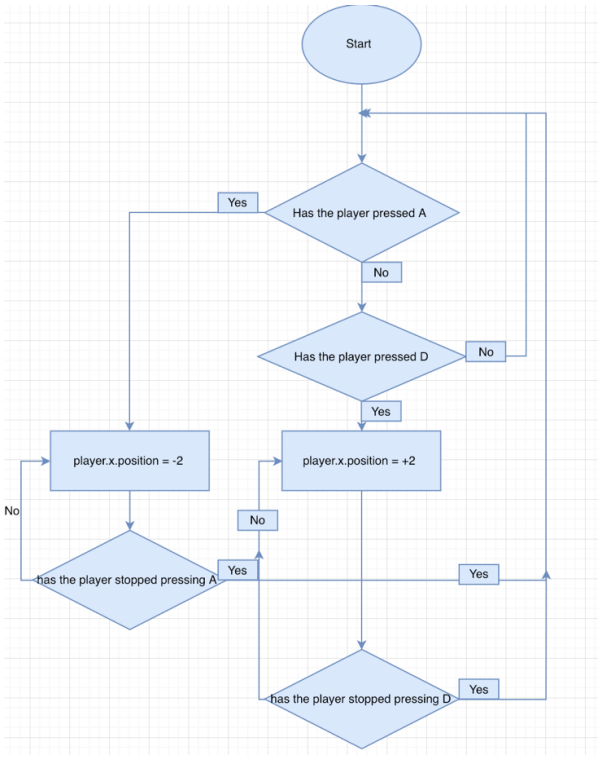
PSEUDOCODE

```
If keypress = A
        Player.xposition = -movementspeed
End If

If keypress = D
        Player.xpositon = +movementspeed
End If
```

PSEUDOCODE EXPLAINED

When the player presses the A key the X axis position value of the player will be decreased. Therefore, the player will move left. It is vice versa for the "D" key however, you are increasing the X axis position value and moving towards the right.

FLOWCHART



KEY VARIABLES, JUSTIFCATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	MoveForce	Integer	An integer that will decide how fast the player can move left and right. The higher the number the faster the movement.	Variable to test if movement commands work. Also used to adjust the speed of movements of the player.	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	X Position	Integer	The integer that holds the players X Position value.	This value will change depending on what button has been pressed. Changing this value results in the player moving left and right.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Valid
S	Invalid
D	Valid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- W is needed to test LR3. It is to check the robustness of the solution by trying to break the script, it will be done by trying to jump and move at the same time.
- A and D are needed to test the code for moving left and right. Without them the code will not run, and we cannot test if the player will move across the X axis.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
LR1	Can the player move to the left?	Holding the button, A	Player should move to the left continuously until button A is let go
LR2	Can the player move to the right?	Holding the button, D	Player should move to the left continuously until button D is let go
LR3	Can the player move right/left and jump at the same time?	Holding the button, D/A and pressing W.	Player should jump towards the right/left and after continue moving right/left.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

DASHING (ACCOMODATING STAKEHOLDER FEEDBACK)

WHY?

Stakeholders Isaac and Ben have requested a dash to improve the movement system. I also believe that a dash would be a great implementation to the game to improve its complexity while keeping the game simple for players.

PSEUDOCODE

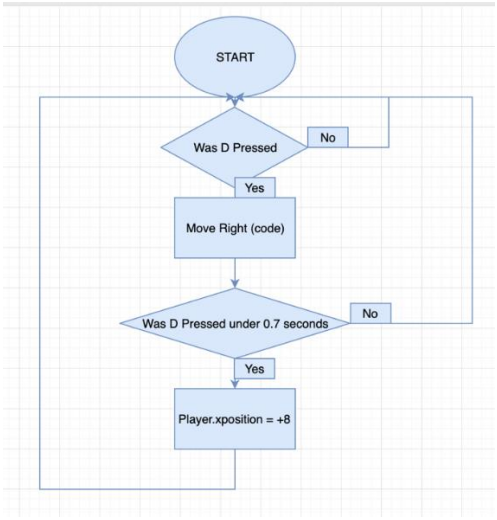
```
Var taptime
If (D is pressed)
        (move right code)
        If (D is pressed again & taptime < 0.6)
                Player.xposition = +8
        End if
End If

If (A is pressed)
        (move left code)
        If (A is pressed again & taptime < 0.6)
                Player.xposition = -8
        End if
End If
```

PSEUDOCODE EXPLAINED

When D or A is pressed it will launch the movement code, however if one of the buttons is pressed again under 0.6 seconds it will launch a dash by warping the player's position on the x axis forwards or backwards depending on the side the player dashed towards.

FLOWCHART

Since dashing to the left and the right are the same expect the position change on the x axis, the flowchart will only show a dash to the right.



KEY VARIABLES, JUSTIFICATION AND VALIDATION

Test Data	Type
W	Valid
A	Valid
S	Invalid
D	Valid
J	Invalid
K	Invalid
L	Invalid
P	Invalid
Double Tap D	Valid
Double Tap A	Valid

Justification

- W is needed to test for a robustness of the solution. A and D are also tested to test for the movement towards the left or right.
- Double tap D and A are needed to test the dash.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Method	Name	Data Type	Description	Justification	Validation
Variable	doubletap	Boolean	Boolean that will determine if the dash is available for the player to use after moving.	It acts as a requirement. The Boolean must be true to allow the player to dash.	Input must be a double tap D or A under 0.6 seconds The Boolean must be true, and the input must be correct for the dash to launch.
Variable	X Position	Integer	The integer that holds the players X Position value.	This value will change depending on what button has been pressed. Changing this value results in the player moving left and right.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

COMBAT

WHY?

Combat is needed! Without it the game is playable but boring and never winnable. It would detour stakeholders and other system that link with combat will never be used.

ATTACKING

WHY?

Allows the players to fight back against creatures and makes the game enjoyable. It is a core feature that is needed to create a horde survival game. How can you survive without attacking?

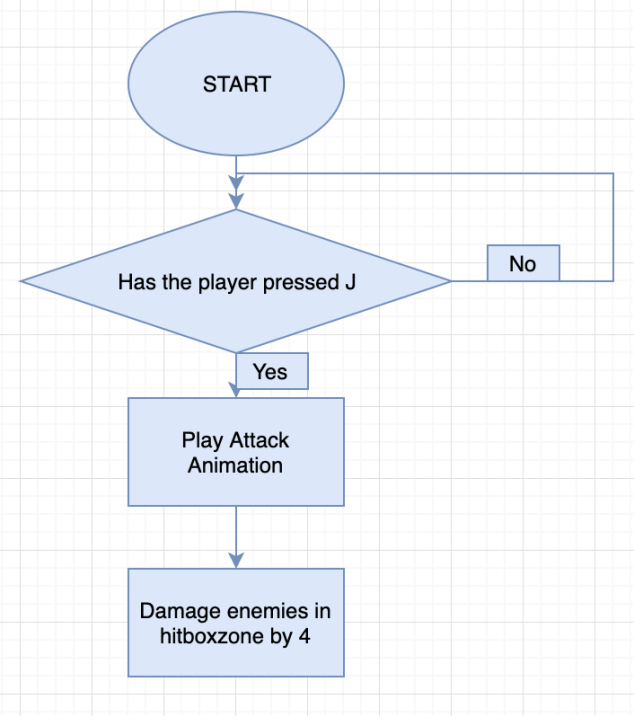
PSEUDOCODE

```
Damage = 4

        Create zone = hitboxzone
                hitboxzone shape = circle
                hitboxzone visibility = none
                zone diameter = 2
                zone position = +5 of player's x position & player's y position/2
        end zone
If J is pressed
    Launch attack animation
        Damage enemies in hitboxzone by damage
End If
```

PSEUDOCODE EXPLAINED

Creates a zone in front of the sprite. The zone is invisible to the player, when the player presses J the animation for an attack will play and the enemies inside this invisible zone will take damage.

FLOWCHART



TEST FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Valid
S	Invalid
D	Valid
J	Valid
K	Invalid
L	Invalid
P	Invalid

Justification

- J is needed to test is the attack will launch. Without it we cannot check if the attack code works.
- W, A, D is needed to test if the player can attack and jump or move at the same time. This is to test the robustness of the solution.

KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	damage	Integer	Integer that will determine how much damage the player does to the creatures.	It is needed to survive the waves of enemies and play the game. As damage is needed to kill enemies to proceed onto the next wave.	No validation. If the input is wrong or nothing, nothing will happen.
Variable	zonediamter	Integer	Integer that determines how big the diameter of the hitbox is.	Creates a zone large enough, where enemies enter to take damage.	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	zoneposition	Integer	Integer that determines how far the hitbox is in front of the player's character.	Needed to create an attack range of the player. Determine how far away the player can stand and still attack enemies.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

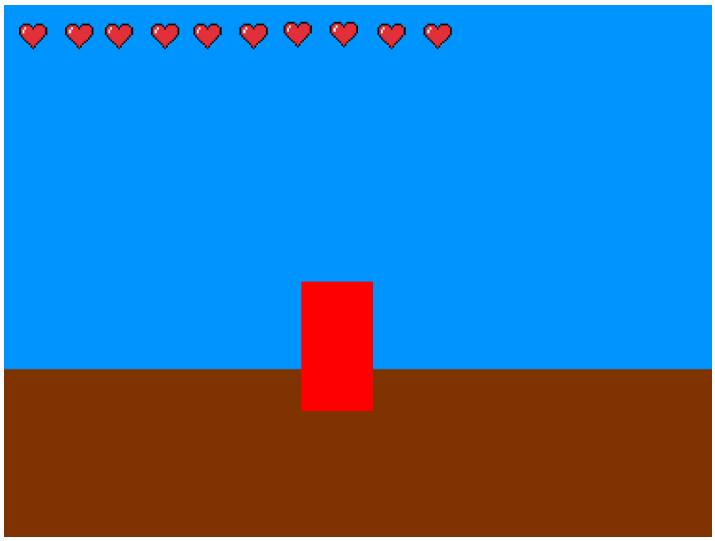
Test Identification	Test	Input	Expected Outcome
ATK1	Does pressing J launch an attack?	J	Pressing J should launch an attack.
ATK2	Does the attack do damage?	J	Player should do damage to enemies.
ATK3	Can the player move or jump and attack at the same time?	Pressing J while pressing A, D or W	Player should be able to move and attack or jump and attack.

HEALTH SYSTEM

WHY?

It is the system that needs to be coded to complete the player's core functionality. Therefore, it is being designed next. The health system is needed because without the game the player can survive with ease. Without a health system it would defeat the purpose of the game, and it would remove the fun and intensity that the game aims to provide.

UI



UI EXPLAINED

- The hearts will be positioned at the top left of the screen. This is done so that the hearts will not block the player's view of the game, also it allows the player to always keep track of their health through their peripheral vision.
- The hearts will be in the 8bit style to match the cartoonish and 8bit graphical style of the game.
- When the player is damaged the heart will blank out but still be on the UI, this is to alert the player of how much health they have lost.

PSEUDOCODE

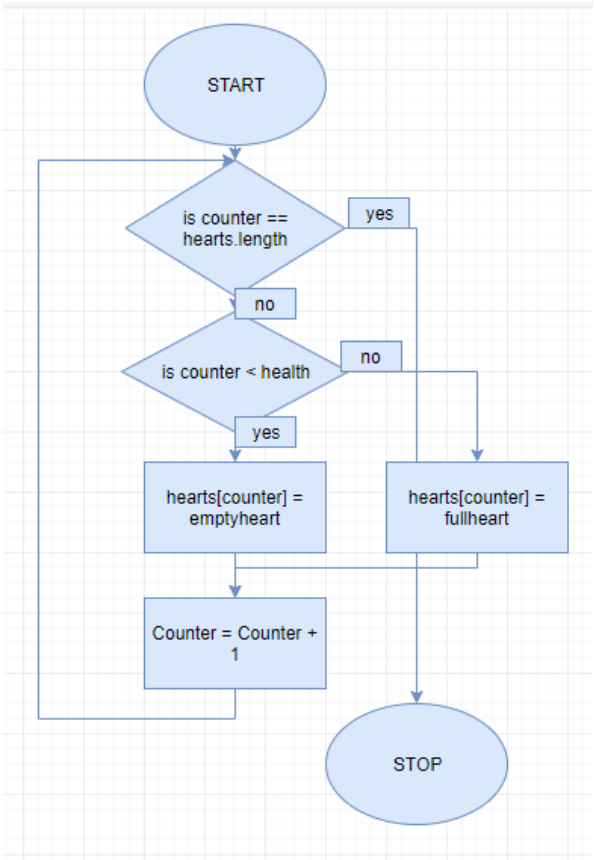
```
Health = 10
New images [ ] = hearts
New images [ ] = emptyhearts

For (int counter = 0) & (counter < hearts.length) & (counter = counter + 1)
        If counter < health
                Hearts[counter] = displayonui
        Else
                Hearts[counter] = emptyheart
        End If
```

PSEUDOCODE EXPLAINED

The code will loop until counter is equal to the number of hearts in the array. If the counter is less than the health value of the player, that means the player has more health remaining, therefore it displays a full heart on the UI. Otherwise, the player will have less health remaining therefore it displays a empty heart so show that the player has loss health.

The code will end when counter reaches the array length. However, it will begin one more and check every frame. Due to the Void Update() method on unity. The code will be run every new frame of the game.

FLOWCHART



TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Invalid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

No input needed as the enemy AI will follow the player and attack the player. When this happens you can check the UI to see if hearts start becoming empty.

KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	counter	Integer	Integer that is the core of the loop.	Without this variable the For loop would not work. Therefore, the code nested in the for loop would never run. Resulting in the health UI and system to never work.	No validation. If the input is wrong or nothing, nothing will happen.
Array	Hearts[]	Integer	Array holding 10 images of hearts.	This is needed to display the hearts on the UI. It is also needed to change full hearts into empty hearts when the player is damaged.	The array is based on an 8-bit pixel heart. Therefore, the image must 32 by 32pixels and a jpeg image.
Variable	health	Integer	Integer that demines how much health the player has.	This variable is accessed and changed during the code to show the player took damage and reflect the damage on to the hearts on the UI.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
HRT1	Is the max number of hearts = to the max health?	No Input Required.	There should be 10 hearts to display the 10 health the player has.
HRT2	Do the hearts turn empty after taking damage?	No input required.	Depending on how much damage the player takes. The number of hearts that turn empty should equal the amount of damage he took.
HRT3	Do the hearts fill up when an powerup restores health of the player.	A and D. Must move and collide with an ambrosia. To restore health.	Hearts should fill up on the amount the ambrosia heals the player.

ENEMY CREATURES

WHY?

After creating and finish the controls for the player. It is logically to create the enemy and add its functions. Enemies are needed otherwise it would make the game very dull. Enemies are needed to create a tense and difficult atmosphere and provide the player with a challenge.

ENEMY CHASE AI

WHY?

The chase AI is needed to create a difficult game. The chase AI allows the enemies of the game to apply pressure to the player. It makes the game harder and more enjoyable to play, it attracts players and stakeholder to keep playing and beat the pressure and survive the wave.

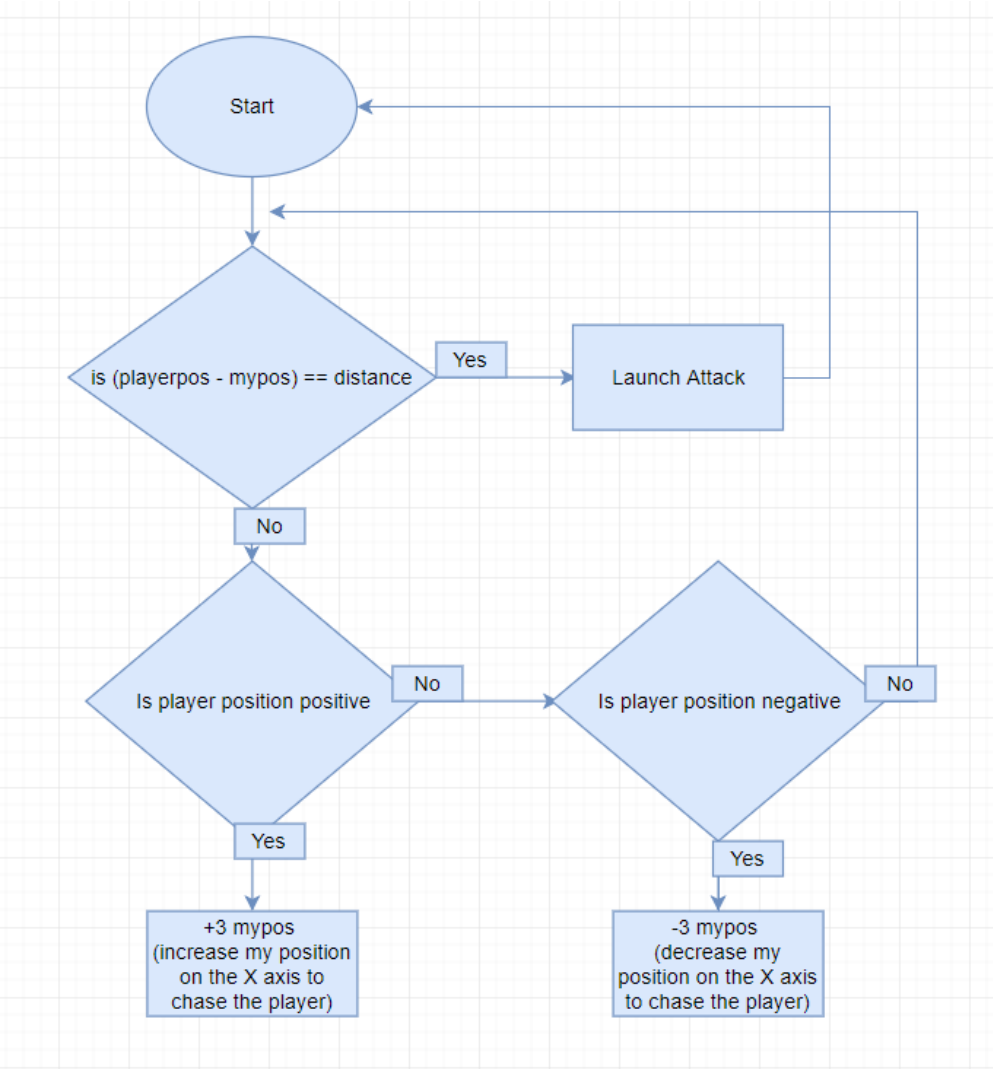
PSEUDOCODE

```
Distance = 1
Mypos
Playerpos
Speed

Do Until player.pos – mypos = distance

        Find object Player
        If playerpos >= to 0
                Mypos = mypos + speed
        End If

        If playerpos =< to 0
                Mypos = mypos – speed
        End if
End Do Until
```

PSEUDOCODE EXPLAINED

If the enemy is not within attacking distance of the player, the enemy will chase the player. It differentiates which direction the player is by determining the position of the player. If the player is to the right then the player position on the X axis is positive, the enemy will chase the player by increasing its position on the X axis. Vice versa for the left side, the player's position will be negative, so the enemy will decrease its position on the X axis.

FLOWCHART


Start
is (playerpos - mypos) == distance
Yes
Launch Attack
No
Is player position positive
No
Is player position negative
No
Yes
Yes
+3 mypos
(increase my position
on the X axis to
chase the player)
-3 mypos
(decrease my
position on the X axis
to chase the player)

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid
A	Valid
S	Invalid
D	Valid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- A and D are needed to move the player to the right and left. This is done to check if the enemy AI script can follow the player in both directions.
- W is needed to test if the AI still follows the player if the player is jumping towards the right or the left.

KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Variable	Playerpos	Integer	Integer that holds the position of the player's current X axis position.	It is needed for the enemy AI to determine the location of the player so it can chase the player.	No validation. If the input is wrong or nothing, nothing will happen.
Variable	Mypos	Integer	Integer that holds the current position of the enemy.	It is needed to allow the movement of the enemy towards the player. This value will increase or decrease depending on	No validation required. If the input is wrong or nothing, nothing will happen.

				the player's position. E.g. to the right or left.	
Variable	Distance	Integer	Integer that determine that the enemy is close to the player.	This variable is needed to stop the enemy from chasing the player once it is close enough. This is done so that the enemy can launch its attacking move against the player when it is in range. It also will make sure the enemy will chase the player until he has died or is out of the range.	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	Speed	Integer	Integer that determines the speed of the enemy.	This is needed to determine the speed the enemy travels towards the player. This is needed as this value will change as the player survives longer waves, this is done to increase the speed of the enemies and make the game more difficult	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
CH1	Does the enemy chase to the left?	A	Player should move to the left and the enemy should follow the player.

CH2	Does the enemy chase to the right?	A	Player should move to the right and the enemy should follow the player.
CH3	Does the enemy chase the player if he jumps to the right or left?	A and W Or D and W	Player should jump to the left or right and the enemy should follow the player.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

ENEMY CREATURES MOVE

WHY?

The final part of the enemy creature system. Only logical to code this next to complete the enemy creature system. The enemy attack move is needed because without it the game would be very dull as the player will have no real threat to deal with. Without the enemy attack move the game loses its difficulty

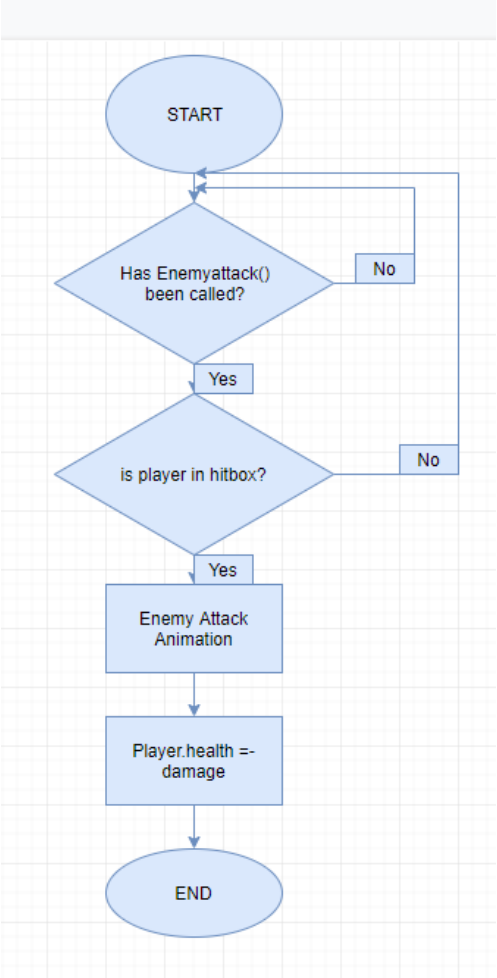
PSEUDOCODE

```
Enemyattack()
Damage = -1
        If player is in Hitbox
                Launch enemyattack animation
                Player.health =  -damage
        End if
End Procedure
```

PSEUDOCODE EXPLAINED

The attack will only launch if it has been called for. When it is called, if the player is in the hitbox radius the enemy will launch an attack animation and damage the player.

FLOWCHART



TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Invalid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- No input is needed as the code is being called in Enemy Movement, and the inputs from the Enemy Movement test will suffice for the test.

KEY VARIABLES, JUSTIFICATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Procedure	Enemyattack()	N/A	A procedure that is called in the Enemy Movement script.	It launches an attack when the enemy is within range of the player.	No validation. If the input is wrong or nothing, nothing will happen.
Variable	Damage()	Integer	Integer that holds how much the player will get damaged by.	It determines how much damage the player takes. The higher the damage the more health the player loses. Without this variable the player would not take damage from enemy attacks.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
EDMG1	Does the enemy damage the player?	No Input Required	The enemy should attack the player and damage him by the damage variable.
EDMG2	Does the attack animation launch?	No Input Required	The enemy attack animation should

			play when the enemy attacks the player.
EDMG2	Does the enemy damage nearby allies.	No Input Required	The enemy should only be able to damage the player.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

WAVE SYSTEM

WHY?

The wave system is needed because without it no enemies would ever spawn to kill the player. There would be no aim to the game as without it the survival and fighting aspect is taken away from the game.

PSEUDOCODE

```
Wave Object
{
        Float rate
        Int numberofenemy
        String name
        Gameobject enemy
}

Wave[ ] wavesofenemyies
Int nextwave
Float wavecooldowntime
gamestates: Spawning, Counting, Waiting
```

```
VOID UPDATE()
```

```
If gamestate = Waiting
        If Enemystillalive() = true
                Wavecompleted()
        Else
                return
        End If
End If

If (wavecountdowntime <= 0)
                Start spawner(waveofenemies[nextwave])
        End if
Else
        Wavecooldowntime = -1 second
End If
```

Functions

```
EnemStilLAlive() Boolean
        Float searchCountdown
        Bool EnemyFound
        searchCountdown = gametime – 1 second

        if (searchcountdown <= 0f)
                searchCountdown = 1f;
                if (Find gameobjcet with tag("enemy"))
                        EnemyFound = true
                Else
                        Return false
End Procedure

WaveCompleted()
        UI.text= "wave completed"
        State = spawnstate.counting
        Wavecountfown = wavecooldowntime
If (next wave > 10)
        DisplayWinner()
Else
        Nextwave = nextwave +1
End Procedure
```

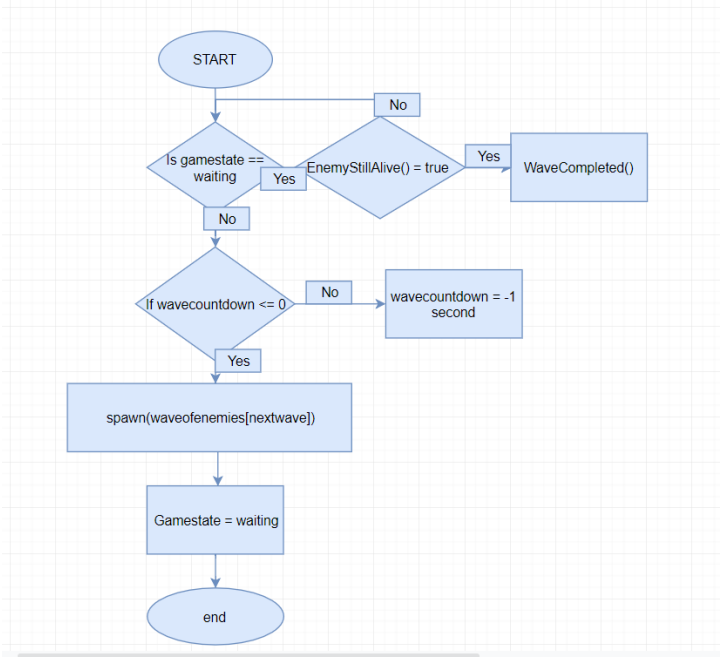
PSEUDOCODE EXPLAINED

Creates a new object called ***Wave***. An array with 10 elements of these ***Wave*** object will be created. During the game it will run procedure ***EnemyStillAlive()*** and check if it's true**.** If there isn't it will launch procedure ***WaveCompleted()***. The procedure will launch a countdown for the next wave. When this countdown is completed it spawns the next Wave in the Wave Array. It will stop when it reaches 10.

In function ***EnemyStillAlive().*** During a set amount of time it searches for any gameobject with enemy in their tag. If it does find one, then EnemyFound is true and the game will not move on to the next wave as the function returns true. If there are no enemies alive the function returns false, therefore the game will carry onto the next wave.

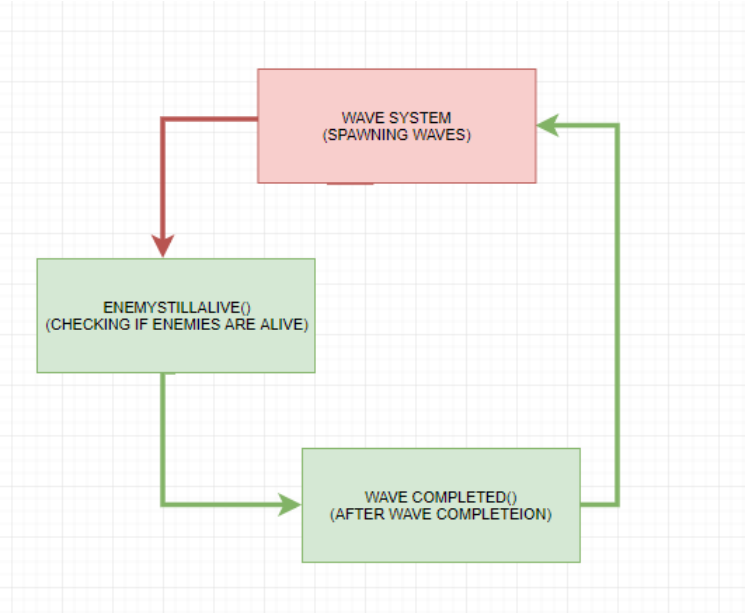
In procedure ***WaveCompleted().*** It changes the game state to counting. During this the player gets time to recuperate before the next wave starts. The game will wait for its set ***wavecooldontime***. Afterwards it checks if the wave Is more than wave number 10. If so the game is completed, and it displays the winner screen. Otherwise it moves onto the next wave in the wave array. (THIS CODE WILL LOOP DUE TO VOID UPDATE())

FLOWCHART



LINKING OF FUNCTIONS/ALGORITHIMS TO COMPLETE THE WAVE SYSTEM

Red = algorithm/code



Green = functions.

This diagram shows how the algorithm and functions call each other and give an overview of how the functions and algorithm form the completed wave system.

For example the algorithim cannot call ***wavecompleted*** without going through the ***enemystillalive*** function.

TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Invalid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid

Justification

- No input needed yet again this is testing the enemy AI and system. The player does not have to input anything. As the wave system is automatic and is called every frame of the game due to void update() (***Unity Syntax***).

KEY VARIABLES, JUSTIFCATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Procedure	Void Update()	N/A	A procedure that is called in the Enemy Movement script.	It launches an attack when the enemy is within range of the player.	No validation. If the input is wrong or nothing, nothing will happen.
Boolean Function	EnemyStillAlive()	N/A	A function that returns true or false depending on the number of enemies.	It's a function that checks if there are still enemies remaining from the wave. This is to make sure the game only launches the next wave if the player has killed all the enemies of the current wave. If this was not implemented the game would be too hard as the player would be overwhelmed with enemies.	No validation. If the input is wrong or nothing, nothing will happen.
Procedure	Wave Completed()	N/A	A procedure that communicates with the player and launches the next wave.	This procedure is here to give the player information that they have beat the wave. Additionally, it gives some time before the next wave is launched.	No validation. If the input is wrong or nothing, nothing will happen

				This needed to inform the player of the situation and give him some time to strategize for the next wave of enemies.	
Object	Wave	N/A	An object that all the waves are children of.	Instead of creating loads of separate wave objects. The player can create waves under the parent object wave. Therefore, the game can access each property of the object to increase the difficulty of the waves.	No validation. If the input is wrong or nothing, nothing will happen
Array	Wave[]	Object	An array that holds the 10 waves in its elements.	This array is used to store the different number of waves. This is done so that each wave can be spawned in order and in increasing difficulty. As this makes the game more structured, intense and fun for stakeholders and players.	You can only store children of the wave object in this array.
Variable	rate	Float	Float that is the rate of which the game will spawn enemies	The game needs to access this variable to change the rate of enemies spawning in each incrementing wave.	No validation. If the input is wrong or nothing, nothing will happen

				This is done to increase the difficulty of the game and increase pressure on the player.	
Variable	numberofenemy	Int	Number of enemies in each wave.	The game accesses this variable and increases it to increase the number of enemies in each wave. This is done to increase the difficulty of the game and put more pressure on the player.	No validation. If the input is wrong or nothing, nothing will happen
Variable	name	String	Name of each wave.	This is done to name each wave. So that it can be broadcasted on the UI to give the player information of what wave they are on.	No validation. If the input is wrong or nothing, nothing will happen
Variable	enemy	Object	What type of enemy the wave will spawn.	This is done to give the player variety. Different enemies will span. This gives the player variation to what he kills.	No validation. If the input is wrong or nothing, nothing will happen
Variable	NextWave	Int	Counter that reflects wave number.		No validation. If the input is wrong or nothing, nothing will happen

Variable	EnemyFound	Integer	Integer that holds how much the player will get damaged by.	It determines how much damage the player takes. The higher the damage the more health the player loses. Without this variable the player would not take damage from enemy attacks.	No validation required. If the input is wrong or nothing, nothing will happen.
Variable	Wavecooldowntime	Float	The time the player gets between waves to strategize and rest.	Gives the player time to rest and strategize for the next wave. It makes sure the game doesn't spawn to waves at once and overwhelm the player.	No validation required. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
WAV1	Does the system spawn enemies?	No Input Required	The system should spawn the set amount of enemies for that wave.
WAV2	Does the system spawn more enemies at a faster rate as the wave number increases?	No Input Required	The speed of spawning and the number of enemies should increase as wave number increases.
WAV3	Does it go past wave 10	No Input Required	The wave number should not go past 10. It should stop and display the winners screen.

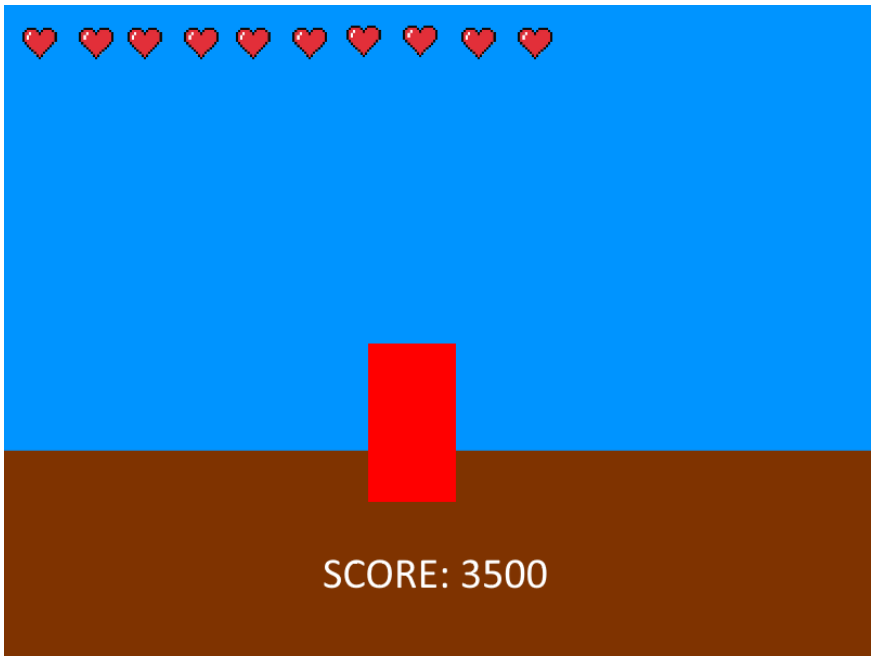
WAV4	Can it spawn more than 2 enemies at once?	No Input Required	The wave system should spawn more than 2 enemies if the rate is high enough.

SCORE SYSTEM AND ENEMY DEATH

WHY?

To accommodate stakeholder needs. Allow the stakeholders/players to separate themselves into groups of good and bad players as the score allows them to compare their skills to each other.

UI



PSEUDOCODE

The score system can be injected into the enemy script.

Score UI during the game.

```
Public Variable Score
UIText Score;

Void Update()
        Score.text = "score = " + score
End Update()
```

Enemy death/rewarding points.

```
If (health <= 0)
        Destroy gameobject
        score = score + 10
End If
```

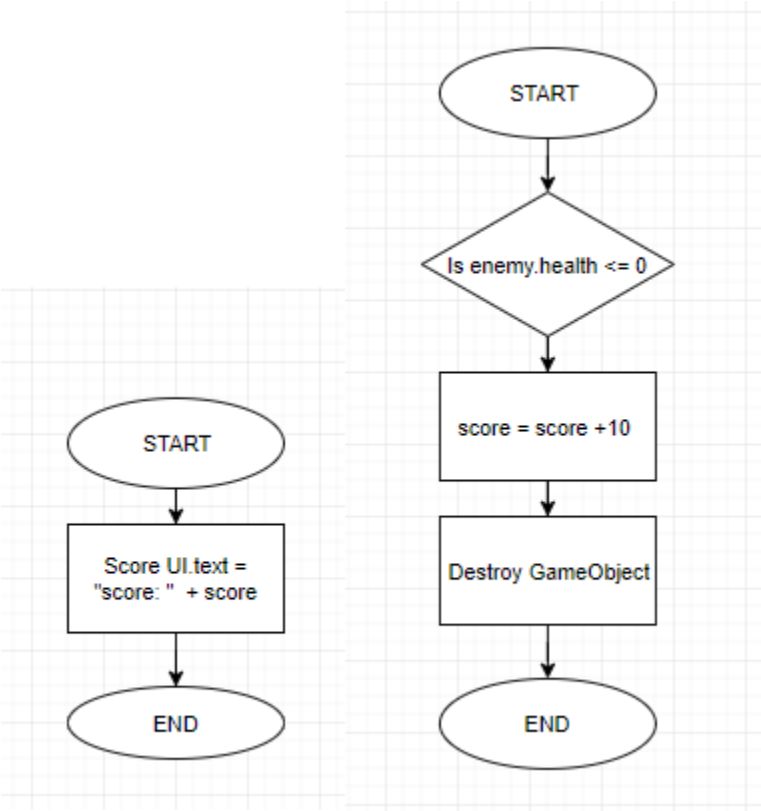
PSEUDOCODE EXPLAINED

Score UI during the game

Void Update(), during every frame it checks the score value of the player and updates the UI so it displays the score correctly.

Enemy death/rewarding points.
When an enemy dies, it destroys itself and awards the player 10 points.

FLOWCHART



TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid/Borderline
A	Valid
S	Invalid
D	Valid
J	Valid
K	Invalid
L	Invalid
P	Invalid

Justification

- Player must move and kill enemies. Therefore A, S and J are needed as inputs. As the player must move around and attack enemies.

KEY VARIABLES, JUSTIFCATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
Procedure	Void Update()	N/A	Unity Syntax	Runs code every new frame of the game. This is done so that the UI for the score in constantly updated in real time as the player earns more points.	No validation. If the input is wrong or nothing, nothing will happen.
Variable	Health	Integer	A variable that determines enemy health	It is needed to make the enemy die when the player attacks the enemy. As without it the player will be fighting against unkillable enemies and would not be awarded points.	No validation. If the input is wrong or nothing, nothing will happen.
Public Variable	Score	Integer	A public variable that stores the score of the player.	Holds the score value of the player. It is a public variable because it needs to be accessed by other systems such as the enemy system, to award the player points. The score system to display the score in real time on the UI and the leaderboard system to store the score.	It has to be an integer. As the score is stored into external game files.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
DE1	Does the enemy die in the game?	A and D to locate enemies. J to attack enemies.	Players should be able to kill enemies by attacking them. When the enemy HP is 0 or less it should die.
DE2	When an enemy dies does it award the player points?	A and D to locate enemies. J to attack enemies.	The player should be awarded 10 points.
SC1`	Does the UI give the current score of the player.	A and D to locate enemies. J to attack enemies.	The UI should display the score of the enemy. It should updates its self in real time when he gets more points.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

LEADERBOARDS

WHY?

To separate the players from the good and bad. Allows the players to separate themselves into good and bad players. This was a requested feature from the stakeholders.

PSEUDOCODE

```
Load int playerprefs bestscore;
Load string playerprefs thebestplayer;

Highscore.text = “highscore = “ + bestscore
Thebest.text = “the best = “ + thebestplayer

If currentscore > bestscore
        Playerprefs bestscore = highscore
```

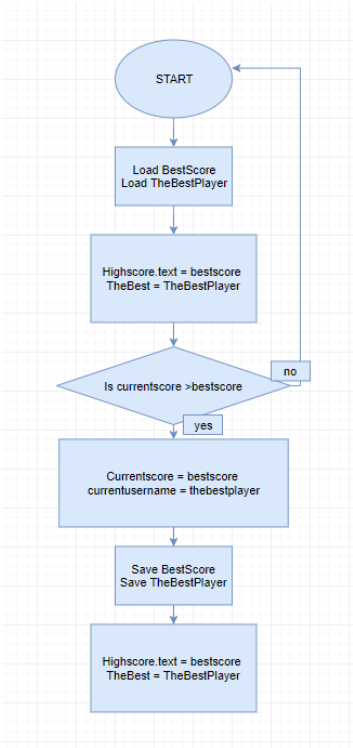
Playerprefs thebest = currentusername
Highscore.text = "highscore = " + bestscore
Thebest.text = "the best = " + thebestplayer
End if

PSEUDOCODE EXPLAINED

Playerprefs is a unity method that allows the game to store data into external files. The game will load the variables ***bestscore*** and ***thebestplayer*** from these external files and display them in the leaderboards screen of the game.

However, if a player beats that score then it changes the ***bestscore*** integer and ***thebestplayer*** string to the ***currentscore*** and the ***currentusername***. Replacing the old high-score with the new one. It also changes the text in the leaderboards so the changes are immediate.

FLOWCHART



TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Valid/Borderline
A	Valid
S	Invalid
D	Valid
J	Valid
K	Invalid
L	Invalid
P	Invalid

Justification

A,S,W and J are inputs as player must survive and kill. To beat the highscore and check if the script is functional and to check if their usernames are on the leaderboards screen

KEY VARIABLES, JUSTIFCATION AND VALIDATION

Method	Name	Data Type	Description	Justification	Validation
PlayerPrefs Variable	BestScore	Integer	Unity Syntax/Variable	Stores this variable outside the game so that the ***bestscore*** can be saved when the game is not running. When the game is reopened this variable is retained and displayed.	Must be an integer or the game will not store the score.
PlayerPrefs Variable	TheBestPlayer	String	Unity Syntax/Variable	Stores this variable outside the game so that the ***thebestplayer*** can be saved when the game is not running. When the game is reoened this variable	Must be a string or the game will not save the username.

				is retained and displayed.	
GameObject	Highscore	Integer	A gameobject that stores an integer.	This gameobject is displayed on the UI. It is changed when the highscore is beaten as a new integer will replace the old integer. It gives players information about the best score on the leaderboards screen.	It must be an integer as it displays the variable bestscore.
GameObject	TheBest	String	A gameobject that stores a string.	This gameobject is displayed on the UI. It is changed when the highscore is beaten as a new string will replace the old string. It gives players information about the best player on the leaderboards screen.	It must be a string as it displays the variable thebestplayer.
Variable	CurrentScore	Integer	An integer that stores the current score of the player.	The current player's score is stored in this variable. If he beats the highscore the variable value will be copied into the thebestscore variable. Making the current player's score the new highscore.	Must be an integer.

Variable	CurrentUsername	String	An integer that stores the current username of the player.	The current player's username is stored in this variable. If he beats the highscore the variable value will be copied into the thebestplayer variable. Making the current player the best at the game.	Must be a string.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
LED1	Are the bestscore and the thebestplayer variables saved externally?	No input required.	These variables should be stored externally in a file so that they can be used when the game is reopened.
LED2	Does the leaderboards display the externally stored variables.	No input required.	The screen should display the externally stored variables. This is done so that we can see the Best Player's name and the score.
LED3	If the highscore is beaten are the new details saved and the old ones removed?	A and D to locate enemies. J to attack enemies. Must beat the old highscore.	The game should replace the old highscore details and replace it with the current player's details.

RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION.

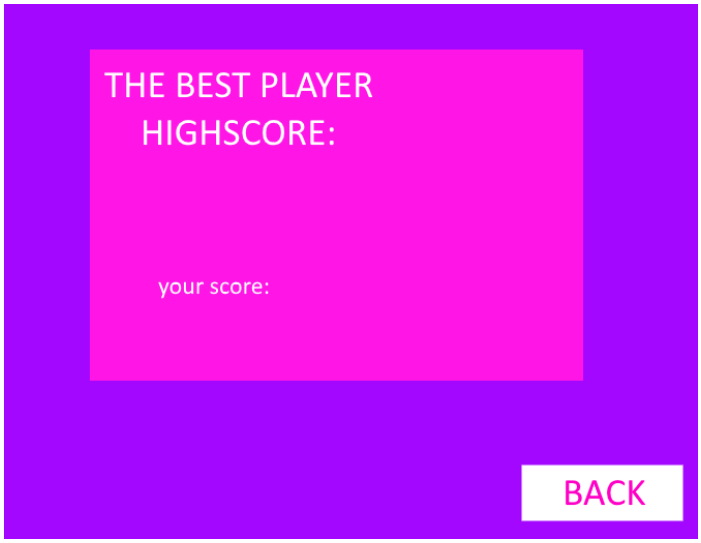
MENUS

UI

MAIN MENU SCREEN

HOW TO PLAY SCREEN

LEADERBOARD SCREEN



UI EXPLAINED

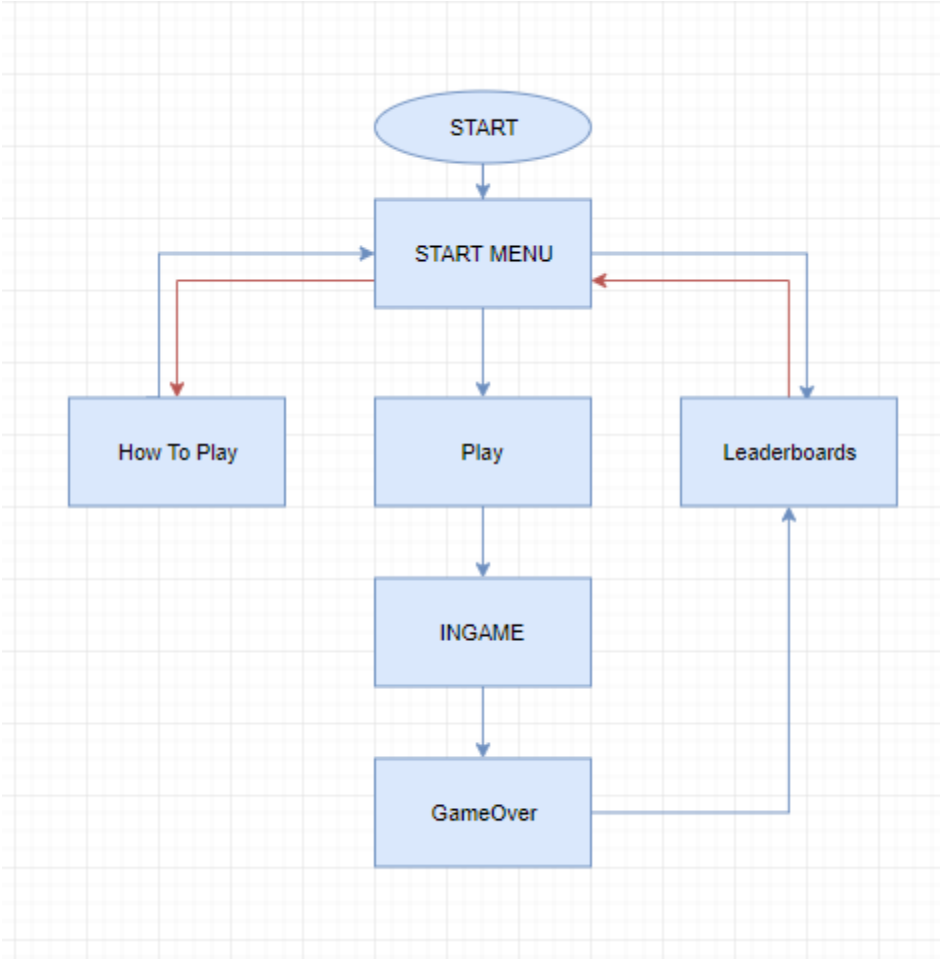
- The background of the menu screens is purple because the enemy creatures are purple. This is to create a theme within the design of the game.
- There is a back button available at the bottom right of the screen in an abstract white colour. This ensure the player does not miss the button. Player can click it to return to the main menu screen.
- In the leaderboard screen the high-score and the best player's username is placed at the top of the screen in a large font size. This is to show that the player who achieved that score is the best.
- The current player's score can be seen at the bottom in a smaller font size. This is to show that the current player is not as good as the best player. It is an incentive for him to keep playing and try to beat the high-score so his username is at the top.

PSEUDOCODE

All these procedures will be called when the corresponding button is pressed. For example, if I press play. LaunchGame() will be called and the game will be loaded.

```
LaunchHTP()
        Load howtoplay
End Procedure

LaunchGame()
        Load game
End Procedure

LaunchLeaderboards()
        Load Leaderboard
End Procedure

LaunchMenu()
        Load Menu
End Procedure
```

HOW THE SCREENS/PROCEDURES FIT TOGETHER



TEST DATA FOR DEVELOPMENT AND JUSTIFICATION

Test Data	Type
W	Invalid
A	Invalid
S	Invalid
D	Invalid
J	Invalid
K	Invalid
L	Invalid
P	Invalid
Left Mouse Click	Valid

Justification

- Mouse is used to navigate through the menu screens. As many other applications use the mouse to maneuverer through the menu..

KEY VARIABLES, JUSTIFCATION AND VALIDATION

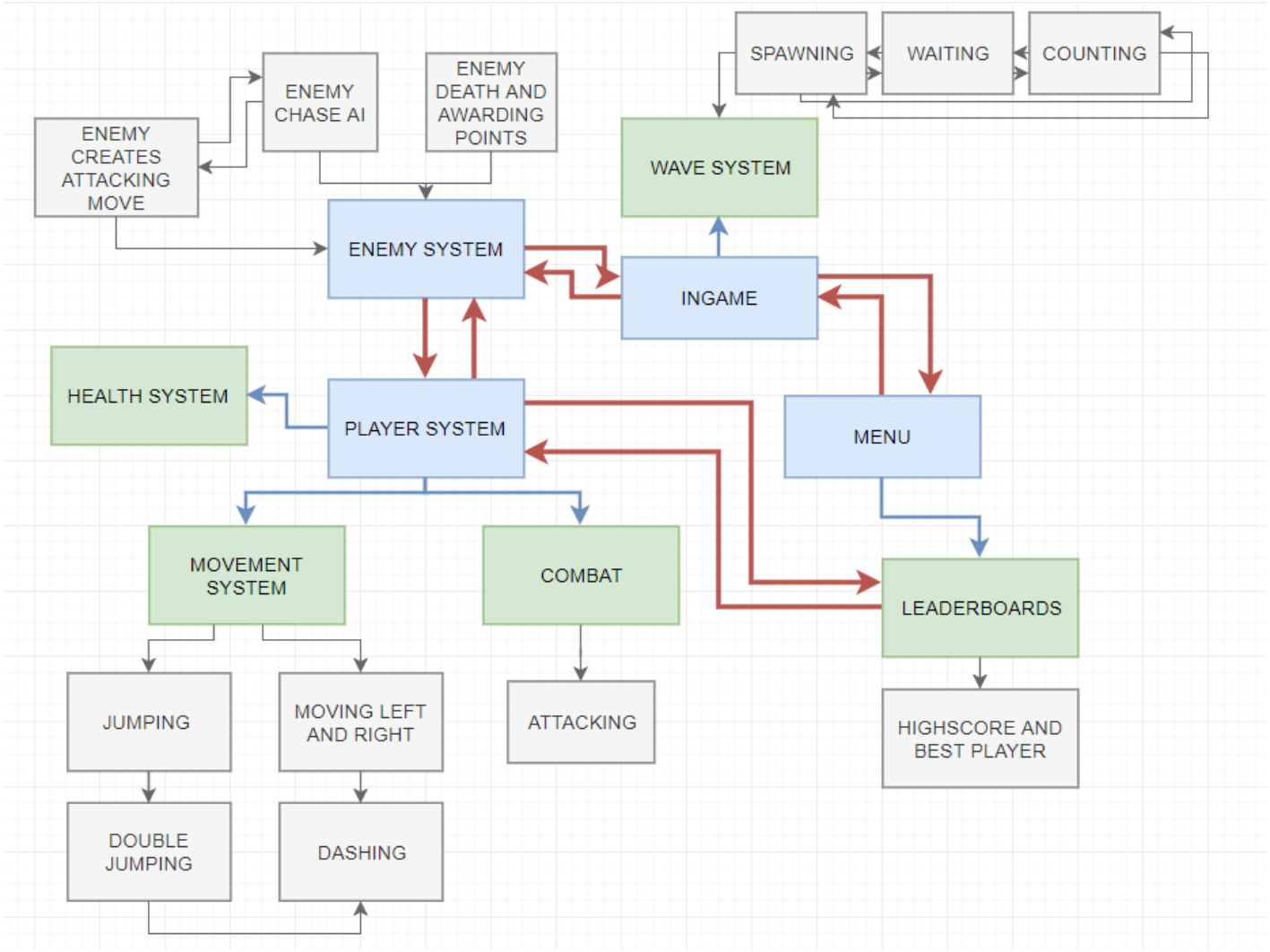
Method	Name	Data Type	Description	Justification	Validation
Prodecure	LaunchGame()	N/A	Procedure that launches when the corresponding button is clicked.	Without it the game cannot be played. It would never leave the menu and start the game.	No validation. If the input is wrong or nothing, nothing will happen.
Prodecure	LaunchHTP()	N/A	Procedure that launches when the corresponding button is clicked.	Allow players to see an overview of the controls at a different menu screen. Giving them an idea of the mechanics of the game.	No validation. If the input is wrong or nothing, nothing will happen.
Prodecure	LaunchLeaderboard()	N/A	Procedure that launches when the corresponding button is clicked.	Allow the player to see who currently holds the best score. This adds competition between players as they play more to beat each other's high score.	No validation. If the input is wrong or nothing, nothing will happen.
Prodecure	LaunchMenu()	N/A	Procedure that launches when the corresponding button is clicked.	Launched when the back button is pressed, so you can return to the main menu from the HowToPlay screen or Leaderboard screen.	No validation. If the input is wrong or nothing, nothing will happen.

TEST DATA FOR BETA TESTING

Test Identification	Test	Input	Expected Outcome
MNU1	Can you traverse to all menu screens.	Left Click on buttons.	The player should be able to access all menu screens from the main menu, if the corresponding button is pressed.
MNU2	Does the back button work?	Left Click on the back button.	After reaching another screen the player should return to the main menu if he clicks the back button.

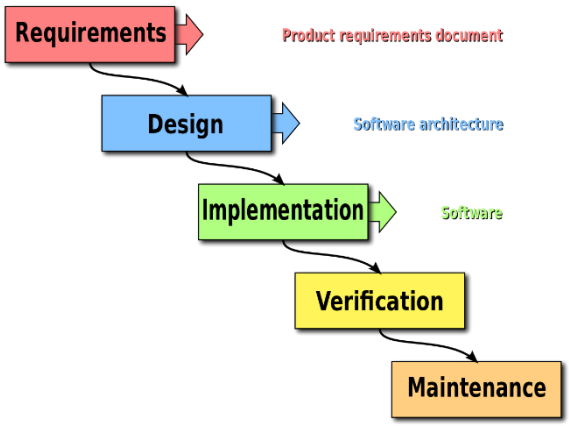
RED TESTS ARE DESGINED TO TEST THE ROBUSTNESS OF THE SOLUTION

HOW THE ALGORITHIMS LINK TO FORM THE COMPLETE SOLUTION



SOFTWARE DEVELOPMENT LIFE CYCLE

During the analysis and design section I will be using the traditional waterfall method approach to the solution. I will take in requirements and add them to the design.



However, during implementation and development I will be changing methods to the agile method. This is to ensure that the stakeholders has some input to what is created. Furthermore, they can review at key stages of development what they don't like and what they like. Additionally, they could suggest improvements that make the solution better.

C. DEVELOPING THE CODED SOLUTION ("THE DEVELOPMENT STORY")

JUMPINNG

REDESIGN REQUIRED

Since I am using Unity to develop my solution rather than Visual Studio there were some issues coding the solution. As the design was tailored towards a Visual Studio solution. The different objects and components, such as RigidBody2D physics component, of Unity caused errors in development. I was not able to implement the initial pseudocode I designed, therefore, I had to redesign my approach and pseudocode for the solution.

PROTOTYPE 1/COMMENTED CODE (11/11/18)

COMMENTED CODE

```
public float jumpforce;
//variables created
//jumpforce will dictate how far up the player can jump.
```

```
void Start()
// Start is called before the first frame update
{
    Debug.Log("im alive");
    //debug tool too see if the script is working.
}
```

```
void Update()
// Update is called once per frame
{
    Rigidbody2D rb = GetComponent<Rigidbody2D>();
    Debug.Log("i got the component");
    //gets the physicis component from unity assests
    //needed to launch character

    if (Input.GetKey(KeyCode.W))
    // if input is w
        Debug.Log("working");
    rb.velocity = (new Vector2(0, jumpforce));
    // player's y velocity is increased by jumpforce variable
}
```

The variables values are modified on the unity program.

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
Ju1	Jumping	W to jump, to see if the player can actually jump.	The character should jump when W is pressed.	Failed
Ju2	Falling down	W to jump to see if the player falls down after jumping.	The character should fall back down after jumping.	Failed

Ju3	Sensible Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.	Failed
Ju4	Pressing other buttons	W to jump and any other buttons.	Jump should still occur regardless the fact that another button was pressed.	Failed

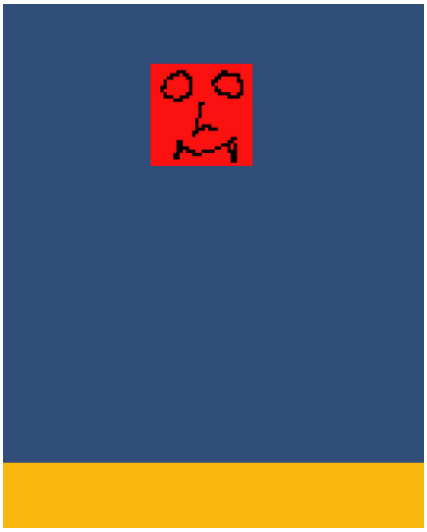
Red = Tests designed to test the robustness of the solution (game breaking).

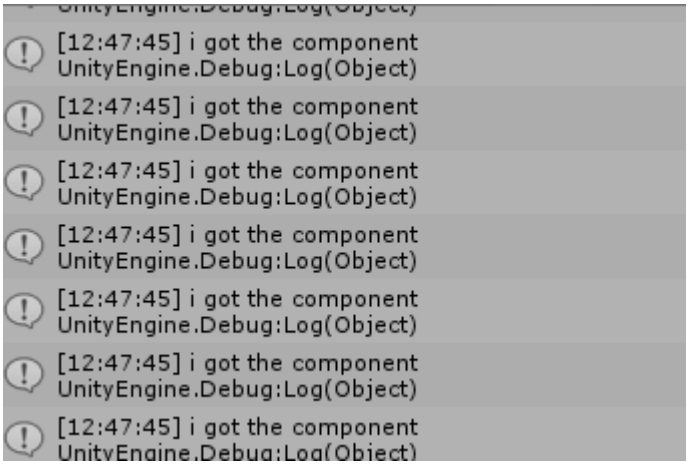
PROBLEMS

Problem	Problem Explained
Ju1	The character would jump even though space was not pressed.
Ju2	Character would continuously float upwards and would not fall down.
Ju3	Gravity pulling force did not work as the character kept floating upwards.
Ju4	Could not conduct test as the character jumped without pressing W.
Ju5	Debug log filled with “I got the component”, making debugging impossible as this was the only message displayed.

Purple = Problem was not because of a planned test, it was discovered.

PICTURES OF PROBLEM





- You can see the character is just floating upwards with nothing being pressed as working is not being outputted by the debug tool. If W was pressed, then the debug tool would output “working”.

- The debug window has been spammed by the “I got the component”. This made debugging near to impossible as it was blocking me from seeing other debug messages.

PROTOTYPE 2/COMMENTED CODE (20/11/18)

```
public float jumpforce;
public Rigidbody2D rb;
//variables created
//jumpforce will dictate how far up the player can jump.
```

```
void Start()
// Start is called before the first frame update
{
    Debug.Log("im alive");
    rb = GetComponent<Rigidbody2D>();
    //gets the physicis component from unity assests
    //needed to launch character
    Debug.Log("i got the componenet");
    //debug tool too see if the script is working.
}
```

```
void Update()
// Update is called once per frame
{
    if (Input.GetKey(KeyCode.W))
    {
        // if input is w
        Debug.Log("working");
        rb.velocity = (new Vector2(0, jumpforce));
        // player's y velocity is increased by jumpforce variable

    }
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
Ju1	Jumping	W to jump, to see if the player can jump.	The character should jump when W is pressed.	Passed
Ju2	Falling	W to jump to see if the player	The character should fall back	Passed

		falls after jumping.	down after jumping.	
Ju3	Sensible Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.	Passed
Ju4	Pressing other buttons	W to jump and any other buttons.	Jump should still occur regardless the fact that another button was pressed.	Passed
Ju5	Debug Log Spam	W to jump.	Debug lines should only be posted once and not spam the log.	Passed

SOLUTIONS TO PROBLEMS FROM PROTOTYPE 1

Test Identification	Outcome/Solution
Ju1	Player Jumps when pressing W It was jumping without pressing w because in prototype 1 the code that made the sprite jump was called outside the If statement. Therefore it kept occurring as the void update() function was carried out, causing the sprite to jump continuously.

Ju2	Player falls back down after jumping. As the character has stopped jumping continuously, it now falls back down to the ground. This was a unity asset that was present in the Rigidbody2D component called *"boxcollider2D"*
Ju3	Adjustable gravity pulling force The character has stopped jumping continuously so now the variables can be changed for the gravity pulling force to bring the character down faster or slower.
Ju4	Should jump even if other keys are pressed. The character can jump even if W and other buttons are pressed.

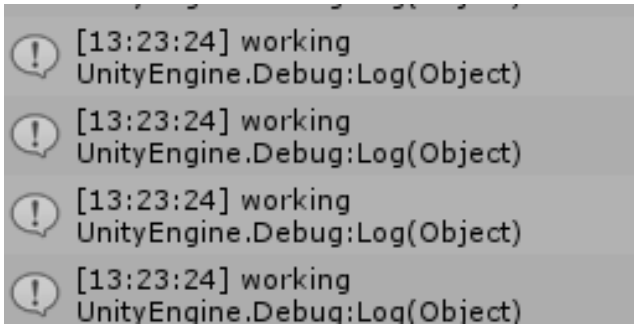
Ju5	The debug line is now outside the ***void update***, therefore the debug line is not posted every time the frame is updated.
	
	Therefore it does not spam the log with "I got the component"
	
	`void Start()`
	`// Start is called before the first frame update`
	`{`
	`  Debug.Log("im alive");`
	`  rb = GetComponent<Rigidbody2D>();`
	`  //gets the physicis component from unity assests`
	`  //needed to launch character`
	`  Debug.Log("i got the componenet");`
	`  //debug tool too see if the script is working.`
	`}`

PROBLEMS

Problem	Problem Explained
Ju6	The player can infinitely press W to jump continuously when in the air. You should only be allowed to jump once.

PICTURES OF THE PROBLEM

- The line of code that outputs “working” in the debug window is nested into the if statement. Therefore, “working” is only outputted when W is pressed. The screenshot below is evidence that the player can press W repeatedly to jump.



FINAL VERSION (27/11/18)

COMMENTED CODE

```
public float jumpforce;
public Rigidbody2D rb;
public bool grounded;
//variables created
//jumpforce will dictate how far up the player can jump.
//grounded will check if the player has touched the ground
```

```
void Start ()
// Start is called before the first frame update
{
    grounded = false;
    Debug.Log("im alive");
    rb = GetComponent<Rigidbody2D>();
    //gets the physicis component from unity assests
    //needed to launch character
    Debug.Log("i got the componenet");
    //debug tool too see if the script is working.

}
```

```
void OnTriggerEnter2D()
 // when the player and the ground touch
{
    grounded = true;
    Debug.Log("ground check yes");

}

void OnTriggerExit2D()
// when the player and the ground are not touching
{
    grounded = false;
    Debug.Log("ground check no");
}
```

```
void Update () {
// Update is called once per frame

    if (Input.GetKey(KeyCode.W) && grounded == true)
        //only allowed to jump is grounded is true, therefore cannot infinitley jump.
    {
        Debug.Log("working");
        rb.velocity = (new Vector2(0, jumpforce));
        Debug.Log("jump sequence launched");
        // player's y velocity is increased by jumpforce variable
    }


}
```

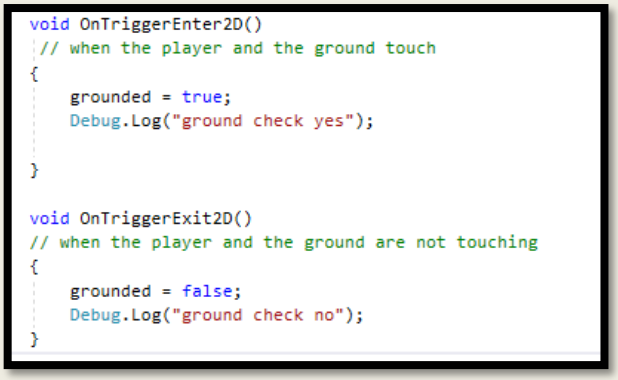
TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
Ju1	Jumping	W to jump, to see if the player can jump.	The character should jump when W is pressed.	Passed

Ju2	Falling	W to jump to see if the player falls after jumping.	The character should fall back down after jumping.	Passed
Ju3	Sensible Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.	Passed
Ju4	Pressing other buttons	W to jump and any other buttons.	Jump should still occur regardless the fact that another button was pressed.	Passed
Ju5	Debug Log Spam	W to jump.	Debug lines should only be posted once and not spam the log.	Passed
Ju6	Infinite Jump	Repeatedly press W	Player should only be able to jump once.	Passed

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 2

Test Identification	Outcome/Solution
Ju6	Using ***BoxColliders (Unity Asset)*** as triggers and combining it with a new Boolean. The Boolean will check if the player is touching the ground. Using this Boolean I updated the If statement therefore, the player can only jump if the player is touching the ground.

	Removing the problem of infinite jumps. 

```
void OnTriggerEnter2D()
 // when the player and the ground touch
{
    grounded = true;
    Debug.Log("ground check yes");

}

void OnTriggerExit2D()
// when the player and the ground are not touching
{
    grounded = false;
    Debug.Log("ground check no");
}
```

STAKEHOLDER FEEDBACK

Ben: *I like the jumping its fluid and responsive, however I think it's a bit too high. Reduce the jump force so the player can't jump too high and avoid enemies.*

Isaac: *Since the game is a platformer game, you should add double jump, it allows the player another movement option to dodge enemies.*

FEEDBACK RESPONSE

- The jumpforce has been reduced to accommodate Ben's feedback.
- Additional code must be written to accommodate Isaac's feedback.

Prototype 1 (Double Jump) (Addressing Isaac's Feedback) (3/12/18)

```
void Update ()
{
// Update is called once per frame

    if (Input.GetKey(KeyCode.W) && grounded == true)
        //only allowed to jump is grounded is true, therefore cannot infinitley jump.
    {
        Debug.Log("working");
        rb.velocity = (new Vector2(0, jumpforce));
        // player's y velocity is increased by jumpforce
        doublejump = true;
        Debug.Log("jump sequence launched");
        if (doublejump == true)
        {
            if (Input.GetKey(KeyCode.W))
            {
                rb.velocity = (new Vector2(0, jumpforce));
                doublejump = false;
            }
        }


    }
```

//After launching the jump command double jump will be turned to true.

//if double jump is true the player has double jump available. When he

After jumping the 2nd time, double jump will be unavailable, this is to ensure

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump	Player should be able to jump once then jump again while still in the air.	Failed
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being	Player should only be allowed to double jump, there should be no additional jumps until he	Untested

		tested as this was an issue with the first original jump design.	falls back on the ground.	

Test DJu2 has not been tested as the double jump is currently not working

PROBLEMS

Problem	Problem Explained
DuJ1	The player can only jump once. There is a problem in the code, the Boolean DoubleJump never turns true therefore the player cannot double jump.

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 1

Test Identification	Outcome/Solution
DJu1	The ***doublejump If statement*** was nested inside the ***initialjump if statement.*** Therefore, the ***doublejump*** if statement was never run as the ***initial if statement*** was only run when ***grounded was true.*** Therefore, the ***doublejump if statement*** was never reached because the double jump occurs when the player is ***not grounded***

PROTOTYPE 2 (13/12/18)

```
void Update()
{
    // Update is called once per frame

    if (Input.GetKey(KeyCode.W) && grounded == true)
    //only allowed to jump is grounded is true, therefore cannot infinitley jump.
    {
        rb.velocity = (new Vector2(0, jumpforce));
        Debug.Log("jump sequence launched");
        jump2 = true;
        // player's y velocity is increased by jumpforce variable
    }

    if (jump2 == true)
        Debug.Log("jump 2 working");
    {
        if (Input.GetKey(KeyCode.W))
        {
            rb.velocity = (new Vector2(0, jumpforce));
            jump2 = false;
            Debug.Log("jump2 is false");
        }

    }
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump	Player should be able to jump once then jump again while still in the air.	Passed
DJu2	Infinite Jump	Pressing W repeatedly to check if the	Player should only be allowed to double jump,	Failed

		character can infinitely jump. This is being tested as this was an issue with the first original jump design.	there should be no additional jumps until he falls back on the ground.	

PROBLEMS

Problem	Problem Explained
DuJ2	The player can jump infinitely after the double jump.

- *I could not find a solution to the problem using my design method, a redesign was needed.*

PROTOTYPE 3(28/12/18)

```
public float jumpforce;
public Rigidbody2D rb;
public bool grounded;
public int jumps;
//variables
```

```
void Start()
{
    grounded = false;
    rb = GetComponent<Rigidbody2D>();
    jumps = 2; //number of jumps declared.
}
```

```
void Update()
{

  if (Input.GetKeyDown(KeyCode.W) && jumps > 0) //if jumps is bigger than 0 and W has been pressed launch jump.
  {
      rb.velocity = Vector2.up * jumpforce;
      jumps--;  //number of jumps reduced.
  }

}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump	Player should be able to jump once then jump again while still in the air.	Passed
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being tested as this was an issue with the first original jump design.	Player should only be allowed to double jump, there should be no additional jumps until he falls back on the ground.	Passed

PROBLEMS

Problem	Problem Explained
DuJ3	The player can only preform 1 complete double jump and cannot jump afterwards.

Purple = problems discovered during testing.

JUMPING (WITH DOUBLEJUMP) – FINAL VERSION (04/01/19)

```
public float jumpforce;
public Rigidbody2D rb;
public bool grounded;
public int jumps;
//variables
```

```
void Start()
{
    grounded = false;
    rb = GetComponent<Rigidbody2D>();
    jumps = 2; //number of jumps declared.
}
```

```
void Update()
{

  if (Input.GetKeyDown(KeyCode.W) && jumps > 0) //if jumps is bigger than 0 and W has been pressed launch jump.
  {
      rb.velocity = Vector2.up * jumpforce;
      jumps--;  //number of jumps reduced.
  }

}
```

```
void OnTriggerEnter2D()
 // when the player and the ground touch
{
    grounded = true;
    jumps = 2;  //number of jumps reset after touching the ground.


}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump	Player should be able to jump once then jump again while still in the air.	Passed
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being tested as this was an issue with the first original jump design.	Player should only be allowed to double jump, there should be no additional jumps until he falls back on the ground.	Passed
DJu3	Player cannot jump again after preforming a double jump.	Pressing W twice to double jump then pressing it twice again to make the player jump again.	After completing a double jump the player should fall back on the ground and be able to double jump again.	Passed

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 3

Test Identification	Outcome/Solution

DJu3	The number of ***jumps*** was never reset after completing the jumps.
	
	After the player has touched the ground again, by checking with the ***Boxcollider2D***. In addition to turning ***grounded = true, jumps*** will be reset back to 2. Therefore, the player can jump again after falling on the ground.
	
	`void OnTriggerEnter2D()`
	`// when the player and the ground touch`
	`{`
	`grounded = true;`
	`jumps = 2; //number of jumps reset after touching the ground.`
	`}`

MOVING LEFT AND RIGHT

PROTOTYPE 1 (04/02/2019)

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 3;
    //variables given values.
}
```

```
public Rigidbody2D rb;
public int movespeed;
//variables declared
```

```
// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        rb.velocity = Vector2.right * movespeed;
        //when D is pressed the character will move to the right
        //movement force will depend on the movespeed variable

    }
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
LR1	Can the player move to the left?	Pressing the D key to move to the right.	Player should move to left when the D key is being held down.	Failed
LR2	Can the player move to the right	Pressing the A key to move to the left.	Player should move to left when the A key is being held down.	Untested
LR3	Can the player move right/left and jump at the same time?	Pressing W to jump, and while in the air pressing A or D to move to the left or right.	Pressing W and holding D or A.	Untested

- Tests LR2 and LR3 were untested due to the failure of LR1.

PROBLEMS

Problem	Problem Explained
LR4	Player has to repeatedly press D to move. You cannot hold the D button and move.

Purple = problems discovered during testing.

PROTOTYPE 2 (04/02/2019)

```
public Rigidbody2D rb;
public int movespeed;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 3;
    //variables given values.
}
```

```
void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        rb.velocity = Vector2.right * movespeed;
        //when D is pressed the character will move to the right
        //movement force will depend on the movespeed variable

    }
}
```

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 1

Test Identification	Outcome/Solution
LR1	Wrong syntax was being used. The syntax ***"getkeydown"*** means that the player has to keep pressing the button for code to run. Therefore, the player had to keep pressing D to make the character move.

	This was fixed by using the correct syntax ***"getkey".*** Which even if you hold it runs the code. `if (Input.GetKey(KeyCode.D))`

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
LR1	Can the player move to the left?	Pressing the D key to move to the right.	Player should move to left when the D key is being held down.	Passed
LR2	Can the player move to the right	Pressing the A key to move to the left.	Player should move to left when the A key is being held down.	Untested (not coded)
LR3	Can the player move right/left and jump at the same time?	Pressing W to jump, and while in the air pressing A or D to move to the left or right.	Pressing W and holding D or A.	Failed
LR4	Does the player need to keep hitting the button to make the character move	Pressing D and holding it.	Player should move to the right when the	Passed

PROBLEMS

Problem	Problem Explained
LR3	Player stutters when trying to move left and right when in the air.

PROTOTYPE 3 (07/02/19)

```
public Rigidbody2D rb;
public float movespeed;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 150f;
    //variables given values.
}
```

```
void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        rb.AddForce(Vector2.right * movespeed);
    }
    if (Input.GetKey(KeyCode.A))
    {
        rb.AddForce(Vector2.left * movespeed);
        //velocity is not reset
        //it adds to the velocity of the object
        //when D or A is pressed force will be added to the velocity in that direction.
    }
}
```

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 2

Test Identification	Outcome/Solution
LR3	The character stuttered during jumping and moving because of an syntax used. The velocity of the character was resetting every time the player pressed A or D therefore causing a stutter when trying to jump and move. *"rb.velocity = vector2 * movespeed"* This was fixed by using a command to add force to the velocity of the object instead of generating a new one.

Test Identification	Test	Input	Expected Outcome	Test Outcome
LR1	Can the player move to the left?	Pressing the D key to move to the right.	Player should move to left when the D key is being held down.	Passed
LR2	Can the player move to the right	Pressing the A key to move to the left.	Player should move to left when the A key is being held down.	Passed
LR3	Can the player move right/left and jump at the same time?	Pressing W to jump, and while in the air pressing A or D to move to the left or right.	Pressing W and holding D or A.	Passed
LR4	Does the player need to keep hitting the button to make the character move	Pressing D and holding it.	Player should move to the right when the	Passed

FINAL VERSION (10/02/2019)

```
public Rigidbody2D rb;
public float movespeed;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 150f;
    //variables given values.
}
```

```
void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        rb.AddForce(Vector2.right * movespeed);
    }
    if (Input.GetKey(KeyCode.A))
    {
        rb.AddForce(Vector2.left * movespeed);
        //velocity is not reset
        //it adds to the velocity of the object
        //when D or A is pressed force will be added to the velocity in that direction.
    }
}
```

USER FEEDBACK

Benjamin Hammond: I really like the movement it is very smooth and versatile. It feels nice that it moves. I think you should add a dash because sometimes just moving by itself is a bit dull and boring. With the dash you can add another way to move around and dodge enemies.

Isaac Woodward: Yea what Ben said is really solid add like a double tap to dash. It should only be a short dash though to keep the intensity of the game and not make it too easy for the player to dodge enemies.

PROTOTYPE 1 (07/02/2019) (DASHING)

```
public Rigidbody2D rb;
public float movespeed;
private bool dashr;
private bool dashl;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 150f;
    dashl = false;
    dashr = false;
    //variables given values.
}
```

```
// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        rb.AddForce(Vector2.right * movespeed);
        dashr = true;
        //after moving right a dash is avaliable
        if (Input.GetKeyDown(KeyCode.D) && true)
            //if d is pressed again and dash r = true
        {
            rb.MovePosition(new Vector2(rb.position.x, rb.position.y));
            //dash will launch
            dashr = false;
            //dash will reset
```

Test Identification	Test	Input	Expected Outcome	Test Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.	Failed
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.	Untested
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left	Untested

PROBLEMS

Problem	Problem Explained
DS1	Dash system does not work, the nested if statement/boolean method causes the dash to launch immediately instead of when it double presses.

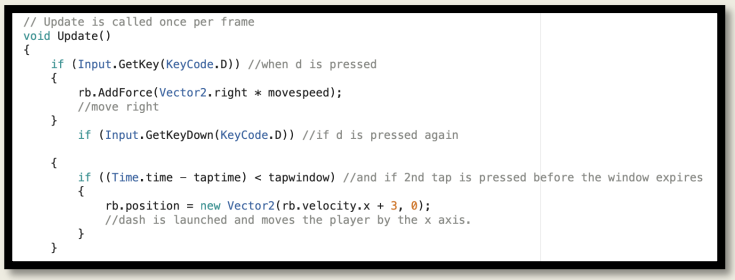
PROTOTYPE 2 (12/02/2019)

```
public Rigidbody2D rb;
public float movespeed;
private float tapwindow;
private float taptime;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 150f;
    tapwindow = 0.5f;
    taptime = 0;

    //variables given values.
}
```

```
// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D)) //when d is pressed
    {
        rb.AddForce(Vector2.right * movespeed);
        //move right
    }
        if (Input.GetKeyDown(KeyCode.D)) //if d is pressed again

    {
        if ((Time.time - taptime) < tapwindow) //and if 2nd tap is pressed before the window expires
        {
            rb.position = new Vector2(rb.velocity.x + 3, 0);
            //dash is launched and moves the player by the x axis.
        }
    }
```

SOLUTIONS/CHANGES TO PROBLEMS FROM PROTOTYPE 1

Test Identification	Outcome/Solution
DS2	The dash system worked for 1 dash and then stopped dashing afterwards. The design of waiting if the player has pressed D in a certain time window has worked. 

```
// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D)) //when d is pressed
    {
        rb.AddForce(Vector2.right * movespeed);
        //move right
    }
        if (Input.GetKeyDown(KeyCode.D)) //if d is pressed again

    {
        if ((Time.time - taptime) < tapwindow) //and if 2nd tap is pressed before the window expires
        {
            rb.position = new Vector2(rb.velocity.x + 3, 0);
            //dash is launched and moves the player by the x axis.
        }
    }
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.	Untested
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.	Partially Passed
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left	Failed

DS1 was untested as I only coded DS2 first as it is the same code with different inputs.

PROBLEMS

Problem	Problem Explained
DS2	After 1 dash the dash will not work afterwards.
DS3	The player cannot jump and dash at the same time. The player freezes in the air.

PROTOTYPE 3 (15/02/2019)

```
public Rigidbody2D rb;
public float movespeed;
private float tapwindow;
private float taptime;
//variables declared
```

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 150f;
    tapwindow = 0.5f;
    taptime = 0;

    //variables given values.
}
```

```
// Update is called once per frame
void Update()
{
    if (Input.GetKey(KeyCode.D)) //when d is pressed
    {
        rb.AddForce(Vector2.right * movespeed);
        //move right
    }
    if (Input.GetKeyDown(KeyCode.D)) //if d is pressed again

    {
        if ((Time.time - taptime) < tapwindow) //and if 2nd tap is pressed before the window expires
        {
            rb.position = new Vector2(rb.position.x + 5, rb.position.y);
            //dash is launched and moves the player by the x axis.
        }
        taptime = Time.time;
    }
```

SOLUTIONS/CHANGES FROM PROTOTYPE 2

Test Identification	Outcome/Solution

DS1	The ***taptime*** was not reset after the first tap. So, it was always missing the tap time window. This was fixed by resetting the tap time to the time of the game, so the window begins as soon as he presses D again.
	
	`}`
	`taptime = Time.time;`

DS3	I was resetting the y position of the player to 0 every dash, therefore the player could not jump and dash at the same time as the game forced the player back to position 0 on the y axis.
	
	This was fixed by setting the y position to the player's current y position instead of resetting it to 0.
	
	`rb.position = new Vector2(rb.position.x + 5, rb.position.y);`
	`//dash is launched and moves the player by the x axis and his y position remains the same`

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.	Untested
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.	Passed
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left	Passed

DS1 has not been tested as it has not been coded yet. It was to save time while trying to find a solution because the code for DS1 and DS2 are identical, the only difference being the input.

PROBLEMS

Problem	Problem Explained
DS4	The character falls over when it collides with enemies and the environment of the game. This makes the game unplayable as the player cannot get up and fight against the enemies.
DS5	The character does not face left when he is moving left. He is continuously facing to the right. This would cause the game to be incomplete and unbeatable as enemies from the left just kill the player while the player can't fight back.
DS6	Movement speed is uncapped. The longer you hold D or A the faster you go in that direction. This would make the game extremely difficult to play as the player would move from extremely fast to extremely slow.

Blue problems are problems that were pointed out by the end user.

JUMPING (WITH DASHING) – FINAL VERSION (22/02/2019)

```
public Rigidbody2D rb;
public float movespeed;
public float tapwindow;
private float taptime;
//variables declared
```

```
// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    movespeed = 7f;
    tapwindow = 0.5f;
    taptime = 0;
    //values given to variables

}
```

```
void Update()
{

if (Input.GetKey(KeyCode.D)) //if D is held down or pressed
        {
        transform.localScale = new Vector2(1f, 1f);
        //will face the player to the right
        rb.velocity = new Vector2(movespeed, rb.velocity.y);
        //carahcter will move to the left at a fixed velocity and same y position
    }
if (Input.GetKeyDown(KeyCode.D)) //if D is pressed again
        {
        if ((Time.time - taptime) < tapwindow) //if D is pressed again between the time window of 0.5 seconds
            {
                rb.position = new Vector2(rb.position.x + 3, rb.position.y);
                //character will dash to the right but remain in the same y position.
            }
            taptime = Time.time;
            //time window reset (dash reset)
        }
```

```
if (Input.GetKey(KeyCode.A)) //if A is held down or pressed
        {
        transform.localScale = new Vector2(-1f, 1f);
        //will face the player to the left

        rb.velocity = new Vector2(-movespeed, rb.velocity.y);
        //chracter will move to the left;
    }
if (Input.GetKeyDown(KeyCode.A)) //if A is pressed again
        {
          if ((Time.time - taptime) < tapwindow) //If a is pressed again between the time window of 0.5 seconds
            {
                rb.position = new Vector2(rb.position.x - 3, rb.position.y); //character will dash to the left.
            }
            taptime = Time.time; //time window reset (dash reset)
        }
```

```
    if (Input.GetKeyDown(KeyCode.H))

  {
      transform.rotation = Quaternion.Euler(0, 0, 0);
      //when H is pressed it will flip the character upright.
  }
```

SOLUTIONS/CHANGES FROM PROTOTYPE 3

Test Identification	Outcome/Solution

DS4	A new control was added, H. The player can now press H to flip the character upright again.
	`if (Input.GetKeyDown(KeyCode.H))`
	`{`
	`transform.rotation = Quaternion.Euler(0, 0, 0);`
	`//when H is pressed it will flip the character upright.`
	`}`

DS5	Added a command in each IF statement to flip the player to the left or right depending on what button is pressed.
	`if (Input.GetKey(KeyCode.D)) //if D is held down or pressed`
	`{`
	`transform.localScale = new Vector2(1f, 1f);`
	`//will face the player to the right`
	`rb.velocity = new Vector2(movespeed, rb.velocity.y);`
	`//carahcter will move to the left at a fixed velocity and same y position`
	`}`
	`if (Input.GetKey(KeyCode.A)) //if A is held down or pressed`
	`{`
	`transform.localScale = new Vector2(-1f, 1f);`
	`//will face the player to the left`
	`rb.velocity = new Vector2(-movespeed, rb.velocity.y);`
	`//character will move to the left at a fixed velocity and same y position`
	`}`

DS6	This was fixed by setting a fixed velocity with a new command. The previous command ***Addforce*** added force every time D was pressed, making the character go faster and faster as the button was held longer. ***Rb.velocity*** command keeps the player at a capped speed removing the problem.
	`rb.velocity = new Vector2(-movespeed, rb.velocity.y);`
	`//character will move to the left at a fixed velocity and same y position`
	`rb.velocity = new Vector2(movespeed, rb.velocity.y);`
	`//character will move to the left at a fixed velocity and same y position`

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.	Passed
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.	Passed
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left	Passed
DS4	Can the player flip back up?	Pressing H	Player should flip back up to upright position.	Passed
DS5	Does the player face the correct way it is walking?	Pressing D and A once	When D is pressed the player should face right and when A is pressed the player should face left.	Passed
DS6	Is the speed of movement capped?	Holding down D or A.	Player should move at a fixed rate not affected by the length of time D or A was held.	Passed

USER FEEDBACK

Benjamin Hammond: It looks good and feels good to move.
Isaac Woodward: Yea you've nailed it.

ENEMY AND ENEMY AI (24/02/2019) FINAL VERSION

FINAL VERSION

```
public class Enemy
{
    public int health;
    public float speed;
    public int points;
    public Transform target;
    //attributes of the class
}
```

Enemy object blueprint created.

```
 Enemy Lanky = new Enemy();
 //new object based on enemy class created
```

New object called lanky created

```
// Start is called before the first frame update
void Start()
{
    Lanky.health = 36;
    Lanky.points = 3;
    Lanky.speed = 5;
    Lanky.target = GameObject.FindGameObjectWithTag("player").GetComponent<Transform>();
    //new enemy object inherits attributes
    //attribuites given values
}
```

CHASE AI PROTOTYPE

```
// Update is called once per frame
void Update()
{
    |
        transform.position = Vector2.MoveTowards(transform.position, Lanky.target.position, Lanky.speed * Time.deltaTime);
        //enemy locates the player and follows the player
        //follows until it is within range of the player
        //or it cannot attack the player
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
CH1	Does the enemy chase to the left?	A	Player should move to the left and the enemy should follow the player.	Passed
CH2	Does the enemy chase to the right?	A	Player should move to the right and the enemy should follow the player.	Passed
CH3	Does the enemy chase the player if he jumps to the right or left?	A and W Or D and W	Player should jump to the left or right and the enemy should follow the player.	Passed

STAKEHOLDER FEEDBACK

Isaac: Its good but the monster should stop when it gets too close, because it will be hard for the player to attack the monster and the monster to hit the player.

Ben: Its good but the creature gets too close to the player.

PROBLEMS

Problem	Problem Explained
CH4	The creature follows the player too hard. It should stop at a certain distance to attack the player and allow the player to attack the creature.

ADRESSING STAKEHOLDER FEEDBACK CHASE AI (25/02/2019)

```
public class Enemy
{
    public int health;
    public float speed;
    public float distance;
    public int points;
    public Transform target;
    //attributes of the class
}
//enemy class declared
```

```
Enemy Lanky = new Enemy();
//new object based on enemy class created
```

```
// Start is called before the first frame update
void Start()
{
    Lanky.health = 36;
    Lanky.distance = 1;
    Lanky.points = 3;
    Lanky.speed = 5;
    Lanky.target = GameObject.FindGameObjectWithTag("player").GetComponent<Transform>();
    //new enemy object inherits attributes
    //attribuites given values
}
```

```
void Update()
{
    if (Vector2.Distance(transform.position, Lanky.target.position) > Lanky.distance)
    //if the player is further away than the stopping distnace
    {
        transform.position = Vector2.MoveTowards(transform.position, Lanky.target.position, Lanky.speed * Time.deltaTime);
        //enemy locates the player and follows the player
        //follows until it is within range of the player
        //or it cannot attack the player

    }
```

SOLUTIONS/CHANGES FROM THE PREVIOUS VERSION

Test Identification	Outcome/Solution

CH4	Adding a new variable called distance.
	`public int health;`
	`public float speed;`
	`public float distance;`
	`public int points;`
	`public Transform target;`
	`//attributes of the class`
	This variable ***distance*** will determine how far away the monster will from the players position.
	`if (Vector2.Distance(transform.position, Lanky.target.position) > Lanky.distance)`
	`//if the player is further away than the stopping distnace`
	`{`
	`transform.position = Vector2.MoveTowards(transform.positi...ky.speed * Time.deltaTi`
	`//enemy locates the player and follows the player`
	`//follows until it is within range of the player`
	`//or it cannot attack the player`
	`}`
	`(field) float Enemy.distance`
	Adding an if statement solved this problem. The creature will only chase the player if the player is further than the distance variable. If he is within distance variable the creature will not chase the player. Solving the problem.

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
CH1	Does the enemy chase to the left?	A	Player should move to the left and the enemy should follow the player.	Passed
CH2	Does the enemy chase to the right?	D	Player should move to the right and the enemy should follow the player.	Passed
CH3	Does the enemy chase the player if he jumps to the right or left?	A and W Or D and W	Player should jump to the left or right and the enemy should follow the player.	Passed
CH4	Does the enemy stop chase when the player is in distance?	A or D	If the player is within distance the creature should stop chasing the player.	Passed

STAKEHOLDER FEEDBACK

Isaac: Nice.

Ben: Cool looks good.

ATTACKING MOVE PROTOPTYPE 1 (27/02/2019)

COMMENTED CODE

```
public float cooldown;
public float cooldownlength;
public Transform attackarea;
public float attackrange;
public LayerMask badguy;
public int damage;
public Animator ani;
public bool attacking;
//variables for attacking
```

```
void Start()
  {
      ani = GetComponent<Animator>();
      //ani assigned to animator object.
  }
```

```
// Update is called once per frame
void Update()

{

    if (cooldown <= 0) //when cooldown reaches 0 or less than 0
    {
        if (Input.GetKey(KeyCode.J)) //if the player presses J
        {

            Collider2D[] KillEnemies = Physics2D.OverlapCircleAll(attackarea.position, attackrange, badguy);
            //creates the attack radius where objects are damaged.
            //this area only damages objects in the enemies layer mask
            for (int i = 0; i < KillEnemies.Length; i++)
            {
                KillEnemies[i].GetComponent<badguy>().hit(damage);
                //hit function is launched, enemies are damaged by the "damage variable"
                Debug.Log("HITTING ENEMIES");
                ani.Play("attacking");
                //plays the attacking animation
            }
            //will loop and damage enemies inside the cricle

        }
        cooldown = cooldownlength;
        //cooldown is reset so the player can attack again
    }
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
ATK1	Does pressing J launch an attack?	J	Pressing J should launch an attack.	Failed
ATK2	Does the attack do damage?	J	Player should do damage to enemies.	Partially Passed
ATK3	Can the player move or jump and attack at the same time?	Pressing J while pressing A, D or W	Player should be able to move and attack or jump and attack.	Untested

PROBLEMS

Problem	Problem Explained
ATK1	Pressing J does launch an attack. However, after pressing J again it doesn't launch the attack.
ATK2	Pressing J does do damage to the enemy. However, since the attack only launches once I don't know if it damages the enemy a second time.
ATK4	When pressing J, the attack animation launches once, and it loops forever. It is meant to play the animation once and stop.
ATK5	The cooldown doesn't decrease therefore the attack only launches once and cannot be launched again.

FINAL VERSION (28/02/2019)

```
public float cooldown;
public float cooldownlength;
public Transform attackarea;
public float attackrange;
public LayerMask badguy;
public int damage;
public Animator ani;
public bool attacking;
//variables for attacking
```

```
void Start()
  {
      ani = GetComponent<Animator>();
      //ani assigned to animator object.
  }
```

```
    if (cooldown <= 0) //when cooldown reaches 0 or less than 0
    {
        if (Input.GetKey(KeyCode.J)) //if the player presses J
        {
            ani.Play("attacking");
            //play attack animation
            Debug.Log("animation");
            Collider2D[] KillEnemies = Physics2D.OverlapCircleAll(attackarea.position, attackrange, badguy);
            //creates the attack radius where enemies are damaged
            for (int i = 0; i < KillEnemies.Length; i++)
            {
                KillEnemies[i].GetComponent<badguy>().hit(damage);
                //hit function is launched, enemies are damaged by the "damage variable"
                Debug.Log("HITTING ENEMIES");
            }

        }
        cooldown = cooldownlength;
        //cooldown is reset
    }
    else

    {
        cooldown -= Time.deltaTime;
        //cooldown time is reduced
    }
}
```

SOLUTION/CHANGES FROM PROTOTYPE 1

Test Identification	Outcome/Solution

ATK1 and ATK5	The cooldown value was never decrementing. So, the attack only launched once because the attack never came off cooldown.
	
	Adding and else statement and reducing the cooldown by ***Time.deltaTime;*** (game time)
	
	`else`
	`{`
	`  cooldown -= Time.deltaTime;`
	`  //cooldown time is reduced`
	`}`

ATK4	The statement that launches the animation was included inside the for loop therefore it kept looping. Moving it out the for loop stopped it repeating.
	
	`if (Input.GetKey(KeyCode.J)) //if the player presses J`
	`{`
	`  ani.Play("attacking");`
	`  //play attack animation`
	`  Debug.Log("animation");`
	`  Collider2D[] KillEnemies = Physics2D.OverlapCircleAll(attackarea.position, attackrange, badguy);`
	`  //creates the attack radius where enemies are damaged`
	`  for (int i = 0; i < KillEnemies.Length; i++)`
	`  {`
	`  KillEnemies[i].GetComponent<badguy>().hit(damage);`
	`  //hit function is launched, enemies are damaged by the "damage variable"`
	`  Debug.Log("HITTING ENEMIES");`
	`  }`

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
ATK1	Does pressing J launch an attack?	J	Pressing J should launch an attack.	Passed
ATK2	Does the attack do damage?	J	Player should do damage to enemies.	Passed
ATK3	Can the player move or jump and attack at the same time?	Pressing J while pressing A, D or W	Player should be able to move and attack or jump and attack.	Passed
ATK4	Does the animation only launch once?	J	The attack animation should only launch once	Passed
ATK5	Does the attack cooldown decrease?	J	The cooldown for the attack should decrease until it reaches 0 and allow the player to attack again.	Passed

ENEMY DEATH AND TAKING DAMAGE (02/03/2019)

FINAL VERSION (28/02/19)

```
if (Lanky.health <= 0) //when enemy health is below 0
{
    Destroy(gameObject);
}
```

```
public void hit(int damage)
    // when the player launches the attack it launches the hit function.
    // the hit function is used here to damage the enemy
{
    Lanky.health -= damage;
    //enemy is damaged by the player's damage variable
    Debug.Log("i took damage");
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
CDE1	Does the creature take damage when is it attacked?	J	The creature should lose HP	Passed
CDE2	Does the creature die after getting attacked?	J	Creature should die after its HP is 0 or below 0.	Passed
DE1	Does the enemy die in the game?	A and D to locate enemies. J to attack enemies.	Players should be able to kill enemies by attacking them. When the enemy HP is 0 or less it should die.	Passed

ENEMY CREATURE ATTACK MOVE (03/03/2019)

COMMENTED CODE

```
private void OnTriggerEnter2D(Collider2D col)
{
    Hleath.health -= 1;
    Debug.Log("die player");
    //if the player collides with the enemy
    //he will take damage.
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
EDMG1	Does the enemy damage the player	No Input Required	The enemy should attack the player and damage him by the damage variable.	Passed
EDMG2	Does the attack animation launch?	No Input Required	The enemy attack animation should play when the enemy attacks the player.	Untested
EDMG2	Does the enemy damage nearby allies?	No Input Required	The enemy should only be able to damage the player.	Passed

EDMG2 has not been tested as I have not made an animation to test the code.

SCORE SYSTEM PROTOTYPE 1 (05/03/2019)

COMMENTED CODE

```
  int scoreval;
  Text score;
  //variable that holds UI text
```

```
// Start is called before the first frame update
void Start()
{
    score = GetComponent<Text>();
    //variable assigned to the text on UI.
}
```

```
void Update()
{
    score.text = "Score: " + scoreval;
    //the text changes to display the cuerrent score on the UI.
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Testing
SC1	When an enemy dies does it award the player points?	A and D to locate enemies. J to attack enemies.	The player should be awarded 10 points.	Failed
SC2	Does the UI give the current score of the player.	A and D to locate enemies. J to attack enemies.	The UI should display the score of the enemy. It should updates its self in real time when he gets more points.	Passed

PROBLEMS

Problem	Problem Explained
SC1	The score variable is inaccessible therefore I cannot change the score variable from the enemy script to award the player points.
SC3	The score value doesn't reset when the game is started again. It has the old score value scored.

Purple problems are problems found during testing

SCORE SYSTEM FINAL VERSION (05/03/2019)

COMMENTED CODE

```
public static int scoreval = 0;
//global variable declared;
Text score;
//variable that holds UI text
```

```
// Start is called before the first frame update
void Start()
{
    score = GetComponent<Text>();
    //variable assigned to the text on UI.
}
```

```
// Update is called once per frame
void Update()
{
    score.text = "Score: " + scoreval;
    //the text changes to display the cuerrent score on the UI.
}
```

```
if (Lanky.health <= 0) //when enemy health is below 0
{
    Destroy(gameObject);
    Score.scoreval = Score.scoreval + 10;
    //then enemy is destroyed and points are awarded to the player.
}
```

SOLUTIONS FROM PROTOTYPE 1

Test Identification	Outcome/Solution

SC1	The variable was inaccessible because it was a private variable locked into that code. Therefore, I could not access it from the enemy death script and award the player points by changing the variable. This was fixed by changing the variable into a global variable.
	
	`public static int scoreval = 0;`
	`//global variable declared;`
	`Text score;`
	`//variable that holds UI text`

SC2	The score did not reset because I did not reset the score value when the game started. When the variable is declared changing the variable value back to 0 fixed the problem
	
	`public static int scoreval = 0;`
	`//global variable declared;`
	`Text score;`
	`//variable that holds UI text`

TESTING

Test Identification	Test	Input	Expected Outcome	Testing
SC1	When an enemy dies, does it award the player points?	A and D to locate enemies. J to attack enemies.	The player should be awarded 10 points.	Passed
SC2	Does the UI give the current score of the player?	A and D to locate enemies. J to attack enemies.	The UI should display the score of the enemy. It should update its self in real time	Passed

			when he gets more points.	
SC3	When the game is restarted the score should be reset.	Left Click to start the game again.	The score should reset back to 0.	Passed

HEALTH SYSTEM PROTOTYPE 1 (08/03/2019)

COMMENTED CODE

```
public static int health;
//global variable health
public int HP;
public int numofhearts;
public Image[] hearts;
//array of images
public Sprite fullheart;
public Sprite emptyheart;
//variables declared
```

```
void Start()
 {
     health = 10;
     Debug.Log("health = 10");
     //health set to 10 as the game starts
 }
```

```
// Update is called once per frame
void Update()
{
    if (health == 0) // if health = 0
    {
        Destroy(gameObject);
        //player will die
    }

    for (int i = 0; i < hearts.Length; i++)//for loop
    {

        if (i < health) //if the counter is less than health
        {
            hearts[i].sprite = fullheart;
            //it will display a full heart. As player stll has HP remaining.
        }
        else //if the counter is more than health
             //player has lost health
        {
            hearts[i].sprite = emptyheart;
            //it will display a empty heart. As player has lost HP.
        }
    }
}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
HRT1	Is the max number of hearts = to the max health?	No Input Required.	There should be 10 hearts to display the 10 health the player has.	Passed
HRT2	Do the hearts turn empty after taking damage?	No input required.	Depending on how much damage the player takes. The number of hearts that turn empty should equal the amount of damage he took.	Passed
HRT3	Do the hearts fill up when a powerup restores health of the player.	A and D. Must move and collide with an ambrosia. To restore health.	Hearts should fill up on the amount the ambrosia heals the player.	Untested

HRT3 is untested because the game has no powerups.

USER FEEDBACK

Isaac: Sometimes your character doesn't die even though it has no hearts left

Ben: Oh yea, what Isaac said. I didn't realize.

PROBLEMS

Problem	Problem Explained
HRT4	If the max health is changed then the number of hearts is till 10.
HRT5	Sometimes the player doesn't die even though he has no hearts remaining.

Blue problems are problems mentioned by the stakeholders

HEALTH SYSTEM FINAL VERSION (09/03/2019)

COMMENTED CODE

```
public static int health;
//global variable health
public int HP;
public int numofhearts;
public Image[] hearts;
//array of images
public Sprite fullheart;
public Sprite emptyheart;
//variables declared
```

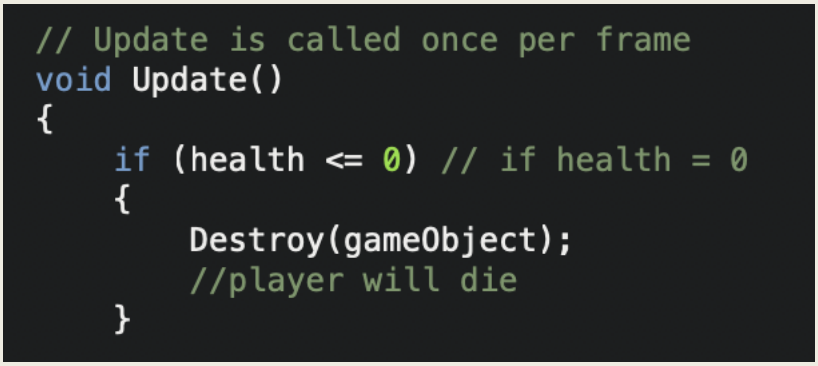
```
void Start()
 {
     health = 10;
     Debug.Log("health = 10");
     //health set to 10 as the game starts
 }
```

```
// Update is called once per frame
void Update()
{
    if (health <= 0) // if health = 0
    {
        Destroy(gameObject);
        //player will die
    }


    for (int i = 0; i < hearts.Length; i++)//for loop
    {

        if (i < health) //if the counter is less than health
        {
            hearts[i].sprite = fullheart;
            //it will display a full heart. As player stll has HP remaining.
        }
        else //if the counter is more than health
            //player has lost health
        {
            hearts[i].sprite = emptyheart;
            //it will display a empty heart. As player has lost HP.
        }

        if (i < numofhearts)
        {
            hearts[i].enabled = true;
            //if the counter is less than number of hearts then it will display a heart
            //the code has not reached the maxiumum HP.
            //it will display the hearts as it checks if it has reached max HP
        }
        else
        {
            hearts[i].enabled = false;
            //if the counter is larger than the number of hearts it shows that the counter has reached max hp.
            //Therefore the rest of the hearts are disbaled and not displayed.
        }
    }
}
```

SOLUTIONS TO THE PROBLEM FROM PROTOTYPE 1

Test Identification	Outcome/Solution
HRT4	This was fixed by adding new if statements that would enable hearts below the max HP and disable hearts above the max HP.

```
if (i < numofhearts)
{
    hearts[i].enabled = true;
    //if the counter is less than number of hearts then it will display a heart
    //the code has not reached the maxiumum HP.
    //it will display the hearts as it checks if it has reached max HP
}
else
{
    hearts[i].enabled = false;
    //if the counter is larger than the number of hearts it shows that the counter has reached max hp.
    //Therefore the rest of the hearts are disbaled and not displayed.
}
```

HRT5	Sometimes the player could take multiple damages from different enemies. Therefore, would take extra damage that would make HP less than 0. However, my code only kills the player if the health is 0. This was changed with the code below to make sure the player dies if his HP is less or equal to 0. 

```
// Update is called once per frame
void Update()
{
    if (health <= 0) // if health = 0
    {
        Destroy(gameObject);
        //player will die
    }
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
HRT1	Is the max number of hearts = to the max health?	No Input Required.	There should be 10 hearts to display the 10 health the player has.	Passed
HRT2	Do the hearts turn empty after taking damage?	No input required.	Depending on how much damage the player takes. The number of hearts that turn empty should equal the amount of damage he took.	Passed
HRT3	Do the hearts fill up when a powerup	A and D. Must move and collide with an	Hearts should fill up on the amount the	Untested

	restores health of the player.	ambrosia. To restore health.	ambrosia heals the player.	
HRT4	The hearts should display the max HP not anything more.	No input required	Number of hearts displayed should equal the max HP	Passed
HRT5	Does the player die when his hearts run out?	No Input Required	Player should die when he has no hearts remaining.	Passed

WAVE SYSTEM FINAL VERSION (13/03/2019)

COMMENTED CODE

```
public enum SpawnState { SPAWNING, WAITING, COUNTING };
//states declared
[System.Serializable]
//allows coroutines in Unity
public class Wave
{
    public string name;
    public Transform enemy;
    public int count;
    public float rate;
    //object wave created
}
public Wave[] waves;
//array of wave objects created
private int nextWave = 0;
public int NextWave
{
    get { return nextWave + 1; }
}
//this variable value nextwave + 1
public Transform[] spawnPoints;
//variables for spawn points created
public float timeBetweenWaves = 5f;
//the time the player has to rest befor the next wave starts
private float waveCountdown;
//the countdown the wave gives
public float WaveCountdown
{
    get { return waveCountdown; }
}
private float searchCountdown = 1f;
//searchcountdown variable declared
private SpawnState state = SpawnState.COUNTING;
public SpawnState State
{
    get { return state; }
}
```

```
void Start()
{
    if (spawnPoints.Length == 0)
    {
        Debug.LogError("there are no spawnpoints");
    }
    //checks if there are spawn points to spawn enemies.

    waveCountdown = timeBetweenWaves;
    //wave countdown is declared
}
```

```
 void Update()
 {
     if (state == SpawnState.WAITING) //if the state is waiting (waiting for the player to finish the wave)
     {
         if (!EnemyIsAlive()) //if there are no more enemies on the map then
         {
             WaveCompleted();
             //run wavecompleted function
         }
         else
         {
             return;
             //do nothing
         }
     }

     if (waveCountdown <= 0)
         //if wavecountdown has finished
     {
         if (state != SpawnState.SPAWNING)
             //if the state is not equal to spawnning
         {
             StartCoroutine( SpawnWave ( waves[nextWave] ) );
             //run coroutine with the next wave in the array.
         }
     }
     else //if the coutndown hasnt finished
     {
         waveCountdown -= Time.deltaTime;
         //keep counting down
     }
 }
```

```
void WaveCompleted()
    //function that handles the game when the wave is completed
{
    Debug.Log("Wave Completed!");

    state = SpawnState.COUNTING;
    waveCountdown = timeBetweenWaves;
    //starts giving the player time to rest.

    if (nextWave + 1 > waves.Length - 1)
        //if next wave is bigger than the number of waves in the array
    {
        nextWave = 0;
        Debug.Log("looping game");
        //there are no more waves remaining
        //the game will loop starting from wave 0
    }
    else //if not
    {
        nextWave++;
        // it goes onto the next wave in the array
    }
}
```

```
  bool EnemyIsAlive()
      //function that checks if there are any enemies alive from the previous wave
  {
      searchCountdown -= Time.deltaTime;
      //countdown decrements
      if (searchCountdown <= 0f)
          //if the countdown reaches 0
      {
          searchCountdown = 1f;
          //countdown is increased, gives more time to search for the enemy.
          if (GameObject.FindGameObjectWithTag("badguy") == null)
          //if the game finds an enemy
          {
              return false;
              // it will return false
          }
      }
      return true;
      //otherwise it will return true
  }
```

```
IEnumerator SpawnWave(Wave _wave)
{
    Debug.Log("Spawning Wave: " + _wave.name);
    //broadcasts what wave the player is on
    state = SpawnState.SPAWNING;
    //state is changed to spawnning

    for (int i = 0; i < _wave.count; i++)
    //loop that spawns the enemy in the wave object
    {
        SpawnEnemy(_wave.enemy);
        //runs another routine with the enemy with the wave object
        yield return new WaitForSeconds( 1f/_wave.rate );
        //uses the rate from the wave object to increase or decrease the rate of spawnning
    }

    state = SpawnState.WAITING;

    yield break;
    //after spawnning it changes the game state to waiting
}
```

```
  void SpawnEnemy(Transform _enemy)
      //routine that spawns enemies
  {
      Debug.Log("Spawning Enemy: " + _enemy.name);

      Transform _sp = spawnPoints[ Random.Range (0, spawnPoints.Length) ];
      Instantiate(_enemy, _sp.position, _sp.rotation);
      //at the spawn point it spawns new enemies every time it called.
      //the point at where it spawns is random
  }

}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
WAV1	Does the system spawn enemies.	No Input Required	The system should spawn the set amount of enemies for that wave.	Passed

WAV2	Does the system spawn more enemies at a faster rate as the wave number increases?	No Input Required	The speed of spawning and the number of enemies should increase as wave number increases.	Passed
WAV3	Does it go past wave 10	No Input Required	The wave number should not go past 10. It should stop and display the winners screen.	Partial Fail

WAV3 was a partial failure. The system was changed, instead of the surviving 10 waves the player must survive for as long as he can in an endless wave of enemies. The wave system will loop continuously and will not stop if the player has beat wave 10.

USER FEEDBACK

Isaac: It is very good, well worked!

Ben: It is well coded, it's cool.

MENU SYSTEM FINAL VERSION (14/03/2019)

COMMENTED CODE

```
    public void LaunchHTP()
    {
        Debug.Log("going to the howtoplay›");
        SceneManager.LoadScene("htp");
        Debug.Log("went to how to play");
    }
//launches the How To Play menu screen

    public void LaunchGame()
    {
        Score.scoreval = 0;
        SceneManager.LoadScene("Game");

    }
//launches the game


    public void LaunchLeaderboards()
    {
        Debug.Log("going to lb");
        SceneManager.LoadScene("lb");

    }
//launches the Leaderboards



    public void LaunchMenu()
    {
        Debug.Log("going to the menu");
        SceneManager.LoadScene("Menu");

    }
//launches the Main Menu screen
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
MNU1	Can you traverse to all menu screens?	Left Click on buttons.	The player should be able to access all menu screens from the main menu if the corresponding button is pressed.	Passed
MNU2	Does the back-button work?	Left Click on the back button.	After reaching another screen the player should return to the main menu if he clicks the back button.	Passed

LEADERBOARD PROTOTYPE 1 (17/03/2019)

COMMENTED CODE

```
public Text score;
public Text highscore;
public Text best;
public Text playername;
//variables declared
//all variables are UI text
```

```
//updates every frame
void Update()
{
    int hs = Score.scoreval;
    //integer that stores the current score
    score.text = "Your score: " + hs.ToString();
    //displays your current score
    playername.text = usernamemanager.username;
    //displays your current username

    if (hs > PlayerPrefs.GetInt("Highscore",0))
    //if the current score is higher than the highscore
    {
        PlayerPrefs.SetInt("Highscore", hs);
        highscore.text = "Highscore: " + hs.ToString();
        //externally stored score is replaced
        //new highscore is displyaed

        PlayerPrefs.SetString("thebest", usernamemanager.username);
        best.text = usernamemanager.username;
        //externally stored username is replaced
        //new username is displayed
    }

}
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
LED1	Are the bestscore and the thebestplayer variables saved externally?	No input required.	These variables should be stored externally in a file so that they can be used when the game is reopened.	Passed
LED2	Does the leaderboards display the	No input required.	The screen should display the externally	Passed

	externally stored variables.		stored variables. This is done so that we can see the Best Player's name and the score.	
LED3	If the high-score is beaten are the new details saved and the old ones removed?	A and D to locate enemies. J to attack enemies. Must beat the old high-score.	The game should replace the old high-score details and replace it with the current player's details.	Passed

USER FEEDBACK

Isaac: Pretty good compromise.

Ben: You should add a reset high-score button to reset the score when the game is copied onto multiple devices so high-score is only on one machine

LEADERBOARD FINAL VERSION (19/03/2019) ADDRESISNG STAKEHOLDER FEEDBACK

COMMENTED CODE

```
    public void resetscores()
    {
        PlayerPrefs.SetInt("Highscore", 0);
        PlayerPrefs.SetString("thebest", null);
    }
//resets highscore values
```

TESTING

Test Identification	Test	Input	Expected Outcome	Test Outcome
LED4	Does the high-score values reset when the reset button is pressed	Left Click on Button	The high-score variables should be reset.	Passed

D. EVALUATION

COMPARING SOLUTION TO THE SUCCESS CRITERIA

CRITERIA POINT

Menu with 3 options 1. Play 2. How to Play 3. Leaderboards	To allow players to navigate through information before the game. For example, to look at the controls of the game before they play, or to look at current records in the leaderboard section.

I have met the criteria point

EVIDENCE OF MEETING SUCCESS CRITERIA

Test Identification	Test	Input	Expected Outcome	Test Outcome
MNU1	Can you traverse to all menu screens?	Left Click on buttons.	The player should be able to access all menu screens from the main menu if the	Passed

			corresponding button is pressed.	
MNU2	Does the back-button work?	Left Click on the back button.	After reaching another screen the player should return to the main menu if he clicks the back button.	Passed

HOW EVIDENCE SHOWS THE CRITERIA HAS BEEN MET?

The test data MNU1 and MNU2 shows that the button worked when they were tested, additionally screenshot evidence shows the leaderboards button light up when the cursor is hovered above it. Showing that the button when clicked will launch the script that will take it to its corresponding page.

EVIDENCE OF USABILITY

JUSTIFICATION OF USABILITY

The menus were a success, as they do what they are planned to do. The buttons lead to the corresponding page and a back button is available in each page that can be used the return to the menu screen. Additionally, they have been glamorised to fit the purple and gloomy theme of the game, which makes the game look more appealing and polished.

CRITERIA POINT

Simple control scheme. • WASD for movment. • J for fast attacks. • K for heavy attacks. • D for blocks.	Interviewees have said that complex controls can frustrate and irritate people. It is better for it to be easy so new players can jump right in to the game and familiarize themselves of the controls.

I have partially met this success criteria. As the game does have a simple control scheme, however there are no heavy attacks or blocking included.

EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABILITY

Test Identification	Test	Input	Expected Outcome	Test Outcome
LR1	Can the player move to the left?	Pressing the D key to move to the right.	Player should move to left when the D key is being held down.	Passed
LR2	Can the player move to the right	Pressing the A key to move to the left.	Player should move to left when the A key is being held down.	Passed
LR3	Can the player move right/left and jump at the same time?	Pressing W to jump, and while in the air pressing A or D to move to the left or right.	Pressing W and holding D or A.	Passed

LR4	Does the player need to keep hitting the button to make the character move	Pressing D and holding it.	Player should move to the right when the	Passed

Test Identification	Test	Input	Expected Outcome	Test Outcome
DS1	Can the player dash to the left?	Double Tap A	Player should dahs to the left.	Passed
DS2	Can the player dash to the right?	Double Tap D	Player should dash to the right.	Passed
DS3	Can the player move right/left and jump at the same time?	Double Tapping D or A while jumping.	Player should jump and dash towards the right or left	Passed
DS4	Can the player flip back up?	Pressing H	Player should flip back up to upright position.	Passed
DS5	Does the player face the correct way it is walking?	Pressing D and A once	When D is pressed the player should face right and when A is	Passed

			pressed the player should face left.	
DS6	Is the speed of movement capped?	Holding down D or A.	Player should move at a fixed rate not affected by the length of time D or A was held.	Passed

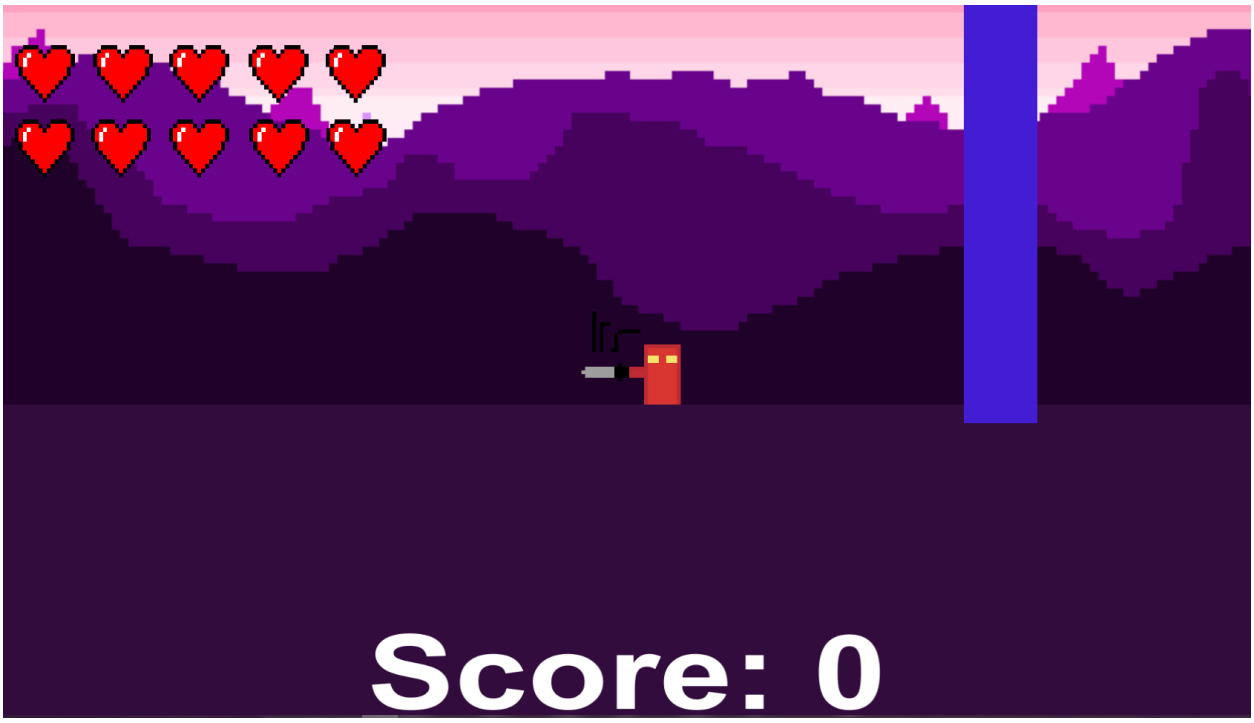
Test Identification	Test	Input	Expected Outcome	Test Outcome
Ju1	Jumping	W to jump, to see if the player can jump.	The character should jump when W is pressed.	Passed
Ju2	Falling	W to jump to see if the player falls after jumping.	The character should fall back down after jumping.	Passed
Ju3	Sensible Gravity Pulling Force	W to jump, to allow for gravity to work.	The pull force of gravity after the character has completed its jump should be sensible. It should not be too fast or too slow.	Passed
Ju4	Pressing other buttons	W to jump and any other buttons.	Jump should still occur regardless the fact that another button was pressed.	Passed
Ju5	Debug Log Spam	W to jump.	Debug lines should only be	Passed

			posted once and not spam the log.	
Ju6	Infinite Jump	Repeatedly press W	Player should only be able to jump once.	Passed

Test Identification	Test	Input	Expected Outcome	Test Outcome
DJu1	Double Jumping	Press W to jump once and then press W again to double jump	Player should be able to jump once then jump again while still in the air.	Passed
DJu2	Infinite Jump	Pressing W repeatedly to check if the character can infinitely jump. This is being tested as this was an issue with the first original jump design.	Player should only be allowed to double jump, there should be no additional jumps until he falls back on the ground.	Passed
DJu3	Player cannot jump again after preforming a double jump.	Pressing W twice to double jump then pressing it twice again to make the player jump again.	After completing a double jump the player should fall back on the ground and be able to double jump again.	Passed

Test Identification	Test	Input	Expected Outcome	Test Outcome
ATK1	Does pressing J launch an attack?	J	Pressing J should launch an attack.	Passed
ATK2	Does the attack do damage?	J	Player should do damage to enemies.	Passed
ATK3	Can the player move or jump and attack at the same time?	Pressing J while pressing A, D or W	Player should be able to move and attack or jump and attack.	Passed
ATK4	Does the animation only launch once?	J	The attack animation should only launch once	Passed
ATK5	Does the attack cooldown decrease?	J	The cooldown for the attack should decrease until it reaches 0 and allow the player to attack again.	Passed

(Player Jumping above) (player double jumping below)



(player moving/dashing and attack)

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN PARTIALY MET

The test series LR, Ds, Ju, DJu and ATK shows that we have partially met the success criteria. LR and DS shows that A and S keys were used to move the character. It uses the same buttons the criteria mentioned. Test series Ju and DJu shows that the W key was used to initiate the jump and meet the criteria. Lastly, test series ATK shows that J was used to launch the attack.

Furthermore, the screenshots show the inputs in game as they occur. We can see the player is jumping as he is off the ground or we can see him double jumping because he is way higher than the ground. Additionally, we can see the player attacking as the player is in his attack animation, with his sword out, we can also see that he has dashed/moved to the edge of the map as there is a blue wall that signals that the player has reached the end of the map.

JUSTIFICATION OF USBALITY

Although the success criteria point was only partially met, the usability features were a success. As it is very easy to learn how to play the game and with the additional features such as double jump and dashing it provides skilful players to exceed and stand out from the average player.

LIMITATIONS AND UNMET USABILITY FEATURES

- Heavy attacks and blocking could be added in the future. This would be increasing the skill ceiling of players, as there are additional features that skilful players can use to survive and get the best score. This can be addressed in further development.
- Additionally, dashing creates a limitation in the game. Due to the code of dashing the player can dash outside the boundaries of the map (the blue wall) and cheat to get more points easily.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- The dashing across the boundaries can be fixed by altering the dash script. Currently the dash script teleports the player to the dashed location, this can be changed to speed him up to the location not teleport him. Therefore, the player cannot teleport outside the boundaries as the dash only speeds him up.
- Blocking can be added using IF statements and creating a new animation for blocking. Essentially it will the same as attacking but instead of creating an area of damage it creates an area where damage is blocked.
- Heavy attacks is the same as fast attacks but with changes. The animation should be slower, the damage of the attack should be higher, and the cooldown time of the heavy attack should be higher.

CRITERIA POINT

1 Map

Why?

The game will be set on 1 map. There will be no additional maps due to the game being a small-scale project.

The criteria point has been met. The game only features 1 map.

EVIDENCE OF MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABILITY

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN MET

The screenshot shows the only map on the game currently.

JUSTIFICATION OF USBAILITY

The success criteria point was met, and the usability feature was a success, as the map matches the theme of the purple/gloomy theme of the game. Furthermore, it adds a sense of fear into players as the creatures begin to hunt them.

CRITERIA POINT

Challenging and Hard • Only a some players should get scores above 100.	The game will be challenging and hard as its target audience are teenagers who have previously played games before. Additionally, the increased difficulty will create an intense situation. It will generate intense emotions to be displayed by the player and will create a sense of reward, pride and satisfaction for completing the game.

The success criteria has been met.

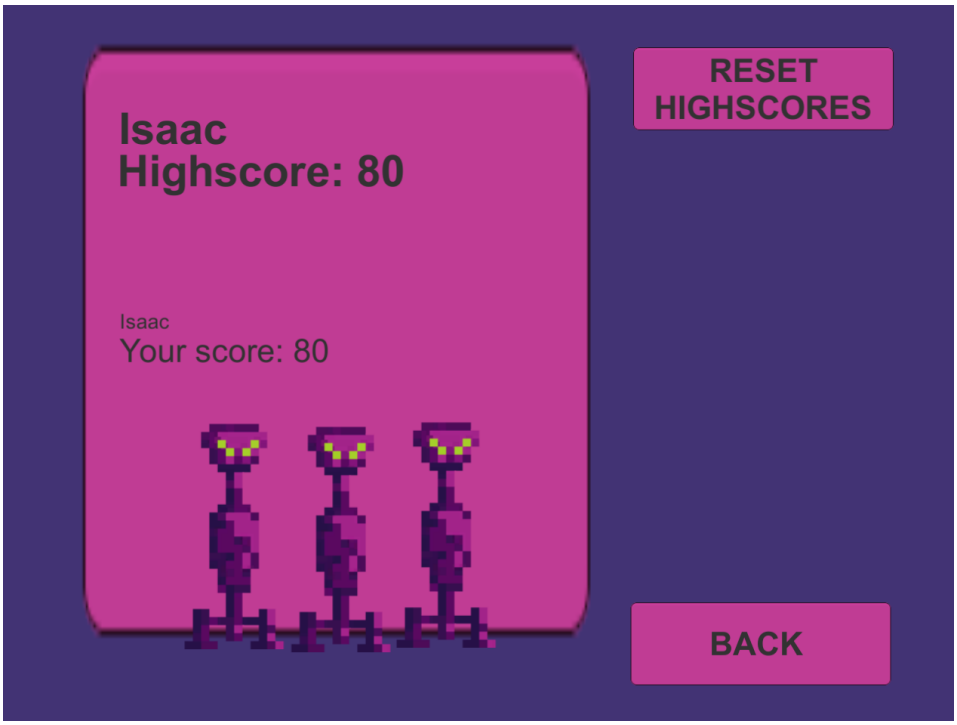
EVIDENCE OF MEETING THE SUCCESS CRITERIA

INTERVIEW TRANSCRIPT

Q: Is the game challenging and hard?

Ben: Yeah it is. Its very hard to cope with the number of enemies and the speed of enemies.

Isaac: It takes a lot of skill and a quick reaction time to try dodge enemies and survive past wave 3. That's why the high-score is so low, everyone keeps dying because the game is so difficult.



HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN MET

Interview transcript shows that the success criteria has been met as both stakeholders have expressed how the game is very difficult. Additionally, the screenshot of the leader-board screen shows that the high-score is very low indicating that the game is difficult as it is very hard to score points.

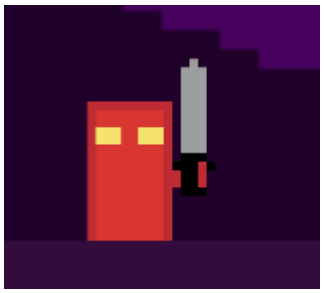
CRITERIA POINT

8-Bit Cartoonish Graphics	Allows low end system to run the game.

.

The success criteria point has been met.

EVIDENCE OF MEETING THE SUCCESS CRITERIA



INTERVIEW TRANSCRIPT

Q: Does the game have an 8-bit retro style?

Ben: Obviously, look at your characters you can see pixels/squares that make up your enemy, background and main character.

Isaac: Yep it looks decent!

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN MET

The interview with stakeholders shows that they believe the game is in the 8-bit style. Furthermore, the screenshots show further evidence that the game is in the 8-bit style as the enemy character, main character and map have distinct pixels/squares that make them up.

CRITERIA POINT

2 Passive and Active Powerups	Allows the player new tools to fight the enemies. Makes the game interesting for players.

The criteria were not met.

LIMITATIONS AND UNMET USABILITY FEATURES

There are no active or passive powerups in the game as I had no time to include them in the game.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- Sprites that spawn on the map that the player can collide with to increase HP points or damage enemies currently on the map. When the player collides with the power up it disappears.
- Passive power ups that can be selected before the game starts. It can speed up the player, or make the player do more damage, or reduce the cooldown of attacks.

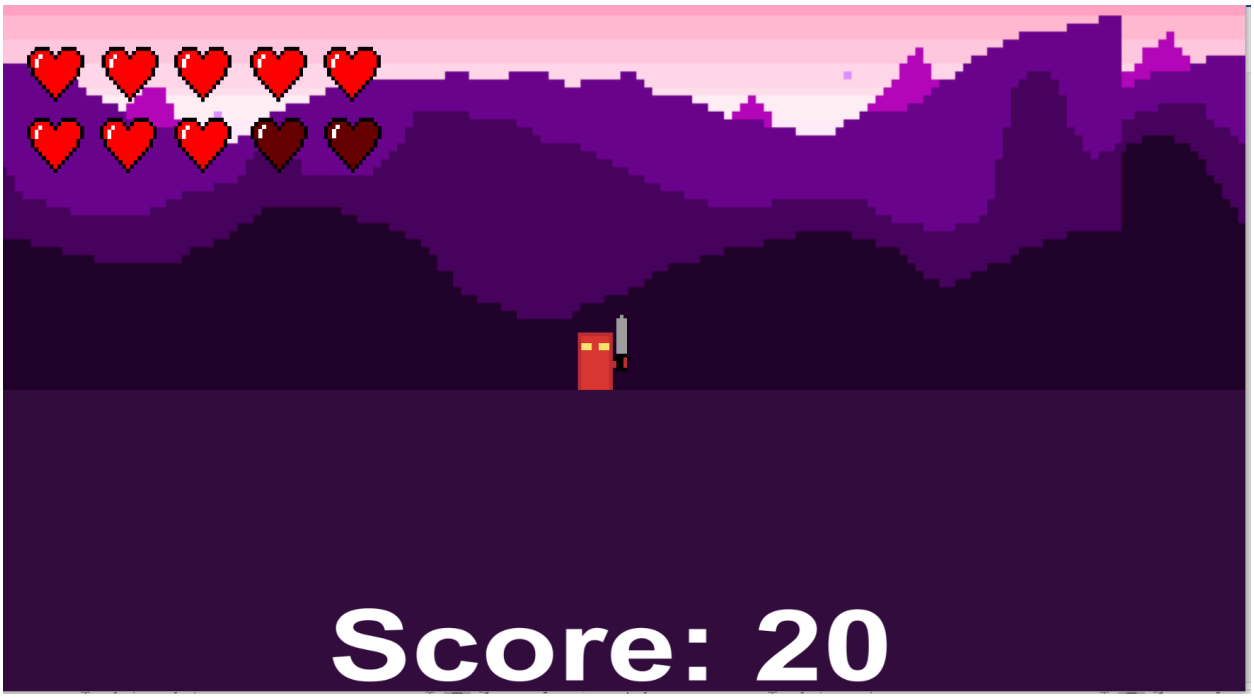
CRITERIA POINT

Point System/Combo Meters • Only some players should manage to get over 100 points	Give a chance for players to evaluate their performance in game to allow room for improvement. Also included to create a leader boards table.

The criteria point was partially met.

EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABLITY

Test Identification	Test	Input	Expected Outcome	Testing
SC1	When an enemy dies, does it award the player points?	A and D to locate enemies. J to attack enemies.	The player should be awarded 10 points.	Passed
SC2	Does the UI give the current score of the player?	A and D to locate enemies. J to attack enemies.	The UI should display the score of the enemy. It should update its self in real time when he gets more points.	Passed
SC3	When the game is restarted the score should be reset.	Left Click to start the game again.	The score should reset back to 0.	Passed



(screenshot showing that the player has earned 20 points)

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN MET

The test series SC shows that the points system has been tested and it works. Additionally, the screenshot shows that the points systems work as the HUD shows the player has 20 points currently.

JUSTIFICATION OF USABILITY

The success criteria point was met, and the usability feature is a success as the point system updates in real time as the player earns more points as he progresses through the game.

LIMITATIONS AND UNMET CRITERIA

- I was not able to add combo meters that awarded players with more points if they were using their attacks in combos. This was because I did not have the skill to implement this feature.

CRITERIA POINT

Wave System	System that spawns waves of enemies and stops after 10.
AI enemies that follow the player around the map and a changeable speed.	It is a fighting game; the player must be attacked by AI controlled enemies. Speed must be changeable to increase the difficulty of the game.

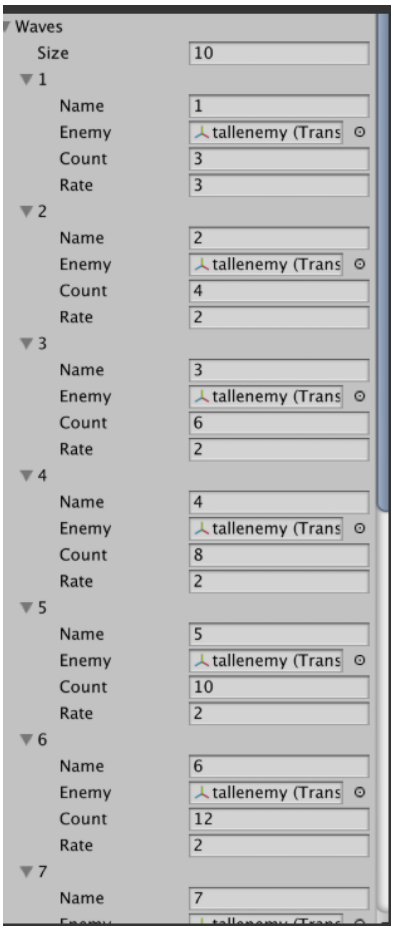
The criteria point was partially met

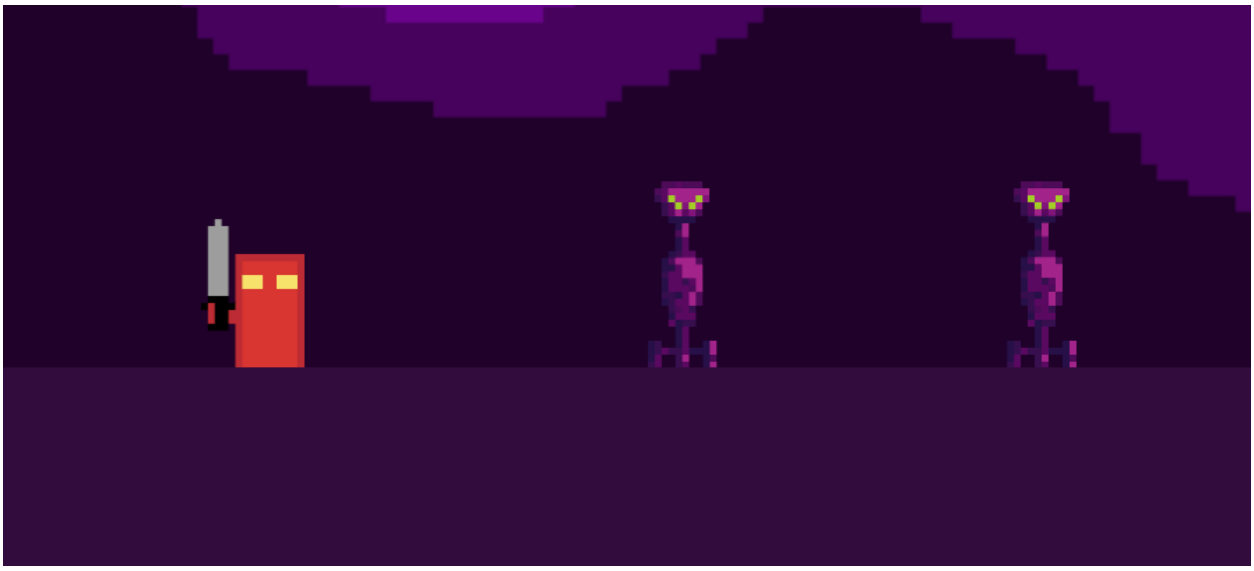
EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABLITY

Test Identification	Test	Input	Expected Outcome	Test Outcome
WAV1	Does the system spawn enemies.	No Input Required	The system should spawn the set amount of enemies for that wave.	Passed
WAV2	Does the system spawn more enemies at a faster rate as the wave number increases?	No Input Required	The speed of spawning and the number of enemies should increase as wave number increases.	Passed
WAV3	Does it go past wave 10	No Input Required	The wave number should not go past 10. It should stop and display the winners screen.	Partial Fail

WAV3 was a partial failure. The system was changed, instead of the surviving 10 waves the player must survive for as long as he can in an endless wave of enemies. The wave system will loop continuously and will not stop if the player has beat wave 10.

Test Identification	Test	Input	Expected Outcome	Test Outcome
CH1	Does the enemy chase to the left?	A	Player should move to the left and the enemy should follow the player.	Passed
CH2	Does the enemy chase to the right?	D	Player should move to the right and the enemy should follow the player.	Passed
CH3	Does the enemy chase the player if he jumps to the right or left?	A and W Or D and W	Player should jump to the left or right and the enemy should follow the player.	Passed
CH4	Does the enemy stop chase when the player is in distance?	A or D	If the player is within distance the creature should stop chasing the player.	Passed


Waves
Size 10
1
Name 1
Enemy tallenemy (Trans
Count 3
Rate 3
2
Name 2
Enemy tallenemy (Trans
Count 4
Rate 2
3
Name 3
Enemy tallenemy (Trans
Count 6
Rate 2
4
Name 4
Enemy tallenemy (Trans
Count 8
Rate 2
5
Name 5
Enemy tallenemy (Trans
Count 10
Rate 2
6
Name 6
Enemy tallenemy (Trans
Count 12
Rate 2
7
Name 7



HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN PARTIALLY MET

Test series WAV shows that the WAV system was tested, and it works. Furthermore, screenshot from the unity development program shows the waves and the number of waves. Additionally, it shows the number of enemies that will spawn during the wave and the rate at which the enemies spawn.

Test series CH shows that the enemy AI chase system was tested and works. Additionally in the later in-game screenshot it shows that the player is being chased by the enemies.

JUSTIFICATION OF USABILITY

The success criteria point was met, and the usability feature is a success as the point system updates in real time as the player earns more points as he progresses through the game. Furthermore, the AI chase system works as the enemy creatures hunt the players as they spawn.

LIMITATIONS AND UNMET USABILITY FEATURES

- The wave system does not stop after 10 waves. It loops until the player has died.
- As the wave number increases the strength, defense and speed of the enemy does not increase.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- Using If statements and creating a winner's screen that the game transitions too after completing wave 10.
  E.g. If Wave10 = true then Display Winner's Screen
- Creating different enemies to spawn with different strength, defense and speed values. Therefore, the enemies get stronger as the player progress through the game.

CRITERIA POINT

Variety of weapons • 3 weapons to select from before beginning the game.	This will be added to give the player freedom and personalization of his kit for combat. This has also been already successfully implemented in other successful finished solutions like Fortnite Save The World.

The criteria point has been partially met.

EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABLITY

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN PARTIALLY MET

The screenshot shows the only weapon in the game that can be used.

JUSTIFICATION OF USABILITY

Was a success as the weapon allows the player to damage/kill enemies to aid in survival.

LIMITATIONS AND UNMET USABILITY FEATURES

- There are no additional weapons to add variety to the game, the game can get dull and boring fighting with the same weapon.
- There should be 2 different weapon types available for the player to use.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- Create a menu screen before the game starts that allows the player to pick the weapon they want. Depending on the weapon selected a different sprite is spawned. Each weapon has a different damaging variable and cooldown, so it feels like a different weapon. For example, a battle-axe will have more damage, but it will take more time to swing than the sword.

CRITERIA POINT

Soundtrack 2 sound track pieces. • 1st trsack for the menu screen.	Interviewees have claimed that a game without sound can be very dull. Soundtracks will be included to convey emotions and increase immersion of the

• 2nd When the player is playing the game.	game. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World.

Criteria was not met.

LIMTATIONS AND UNMET USABLITY FEATURES

This was unmet because I did not have time to add soundtracks into the game.

- There should be 2 soundtracks implemented in the game to increase the intensity of the game and increase immersion. This is to stop the game feeling dull and boring.

CRITERIA POINT

Environmental Sound • 2 tracks of dialogue. • 2 tracks of swords clashing. • 1 track of arrow firing • 1 track played when the player dies.	This will be added to increase the immersion the player feels. Additionally, this has also been included in successful finished solutions such as Nazi Zombies and Fortnite Save The World.

This criteria point was not met.

LIMTATIONS AND UNMET USABLITY FEATURES

This was unmet because I did not have time to add soundtracks into the game.

Environmental soundtracks could have increase immersion of the game and make it seem less boring and dull. It would make the game more appealing for the players.

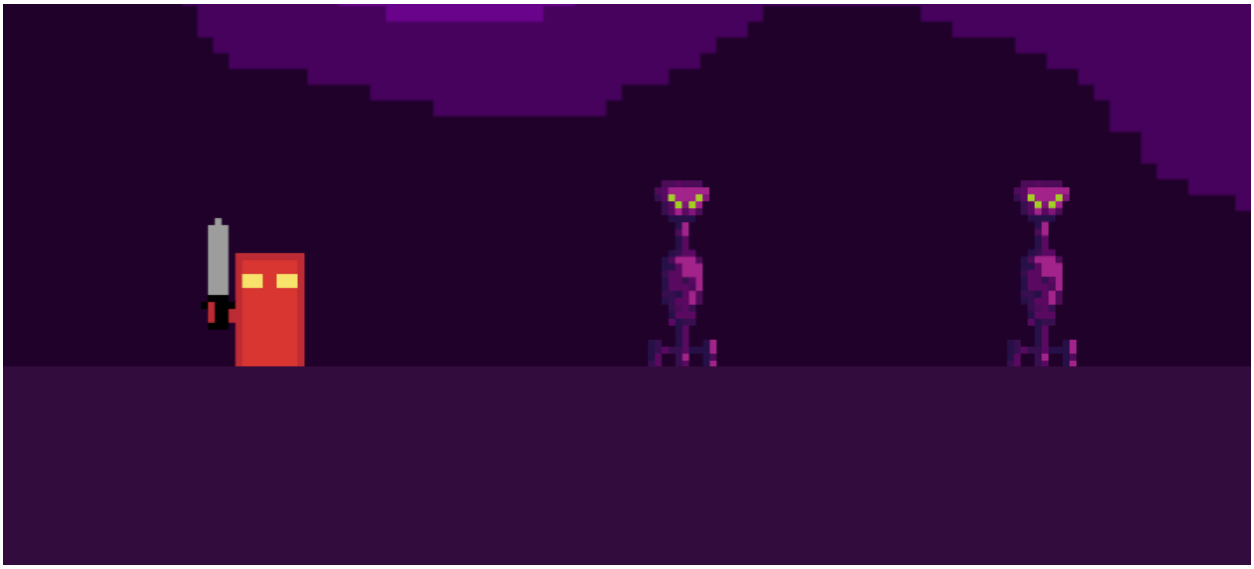
CRITERIA POINT

2 Different type of enemies. • Melee enemies • Ranged enemies	To introduce variation of the enemies the player fights.

This criteria point was partially met.

I did not have the skill required to create an enemy that fired ranged projectiles at the player. However, I did manage to create a melee enemy.

EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABLITY



Test Identification	Test	Input	Expected Outcome	Test Outcome
EDMG1	Does the enemy damage the player	No Input Required	The enemy should attack the player and damage him by the damage variable.	Passed
EDMG2	Does the attack animation launch?	No Input Required	The enemy attack animation should play when the enemy attacks the player.	Untested

EDMG2	Does the enemy damage nearby allies?	No Input Required	The enemy should only be able to damage the player.	Passed

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN PARTIALLY MET

Test series EDMG shows that enemy attack has been tested and is working, additionally the screenshot shows the melee enemies chasing the player to attack the enemy.

LIMTATIONS AND UNMET USABLITY FEATURES

- The melee enemies do not have an attack animation was I did not have enough time to create and implement the feature.
- The ranged enemies have not been included as I lack the skill to create an enemy that fires projectiles at the player.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- Create an animation that launches when the enemy launches its attack. This can be done using if statements and Booleans.

CRITERIA POINT

Leaderboard showing the Top 10 players and scores. Stored outside the game so it can be reloaded when the game is closed and reopened.	Isaac wants the game to show who the best is and split people into different skill levels. It increases the competitive nature of the game and motivates people to try beat the top scores on the leaderboards.

This criteria point was partially met.

EVIDENCE OF PARTIALLY MEETING THE SUCCESS CRITERIA AND EVIDENCE OF USABLITY

Test Identification	Test	Input	Expected Outcome	Test Outcome
LED1	Are the bestscore and the thebestplayer variables saved externally?	No input required.	These variables should be stored externally in a file so that they can be used when the game is reopened.	Passed
LED2	Does the leaderboards display the externally stored variables.	No input required.	The screen should display the externally stored variables. This is done so that we can see the Best Player's name and the score.	Passed
LED3	If the high-score is beaten are the new details saved and the old ones removed?	A and D to locate enemies. J to attack enemies. Must beat the old high-score.	The game should replace the old high-score details and replace it with the current player's details.	Passed
LED4	Does the high-score values reset when the reset button is pressed	Left Click on Button	The high-score variables should be reset.	Passed

HOW THE EVIDENCE SHOWS THAT THE SUCCESS CRITERIA HAS BEEN PARTIALLY MET

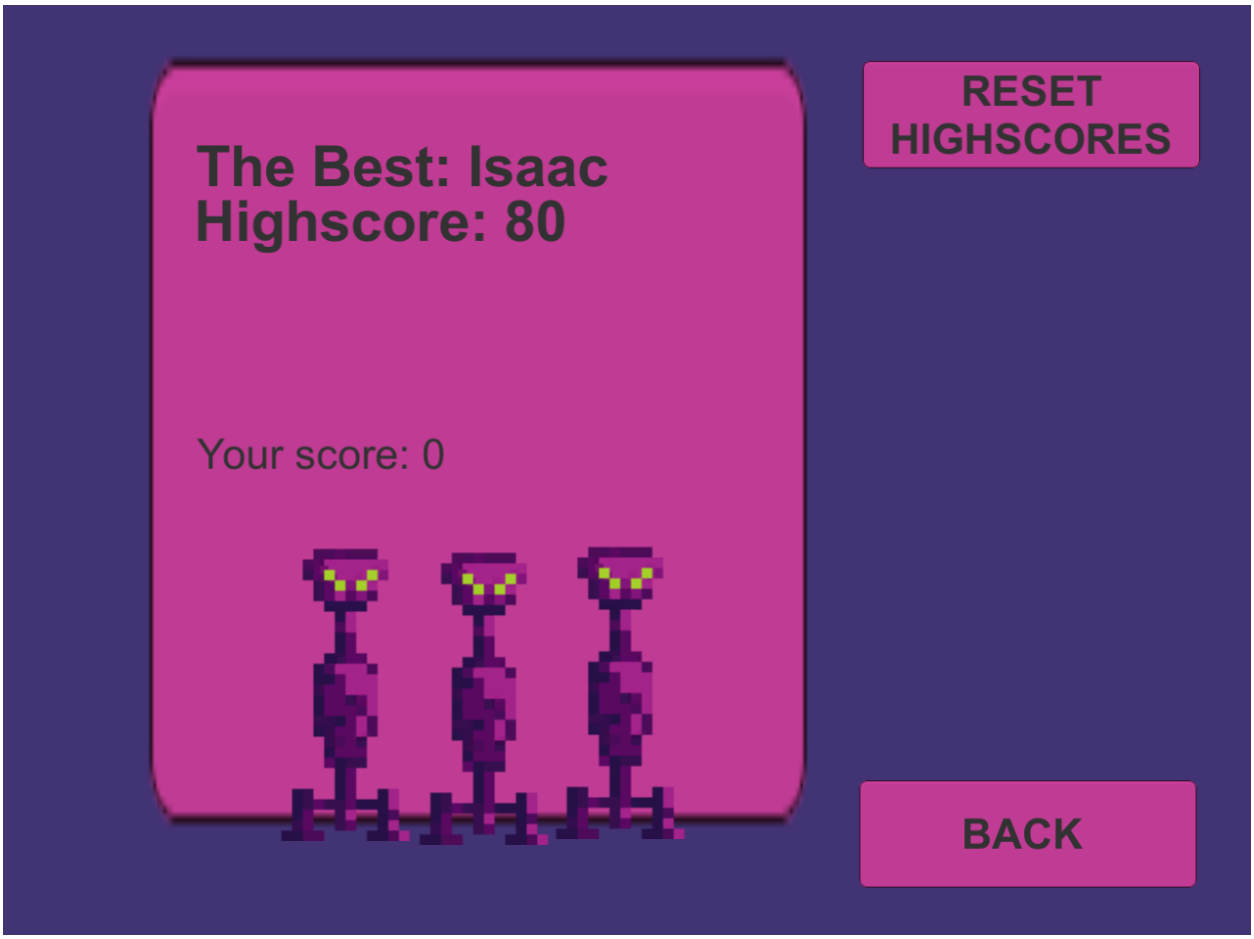
Test series LED shows the leaderboards being tested and shows that they work, additionally the screenshot shows the current high-score holder which is Isaac.

LIMTATIONS AND UNMET USABLITY FEATURES

- I did not have the skill to create a database of the top 10 players of the game, therefore I had to compromise and only store the best score of the game.

HOW THE PROGRAM CAN BE DEVELOPMED TO DEAL WITH THE LIMITATIONS AND CHANGES

- Create a database that is stored externally. When the leaderboards is loaded this file is read and the leaderboards are displayed. Additionally when there is a score that has been beaten the game writes to the file and replaces the beaten score and saves the new table and displays it.



MAINTAINANCE AND ADDITONAL USABILITY FEATURES

These are features that can be added and used to maintain the game in further development.

1. Add soundtracks and environmental sound
2. Add a ranged enemy
3. Add a heavy attack.
4. Add a blocking move.
5. Add melee enemy attack animation.
6. Add a new map.
7. Add extra weapons.
8. Top 10 leaderboard

PROJECT APPENDIXES

CODE LISTING

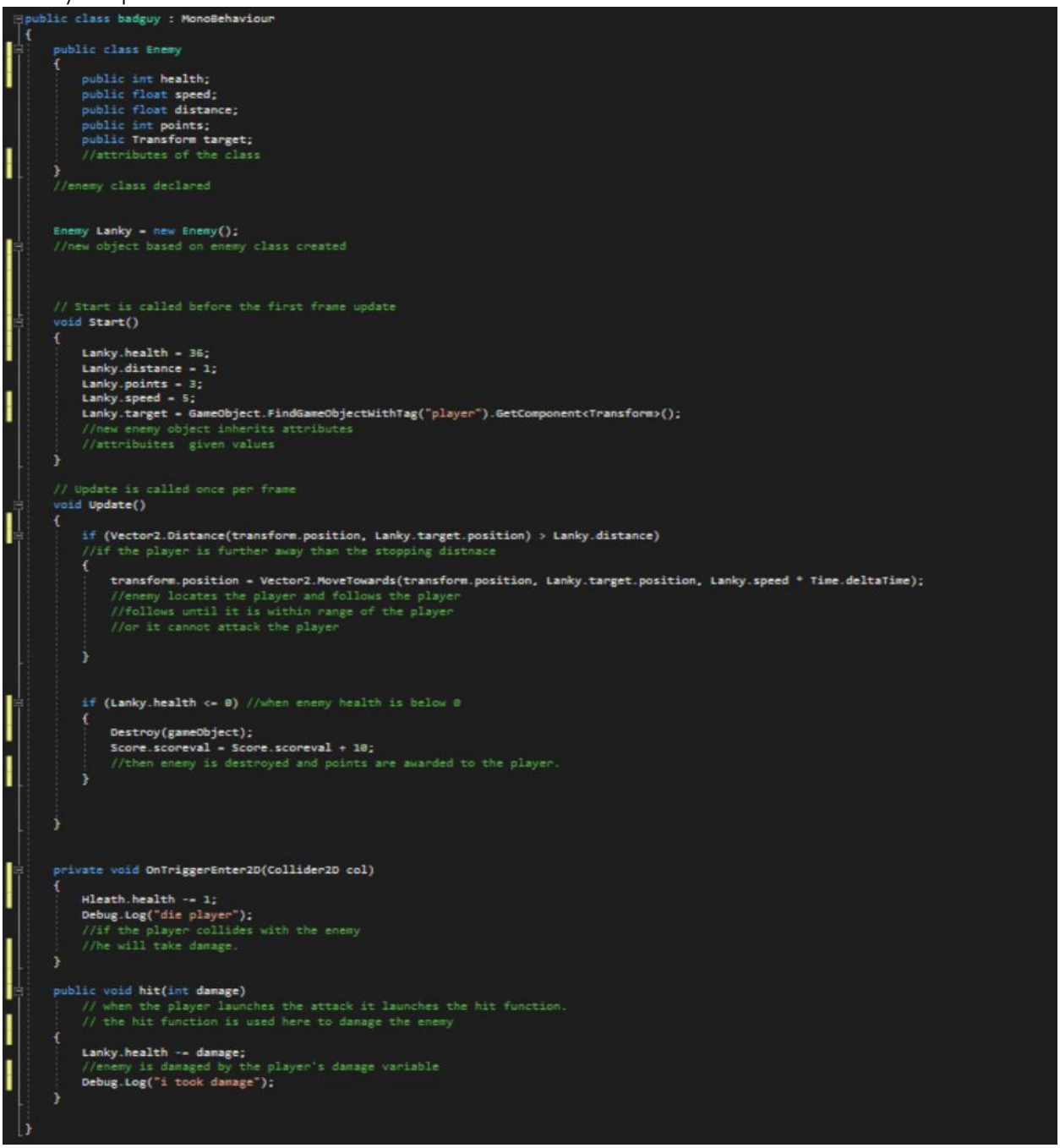
Player Scripts

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

public class playerscript : MonoBehaviour {
    public float jumpforce;
    public Rigidbody2D rb;
    public bool grounded;
    public int jumps;
    //variables for jumping

    public float movespeed;
    public float tapwindow;
    private float taptime;
    //variables for dashing & moving

    public float cooldown;
    public float cooldownlength;
    public Transform attackarea;
    public float attackrange;
    public LayerMask badguy;
    public int damage;
    public Animator ani;
    public bool attacking;
    //variables for attacking



    void Start()
    {
        grounded = false;
        rb = GetComponent<Rigidbody2D>();
        jumps = 2; ;
        //variables for jumping given values.

        movespeed = 7f;
        tapwindow = 0.5f;
        taptime = 0;
        //variables for dashing & moving given values

        ani = GetComponent<Animator>();
        //variables for attacking given values

       //healthbar = GetComponent<Image>();
        //variables for healthbar script.

    }


    void Update()
    {

        //Jumping
        if (Input.GetKeyDown(KeyCode.W) && jumps > 0) //if jumps is bigger than 0 and W has been pressed launch jump.
        {
            rb.velocity = Vector2.up * jumpforce;
            jumps--;  //number of jumps reduced.
        }
```

```
//Left Right
if (Input.GetKey(KeyCode.D)) //if D is held down or pressed
{
    transform.localScale = new Vector2(1f, 1f);
    //will face the player to the right
    rb.velocity = new Vector2(movespeed, rb.velocity.y);
    //character will move to the left at a fixed velocity and same y position
}
if (Input.GetKeyDown(KeyCode.D)) //if D is pressed again
{
    if ((Time.time - taptime) < tapwindow) //if D is pressed again between the time window of 0.5 seconds
    {
        rb.position = new Vector2(rb.position.x + 3, rb.position.y);
        //character will dash to the right but remain in the same y position.
    }
    taptime = Time.time;
    //time window reset (dash reset)
}


if (Input.GetKey(KeyCode.A)) //if A is held down or pressed
{
    transform.localScale = new Vector2(-1f, 1f);
    //will face the player to the left

    rb.velocity = new Vector2(-movespeed, rb.velocity.y);
    //character will move to the left at a fixed velocity and same y position
}
if (Input.GetKeyDown(KeyCode.A)) //if A is pressed again
{
    if ((Time.time - taptime) < tapwindow) //If a is pressed again between the time window of 0.5 seconds
    {
        rb.position = new Vector2(rb.position.x - 3, rb.position.y); //character will dash to the left.
    }
    taptime = Time.time; //time window reset (dash reset)
}


//Flip Upright
if (Input.GetKeyDown(KeyCode.H))
{
    transform.rotation = Quaternion.Euler(0, 0, 0);
    //when H is pressed it will flip the character upright.
}


//Attacking

if (cooldown <= 0) //when cooldown reaches 0 or less than 0
{
    if (Input.GetKey(KeyCode.J)) //if the player presses J
    {
        ani.Play("attacking");
        //play attack animation
        Debug.Log("animation");
        Collider2D[] KillEnemies = Physics2D.OverlapCircleAll(attackarea.position, attackrange, badguy);
        //creates the attack radius where enemies are damaged
```

```
                for (int i = 0; i < KillEnemies.Length; i++)
                {
                    KillEnemies[i].GetComponent<badguy>().hit(damage);
                    //hit function is launched, enemies are damaged by the "damage variable"
                    Debug.Log("HITTING ENEMIES");
                }

            }
            cooldown = cooldownlength;
            //cooldown is reset
        }
        else

        {
            cooldown -= Time.deltaTime;
            //cooldown time is reduced
        }
    }







    //other

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawSphere(attackarea.position, attackrange);
        //sets up invisible attack radius where enemies are damaged
    }


    void OnTriggerEnter2D() // when the player and the ground touch
    {
        grounded = true;
        jumps = 2;
        //number of jumps reset after touching the ground.
    }


    void OnTriggerExit2D() // when the player and the ground are not touching
    {
        grounded = false;
    }
}
```

Enemy Script



```
public class badguy : MonoBehaviour
{
    public class Enemy
    {
        public int health;
        public float speed;
        public float distance;
        public int points;
        public Transform target;
        //attributes of the class
    }
    //enemy class declared


    Enemy Lanky = new Enemy();
    //new object based on enemy class created



    // Start is called before the first frame update
    void Start()
    {
        Lanky.health = 36;
        Lanky.distance = 1;
        Lanky.points = 3;
        Lanky.speed = 5;
        Lanky.target = GameObject.FindGameObjectWithTag("player").GetComponent<Transform>();
        //new enemy object inherits attributes
        //attribuites  given values
    }

    // Update is called once per frame
    void Update()
    {
        if (Vector2.Distance(transform.position, Lanky.target.position) > Lanky.distance)
        //if the player is further away than the stopping distnace
        {
            transform.position = Vector2.MoveTowards(transform.position, Lanky.target.position, Lanky.speed * Time.deltaTime);
            //enemy locates the player and follows the player
            //follows until it is within range of the player
            //or it cannot attack the player

        }


        if (Lanky.health <= 0) //when enemy health is below 0
        {
            Destroy(gameObject);
            Score.scoreval = Score.scoreval + 10;
            //then enemy is destroyed and points are awarded to the player.
        }


    }


    private void OnTriggerEnter2D(Collider2D col)
    {
        Hleath.health -= 1;
        Debug.Log("die player");
        //if the player collides with the enemy
        //he will take damage.
    }

    public void hit(int damage)
        // when the player launches the attack it launches the hit function.
        // the hit function is used here to damage the enemy
    {
        Lanky.health -= damage;
        //enemy is damaged by the player's damage variable
        Debug.Log("i took damage");
    }
}
```

Leaderboard Script

```
using UnityEngine;
using UnityEngine.UI;

public class dice : MonoBehaviour
{

    public Text score;
    public Text highscore;
    public Text best;
    public Text playername;
    //variables declared
    //all variables are UI text


    void Start()
    {
        highscore.text = "Highscore: " + PlayerPrefs.GetInt("Highscore", 0).ToString();
        //externally stored highscore is loaded and is displayed on the UI of the leaderboard screen.
        best.text = "The Best: " + PlayerPrefs.GetString("thebest");
        //externaly stored username is loaded and is displayed on the UI of the leaderboard screen.

    }


    //updates every frame
    void Update()
    {
        int hs = Score.scoreval;
        //integer that stores the current score
        score.text = "Your score: " + hs.ToString();
        //displays your current score
        playername.text = usernamemanager.username;
        //displays your current username

        if (hs > PlayerPrefs.GetInt("Highscore",0))
        //if the current score is higher than the highscore
        {
            PlayerPrefs.SetInt("Highscore", hs);
            highscore.text = "Highscore: " + hs.ToString();
            //externally stored score is replaced
            //new highscore is displyaed

            PlayerPrefs.SetString("thebest", usernamemanager.username);
            best.text = usernamemanager.username;
            //externally stored username is replaced
            //new username is displayed
        }

    }

}
```

Health System

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class Hleath : MonoBehaviour
{
    // Start is called before the first frame update

    public static int health;
    //global variable health
    public int HP;
    public int numofhearts;
    public Image[] hearts;
    //array of images
    public Sprite fullheart;
    public Sprite emptyheart;
    //variables declared


    void Start()
    {
        health = 10;
        Debug.Log("health = 10");
        //health set to 10 as the game starts
    }

    // Update is called once per frame
    void Update()
    {
        if (health <= 0) // if health = 0
        {
            Destroy(gameObject);
            //player will die
        }


        for (int i = 0; i < hearts.Length; i++)//for loop
        {

            if (i < health) //if the counter is less than health
            {
                hearts[i].sprite = fullheart;
                //it will display a full heart. As player stll has HP remaining.
            }
            else //if the counter is more than health
                //player has lost health
            {
                hearts[i].sprite = emptyheart;
                //it will display a empty heart. As player has lost HP.
            }

            if (i < numofhearts)
            {
                hearts[i].enabled = true;
                //if the counter is less than number of hearts then it will display a heart
                //the code has not reached the maxiumum HP.
                //it will display the hearts as it checks if it has reached max HP
            }
            else
            {
                hearts[i].enabled = false;
                //if the counter is larger than the number of hearts it shows that the counter has reached max hp.
                //Therefore the rest of the hearts are disbaled and not displayed.
            }
        }
    }
}
```

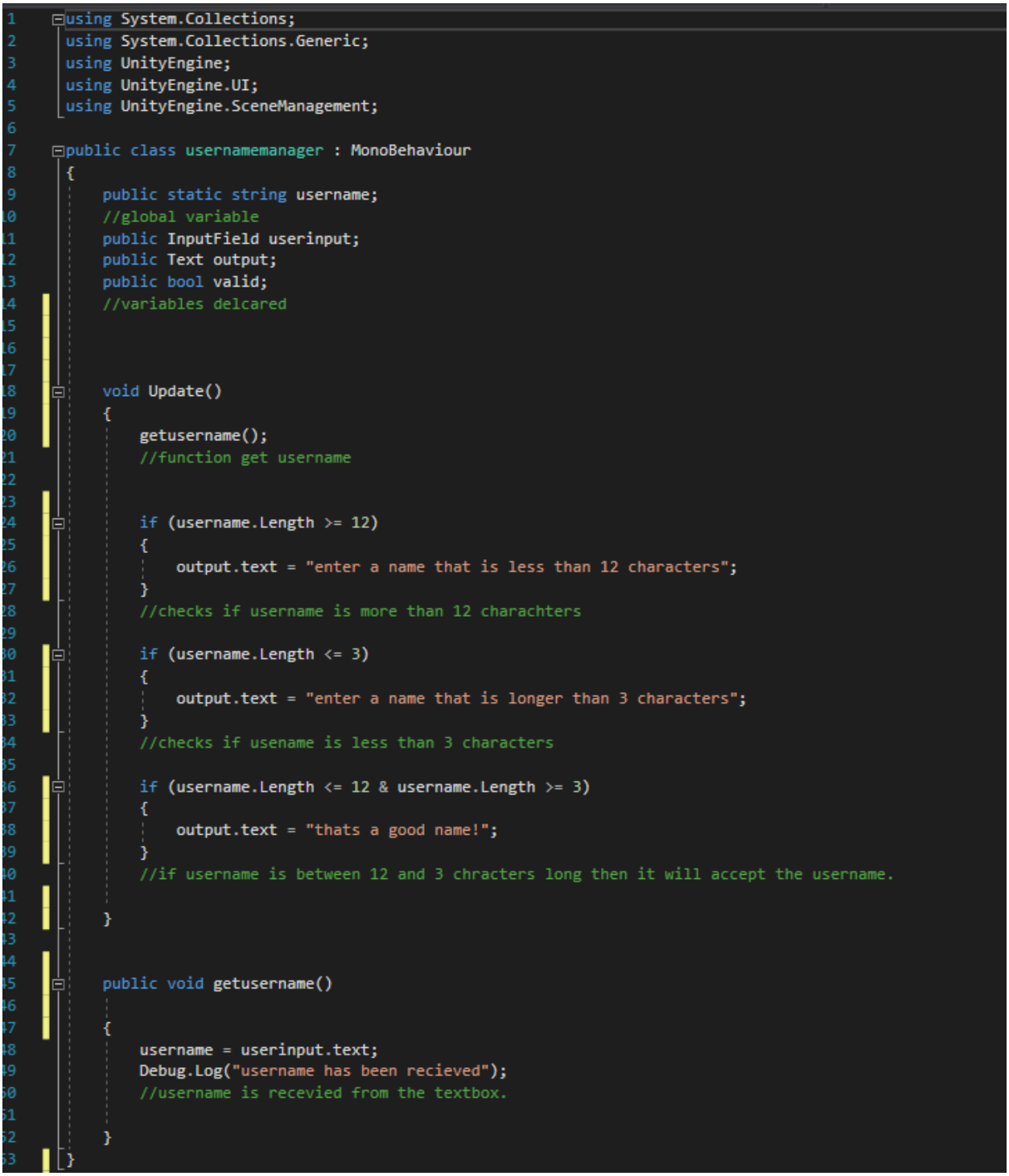
Wave System

```
[System.Serializable]
//allows coroutines in Unity
public class Wave
{
    public string name;
    public Transform enemy;
    public int count;
    public float rate;
    //object wave created
}
public Wave[] waves;
//array of wave objects created
private int nextWave = 0;
public int NextWave
{
    get { return nextWave + 1; }
}
//this variable value nextwave + 1
public Transform[] spawnPoints;
//variables for spawn points created
public float timeBetweenWaves = 5f;
//the time the player has to rest befor the next wave starts
private float waveCountdown;
//the countdown the wave gives
public float WaveCountdown
{
    get { return waveCountdown; }
}
private float searchCountdown = 1f;
//searchcountdown variable declared
private SpawnState state = SpawnState.COUNTING;
public SpawnState State
{
    get { return state; }
}

void Start()
{
    if (spawnPoints.Length == 0)
    {
        Debug.LogError("there are no spawnpoints");
    }
    //checks if there are spawn points to spawn enemies.

    waveCountdown = timeBetweenWaves;
    //wave countdown is declared
}

void Update()
{
    if (state == SpawnState.WAITING) //if the state is waiting (waiting for the player to finish the wave)
    {
        if (!EnemyIsAlive()) //if there are no more enemies on the map then
        {
            WaveCompleted();
            //run wavecompleted function
        }
        else
        {
            return;
            //do nothing
        }
    }
```

```
    if (waveCountdown <= 0)
        //if wavecountdown has finished
    {
        if (state != SpawnState.SPAWNING)
            //if the state is not equal to spawnning
        {
            StartCoroutine( SpawnWave ( waves[nextWave] ) );
            //run coroutine with the next wave in the array.
        }
    }
    else //if the coutndown hasnt finished
    {
        waveCountdown -= Time.deltaTime;
        //keep counting down
    }
}

void WaveCompleted()
    //function that handles the game when the wave is completed
{
    Debug.Log("Wave Completed!");

    state = SpawnState.COUNTING;
    waveCountdown = timeBetweenWaves;
    //starts giving the player time to rest.

    if (nextWave + 1 > waves.Length - 1)
        //if next wave is bigger than the number of waves in the array
    {
        nextWave = 0;
        Debug.Log("looping game");
        //there are no more waves remaining
        //the game will loop starting from wave 0
    }
    else //if not
    {
        nextWave++;
        // it goes onto the next wave in the array
    }
}

bool EnemyIsAlive()
    //function that checks if there are any enemies alive from the previous wave
{
    searchCountdown -= Time.deltaTime;
    //countdown decrements
    if (searchCountdown <= 0f)
        //if the countdown reaches 0
    {
        searchCountdown = 1f;
        //countdown is increased, gives more time to search for the enemy.
        if (GameObject.FindGameObjectWithTag("badguy") == null)
        //if the game finds an enemy
        {
            return false;
            // it will return false
        }
    }
    return true;
    //otherwise it will return true
}
IEnumerator SpawnWave(Wave _wave)
{
    Debug.Log("Spawning Wave: " + _wave.name);
    //broadcasts what wave the player is on
    state = SpawnState.SPAWNING;
    //state is changed to spawnning
```

```
        for (int i = 0; i < _wave.count; i++)
        //loop that spawns the enemy in the wave object
        {
            SpawnEnemy(_wave.enemy);
            //runs another routine with the enemy with the wave object
            yield return new WaitForSeconds( 1f/_wave.rate );
            //uses the rate from the wave object to increase or decrease the rate of spawnning
        }

        state = SpawnState.WAITING;

        yield break;
        //after spawnning it changes the game state to waiting
    }

    void SpawnEnemy(Transform _enemy)
        //routine that spawns enemies
    {
        Debug.Log("Spawning Enemy: " + _enemy.name);

        Transform _sp = spawnPoints[ Random.Range (0, spawnPoints.Length) ];
        Instantiate(_enemy, _sp.position, _sp.rotation);
        //at the spawn point it spawns new enemies every time it called.
        //the point at where it spawns is random
    }
```

Username Validation



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

public class usernamemanager : MonoBehaviour
{
    public static string username;
    //global variable
    public InputField userinput;
    public Text output;
    public bool valid;
    //variables delcared



    void Update()
    {
        getusername();
        //function get username


        if (username.Length >= 12)
        {
            output.text = "enter a name that is less than 12 characters";
        }
        //checks if username is more than 12 charachters

        if (username.Length <= 3)
        {
            output.text = "enter a name that is longer than 3 characters";
        }
        //checks if usename is less than 3 characters

        if (username.Length <= 12 & username.Length >= 3)
        {
            output.text = "thats a good name!";
        }
        //if username is between 12 and 3 chracters long then it will accept the username.


    }


    public void getusername()

    {
        username = userinput.text;
        Debug.Log("username has been recieved");
        //username is recevied from the textbox.

    }
}
```

Leaderboard Script

```
using UnityEngine;
using UnityEngine.UI;

public class dice : MonoBehaviour
{

    public Text score;
    public Text highscore;
    public Text best;
    public Text playername;
    //variables declared
    //all variables are UI text


    void Start()
    {
        highscore.text = "Highscore: " + PlayerPrefs.GetInt("Highscore", 0).ToString();
        //externally stored highscore is loaded and is displayed on the UI of the leaderboard screen.
        best.text = "The Best: " + PlayerPrefs.GetString("thebest");
        //externaly stored username is loaded and is displayed on the UI of the leaderboard screen.

    }


    //updates every frame
    void Update()
    {
        int hs = Score.scoreval;
        //integer that stores the current score
        score.text = "Your score: " + hs.ToString();
        //displays your current score
        playername.text = usernamemanager.username;
        //displays your current username

        if (hs > PlayerPrefs.GetInt("Highscore",0))
        //if the current score is higher than the highscore
        {
            PlayerPrefs.SetInt("Highscore", hs);
            highscore.text = "Highscore: " + hs.ToString();
            //externally stored score is replaced
            //new highscore is displyaed

            PlayerPrefs.SetString("thebest", usernamemanager.username);
            best.text = usernamemanager.username;
            //externally stored username is replaced
            //new username is displayed
        }

    }
}
```

Menu Manager

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

public class Menumanager : MonoBehaviour
{
    public void LaunchHTP()
    {
        Debug.Log("going to the howtoplay>");
        SceneManager.LoadScene("htp");
        Debug.Log("went to how to play");
    }
//launches the How To Play menu screen

    public void LaunchGame()
    {
        Score.scoreval = 0;
        SceneManager.LoadScene("Game");

    }
//launches the game


    public void LaunchLeaderboards()
    {
        Debug.Log("going to lb");
        SceneManager.LoadScene("lb");

    }
//launches the Leaderboards



    public void LaunchMenu()
    {
        Debug.Log("going to the menu");
        SceneManager.LoadScene("Menu");

    }
//launches the Main Menu screen

    public void resetscores()
    {
        PlayerPrefs.SetInt("Highscore", 0);
        PlayerPrefs.SetString("thebest", null);
    }
//resets highscore values



}
```

Score System

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

public class Score : MonoBehaviour
{

    public static int scoreval = 0;
    //global variable declared;
    Text score;
    //variable that holds UI text

    // Start is called before the first frame update
    void Start()
    {
        score = GetComponent<Text>();
        //variable assigned to the text on UI.
    }

    // Update is called once per frame
    void Update()
    {
        score.text = "Score: " + scoreval;
        //the text changes to display the cuerrent score on the UI.
    }
}
//public static int scoreval = 0;
```

REFERENCES:

Used for general information about the unity system and program.
https://stackoverflow.com/
https://docs.unity3d.com/ScriptReference/
https://docs.unity3d.com/Manual/index.html
https://forum.unity.com/