Josh Shepherd 1700471
Savannah Coward 1801678

“Quinn’s Escape”

Games Design Document

v1.0
01/02/2021

Team J&S

Josh Shepherd - 1700471
Savannah Coward - 1801678

Josh Shepherd 1700471
Savannah Coward 1801678

Contents

Josh Shepherd 1700471
Savannah Coward 1801678

Section I - Game Overview

(By Savannah Coward)

Introduction

The game is staged as a platform scroller, focused on a main character called “Quinn”. With heavy inspiration from scrolling games such as Mario, Quinn’s Escape follows Quinn as he tries to escape a world plagued with monsters and fierce enemies. Quinn will face a varying degree of enemy types to prepare him for the final boss, who stands between him and true freedom.

Genre

The genre is platformer, similar to Mario. It is a side scrolling version rather than a three dimensional. The game will feature climbing, jumping and some puzzles to allow the player to gather some enjoyment.

Target Audience

The target audiences is for all ages, primarily. We want to appeal to any audience that is out there and enjoys platformers. We do not want to be restricted by children, like Nintendo, but we do not want to be only aimed for older generations, either.

Game Flow

The flow of the game is simple, pushing the player to move along the level. The camera may move to urge the player to be faster, but that will have to be considered alongside how much hinderance the enemies become. Enemies are there to block the flow, albeit briefly, similar to the goombas in Mario.

Look and Feel/Theme

The theme will be very basic, to allow the player to use their imagination and insert themselves into the world. The main character will be a basic dummy – a mannequin (Quinn) – so the player is not restricted by gender choices or character designs. The game isn’t RPG, so the design of the main character doesn’t matter as much as the surrounding worlds. As for the world itself, the world will be bright and colourful to contrast the character. The enemies will begin and be eye catching, so the player is immediately brought to them. The theme will be mainly bright, but not childish. Something eye-catching for all the audiences we are aiming for.

Project Scope

The scope is twelve weeks – possibly thirteen if we need an extension because we aren’t quite polished. We aim to have one level, or a slice if the scope is too far. We aim to have an entire level that goes through enemy ideas and the boss at the end, but we may add a “bonus” level to showcase ideas we have for the full scoped game. For NPC’s, we aim to have four.

- Firstly, a chasing NPC. This will teach the player that some enemies will chase you, and when they touch you then you will be harmed.
- Secondly, a shooting NPC. This NPC will not move, but instead teach the player that the NPC does not have to be on screen to harm you.
- Thirdly, a punching NPC. This is similar to the chasing, however, will have explicit animation for the player to predict. This will teach the player that there is often patterns to the enemies.

- Lastly, the final NPC will be the boss, which will be a combination of all the enemies that they have encountered. This is so the player has a puzzle, so to speak, to work out how to battle based on the NPCs they have encountered previously.

Due to the fact that it is only one level, and the first level, we wanted to ensure that the player would not be overwhelmed and would be eased into the game. The NPCs are simple to begin with, so that players are not overwhelmed by the difficulty. Starting with the first level should give clients an idea of the concepts we have and allow for full choices to be made.

Within the first week seven weeks, we aim to have the entire design sorted, so we can pour the efforts following this into coding and polishing up the game ready for the client to see. There will be separate projects made by us to highlight the ideas that the client may see towards the end, so that they can see a prototype.

After such, week 8 should be focused on attempting to implement the separate games that we have been working on; now, this isnt in the official first seven week plan, as it's mostly going to be efforts to test design ideas that we have, rather than start the official game. After we implement our codes together, we can begin working on the game entirely.

Weeks 9 to 11 will be focused on polishing and ensuring that the game looks as we intended. This involves coding it, designing the levels within the engine and ideas that we have.

Section II – Gameplay & Mechanics

(By Josh Shepherd)

Game State

The game state will contain all of the values that will change at runtime. Since our game is a side scroller, these games still contain a lot of values that change at runtime, such as:

- Player('s):
  - Position, health, lives, mouse position (X & Y), direction input
- Enemy('s):
  - Position, health, target, projectile cooldown, AI state
- Projectiles shot, their position, amount of damage to deal
- Current score, current play time, camera position.
- Created power-ups, their position, type of power-up

Game Space

The complete space of our game will be a main menu scene and a main level scene that will be in 2.5D (created in 3D but locked to 2D movement). The main level scene will start and progress right to the end of the level. All area in the level will be accessible to the player, including hidden puzzle rooms that can be accessed through secret entrances. The level will a base floor that will contain holes that the player can fall down, and also contain many static and moveable platforms that can be jumped on to. The player can use these platforms and the terrain to take cover from enemy projectiles.

Game Rules

In the game, there will be many rules that the player, enemies, and world will adhere to. Below is a list of the most important game rules:

- If the player collides with an enemy or an enemy projectile, the player loses health.
- If the player loses all of their health, the player loses a life and starts at a checkpoint.
- Player falls out of the level; the player loses a life.
- Once the player loses all of their lives, it is game over.
- Once the player defeats the boss enemy, the game is won.
- Pressing the movement/jump keys will move the player left and right or make them jump.

If a player adheres to the rule and is able to complete the goals, then the player will have an enjoyable experience and find the game fun.

Mechanics

Short list of available mechanics in the game:

- If the player jumps below a power-up box, the box creates a power-up above the box which can be picked up by the player.
- The player can shoot a fireball projectile toward the user's mouse position once the fire button is pressed.
- If the player defeats the final boss, the level is complete, and the player wins.
- The player can enter a hidden room to solve its puzzle and earn a reward.

Section III – Player

(By Josh Shepherd)

Objective

The main objective for the player is to reach to the end of the level without dying and to defeat the main boss and escape. Computer controlled enemies will be in the player's way to stop and prevent them from escaping. The player can have mini objectives along the way to find and discover secret puzzle rooms to gain more score and gain extra power-ups to help them to the end.

Avatar

Overview & Backstory

The player will take the form Quinn, named after his appearance. Quinn is a mannequin who ended up with a life as a display model in a clothing store. Many years after the store closed down, Quinn had finally had enough of being trapped, alone inside the abandoned store and attempts to make his escape. However, since the store has closed, there are many new friends that have moved in and occupied the store, some of those who are friends, some who are enemies.

The mannequin Unreal Engine 4 model to use for Quinn

Technical

The mannequin model is the default model for Unreal Engine 4, and we will utilize it and lean into its usage to maximise the final game. Since a lot of Unreal Engine animations exist and are shared against the default mannequin model, it means we will have a wide range of animations at our disposal to utilize and use. Since the player can walk, jump, fire, and slam down, we will need animations related to these actions to be used. We can search the Unreal Marketplace for free animations that can be utilized.

Actions

Player can perform an array of actions that can assist them in playing the game. The player can perform offensive actions such as firing and defensive and movement actions such as jumping and walking to avoid damage.

Movement

As movement is a key part of the player's input, the player can walk left and right throughout the level on a prebuilt environment of land and platforms. The player is also able to jump to traverse to different platforms and to cross big gaps in the environment. As well as jumping up, the player can also slam down to the ground quickly which can also be used as an offensive action. While the player is slamming down, no other movement input can be used until the player reaches the ground.

Input

The movement input from the player should feel snappy and respond to each key press immediately, instead of adding velocity which would cause the character to slow down and slide once a button has been released. If the player is in the air, they should be able to move left and right with the correct speed to be able to dodge any projectiles and accurately position themselves to where they wish to land.

Combat

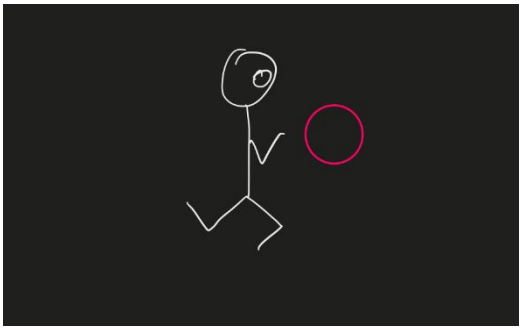
The player will come in contact with enemies throughout the level and will have to defend themselves. The main weapon of the player will be their ability to fire fireballs at the enemies. Once the action button is pressed to fire, a fireball will fire towards the player's mouse at medium speed. If the fireball collides with an enemy, wall, or platform, then the fireball will stop and be destroyed.

The fireball's damage will be a small and have a low cooldown which allows the player to fire often which will give the player confidence in dealing damage.

As the player can jump, they can also land on top of certain enemies to deal damage to them. If the player lands on top of an enemy that does not take damage from being jumped on, then the player will receive a small amount of damage from that enemy.

Fireball Action Technical

The fireball will work by being spawning an actor in front of the player. The fireball actor will be able to collide with any enemy inheriting from a EnemyBase class and will reduce the health of the enemy for the amount of the damage. As the fireball will be the main fire action, the fireball should only shoot when the player releases a button, instead of being able to hold the button down to repeatedly fire. There should be a small cooldown amount in between each fireball, which can be displayed to the user in the UI. If the fireball is on cooldown when the button is pressed, then a fireball should not be spawned. The button to fire a fireball could either be left mouse button or spacebar.



Basic spawn location of a fireball projectile relative to the player

Scoring

As the player travels through the level, the player can gain score by landing attacks on enemies, discovering gold coins and more. The score is shown in the user interface to the player which allows them to see how good or bad they are performing. The player is never able to lose any points through losing a life of getting hit. Below is a list of the available methods a player can earn points:

- A fireball hits an enemy - +0.75
- A small/medium/boss enemy dies - +5/+7/+20
- Player picks up a gold coin - +15
- Reaches a standard checkpoint (ahead of time target bonus) - +10 (+10)
- Level complete (with time bonus) - +25 (* time bonus multiplier)

The player can also earn a time bonus for completing the level. The time bonus will be calculated by having an average/target time to complete the level which gets divided by the player's time to complete the level.

$$\frac{average\ time}{player\ time} = bonus\ multiplier$$

This formula rewards players who can beat the average time for the level but also does not punish players who are unable to reach it and go over. The resulting multiplier will never reach zero

but will greatly reduce the bonus multiplier so the player will always be rewarded with some points. If the resulting bonus multiplier is below 0.25, then that should be used as the minimum bonus.

Camera

The camera will be positioned alongside the player like most 2D side scroller games. The camera will have a slight lag behind the player for a smooth effect and will also be locked to height axis when the player is within the floor's dead zone, similar to other 2D games. If the player jumps and begins to rise to the top of the camera, the camera will pan up and follow the player, but once the player reaches the floor again, the camera will again be locked on the height axis to 0.

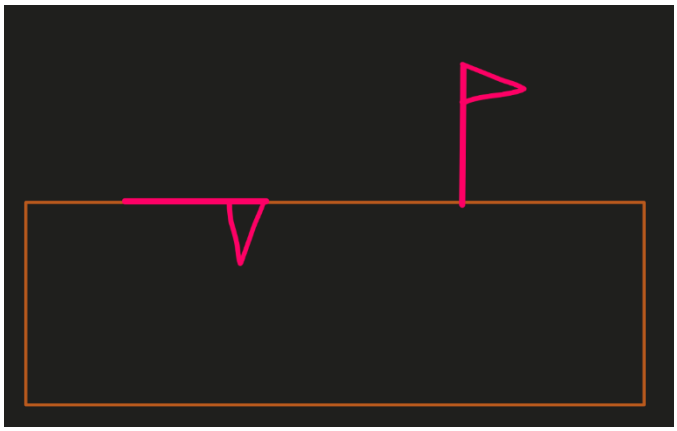
As the puzzle rooms are located below the ground, if the player enters a room, the camera will be set to an appropriate height value and be locked to it until the player exists the room.

Health Points & Lives

The player will have health like a standard game. Their health will start at maximum and can be decreased by being hit by any enemies within the level. The player will also start with 3 hearts (lives) that will be decreased when the player fully loses their health or by falling outside of the level. As the player loses all of their health, they will lose one heart and will continue the game from a checkpoint if one was reached. Once the player loses all of their lives, the game is reset and begins from the start with the player having 3 hearts and full health. The player is able to increase their health and gain extra hearts, if they are not full, through a pickup.

Checkpoints

Once the player loses all their health and loses a life, they will be restored to the latest checkpoint they have passed through. If the player has not passed through any checkpoints yet, they will be restored to the start position of the level. Checkpoints can be triggered by walking past the flag to represent a checkpoint, and can only be triggered once, overlapping any previous checkpoints. If the player goes back through the level and past previous checkpoints, they will still restore to the far-right checkpoint to ensure the player progresses in the right direction and towards the goal of escape.



Drawing of two checkpoints
(Left has not been passed through by player, right has been passed)

Power-Ups

The player can pick up and use power-ups which can be gained from jumping and hitting a special unique box from below. The power-up will be created above the box and will remain static until picked up or expires after 10 seconds. Power-ups will also be randomly given to the player during the final boss battle. There will be four power-ups in game:

Coin Bonus Power-Up

The coin bonus power-up will reward the player with an instant coin bonus value, which will be randomly determined between two values, a minimum and maximum. The coin power-up will also give the player a multiplier for 30 seconds that is randomly selected using `FMath::RandRange` between two values.

Invulnerability Power-Up

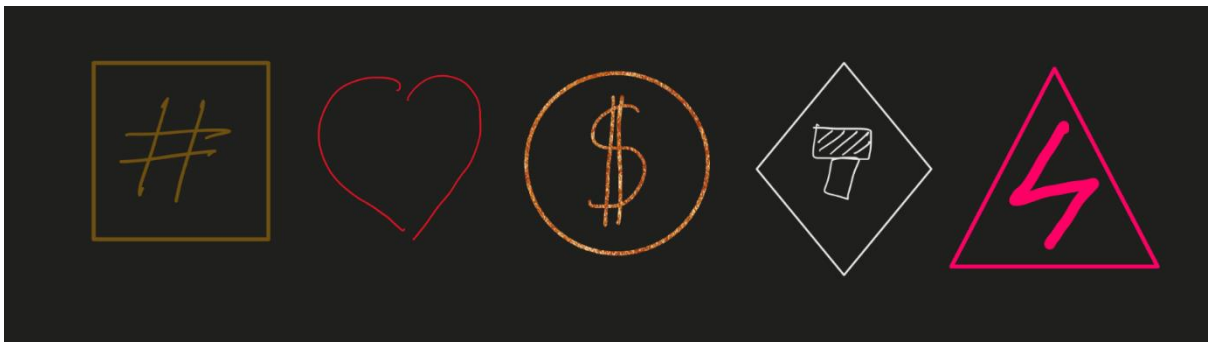
The invulnerability power-up allows the player to be free of damage for 15 seconds, excluding falling out of the map. The player also gains a small movement bonus to assist them to run through the level and avoid damage.

Bonus Damage Power-Up

The bonus damage power-up gives the player's projectiles bonus damage for 20 seconds, allowing them to deal more damage to enemies. The player's damage will be multiplied by a value that is randomly selected using `FMath::RandRange` between two values.

Extra Life Power-Up

The extra life power-up will reward the player with an extra life, giving them an extra chance to progress and earn more score. If the player already has all lives, then they will not be able to pick up this power-up but will be rewarded with additional score, such as +100 points.



Visual representation of the breakable box & the pick-ups
(Breakable box, extra life, coin bonus, bonus damage & invulnerability pick-ups)

Section IV - Artificial Intelligence & Enemy NPCs

(By Savannah Coward)

AI Overview

Enemies

There will be four types of enemies in the game. For implementation, we aim to mostly use finite-state machines to base each non playable character's (NPC) actions. Being able to use a finite-state machine will assist in keeping the code clean and easy to append any future actions, if required.

Figure: FSM of enemy 1

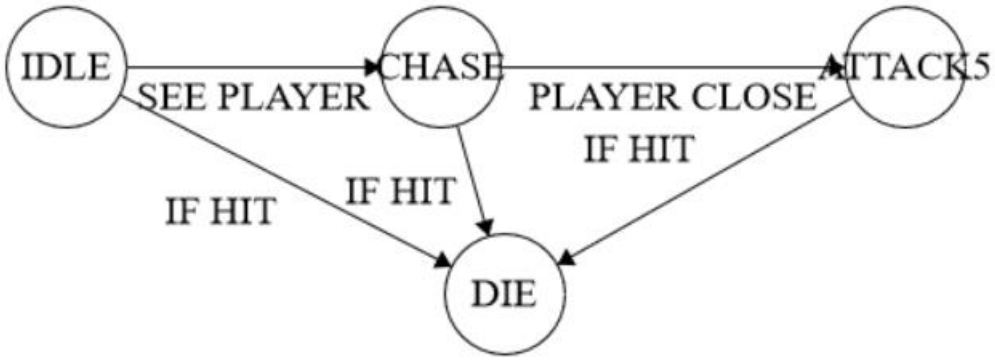


Figure 2: FSM of enemy 2

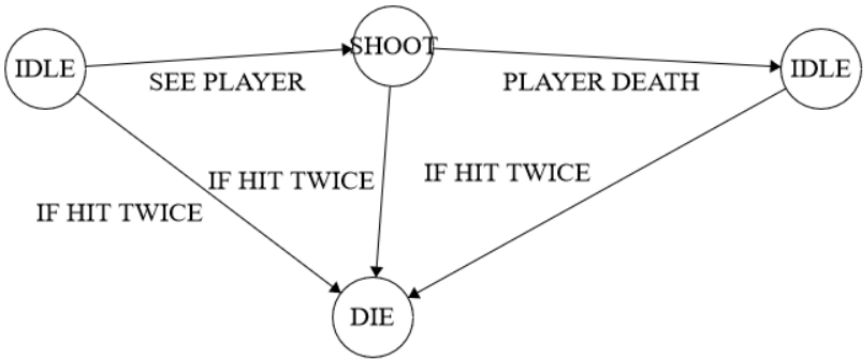


Figure: FSM of enemy 3

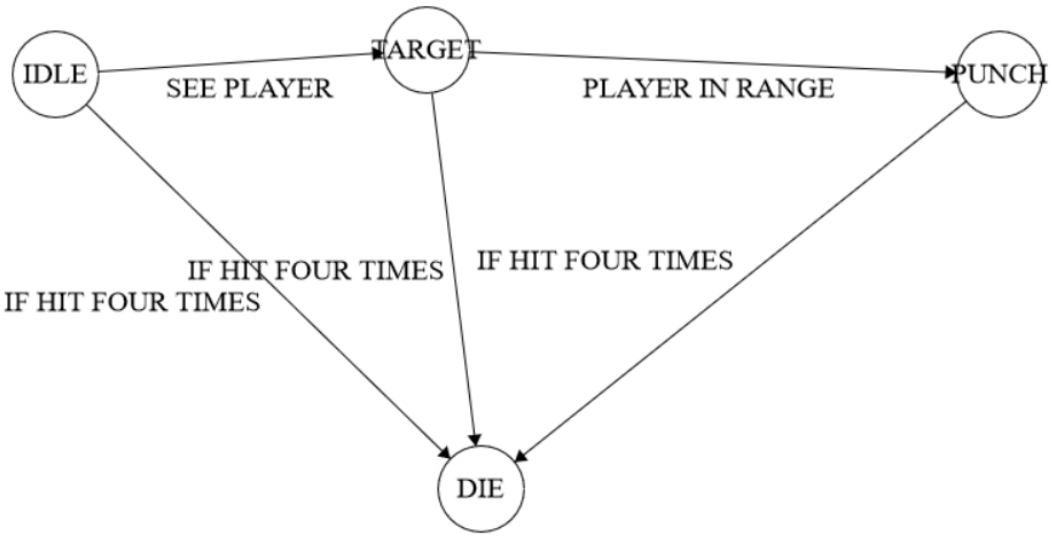
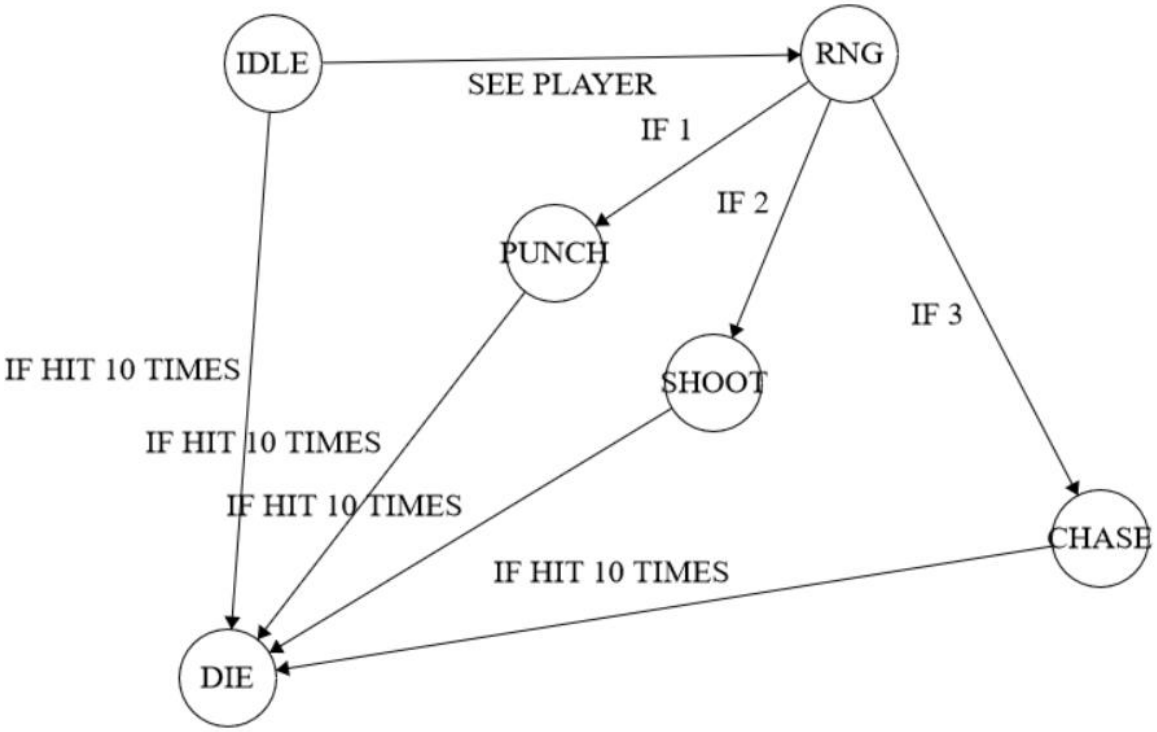


Figure: FSM of Boss enemy



Chasing NPC

Firstly, the chasing NPC. The NPC will target the player, and once they have done so, they will chase the player. Once the player is targeted and chased, if the NPC touches the player, then the player will be harmed. The hurt will not be anything impactful. The NPC can be defeated by either jumping on or punching them.

Shooting NPC

Secondly, the shooting NPC. The NPC will target the player, and once they have done so, they will shoot at the player. Even if the enemy is off screen, the player will still be attacked. This enemy can be defeated by shooting – via getting the power up that the player will be able to discover – or jumping on.

Punching NPC

Thirdly, a punching NPC. This NPC will target the player, and once they have done so, and are in proximity, an animation will play and they will attack the player. The hurt will be far more impactful than the beginning AI, to encourage the player to figure out how to defeat them faster. To defeat them, the player must ONLY jump on them.

Final Boss NPC

Through this, there will be a final NPC, the boss character, that will be a combination of all three of the previous NPCs. Taking the previous ideas into account, the player will only be able to defeat them by using the tactics they have learnt previously.

Section V – User Interface

(By Josh Shepherd)

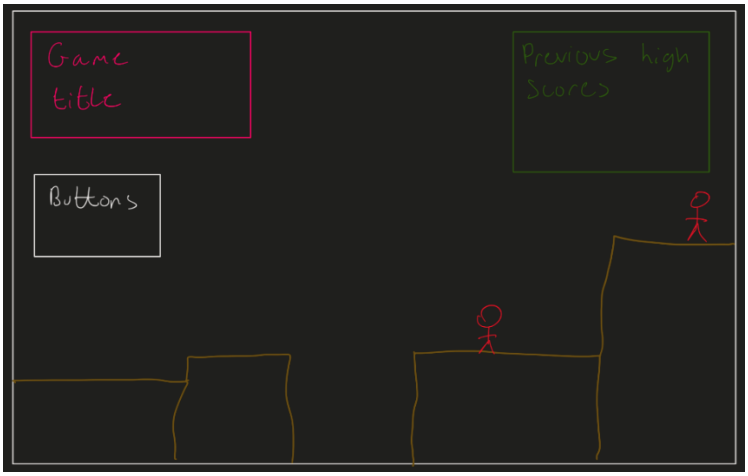
Josh Shepherd 1700471
Savannah Coward 1801678

Overview

The user interface in the game will have two main components; the main menu and all of its child interfaces and the player's interface which can be seen when the game is in progress. The player's interface will contain the vital information needed for the player to see during play and will also display the final score and progress buttons once the player has completed the level or has died.

Main Menu

As the main menu is the first element the player will see when starting the game, it will be the screen that contains the most refinement and joy to look at. The main menu will contain buttons, as the most prominent elements, to allow the player to navigate around the game, such as to start the game and exit the game. The background of the main menu will be a slow-moving camera that start at the beginning of the level and will pan over the level and restart once near the end. The menu will also contain the game's logo or name at the top and be the largest element on screen.

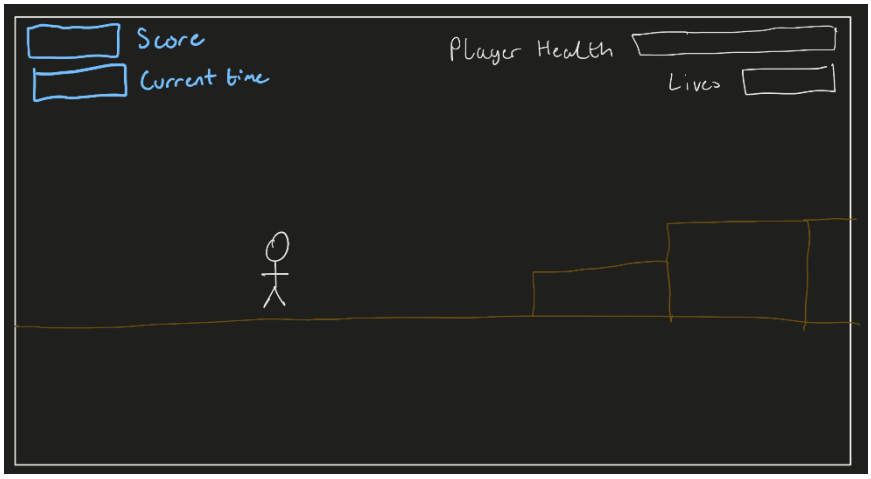


Initial layout of the main menu

Also, on the main menu, located in the top right corner, will be the players previous high scores, the date they achieved it on and their time for that score.

Gameplay Interface

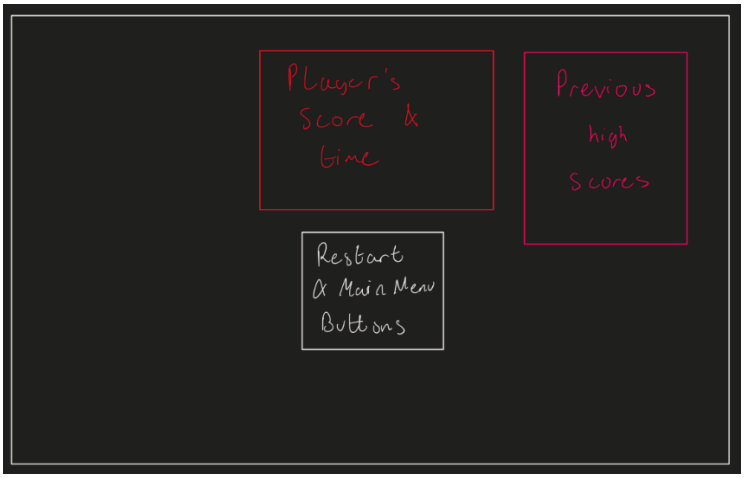
The player will have two components on screen during regular gameplay. The current score of the player, located in the top left, and the player's current health bar and their total lives remaining, located in the top right. The interface will be updated every frame with the latest values drawn from the player character.



Initial drawing of the player's user interface during gameplay

Game Over/Ending Interface

Once the player has lost their total lives or has completed the game by defeating the boss, then the game ending user interface will be displayed. This interface will display the player's score and time in a more prominent place in the centre screen. The interface will also display any best scores the player has previously had to the right side of the screen to allow the players to compare against their previous attempts. Finally, the interface will also contain two buttons, one to restart the level and retry the game again, and another to return to the main menu. The background will be the last frame of where the player was before the interface appeared. For example, if the player died as a result of an enemy, then the background would be the player's death position with the enemy continuing their AI routine.



Draft of the game ending user interface

Interface Technical

All of the interface elements can be split up and contained within their own widgets, which helps to improve maintaining and amending any elements when needs be. All elements should be designed and created using the Unreal Engine 4's UMG designer

Each interface screen can inherit from the same base HUD class that has a `TArray<UUserWidget*> UIElements` property, which gets populated in the UI with the elements

that will be shown for that screen. Each HUD should be created as a blueprint so that we are able to set the elements in the editor.

All of the player's gameplay elements can inherit from the same widget class that contains C++ code to interact with the main character's class through helper functions. The functions should be made blueprint callable so that the elements can be updated in blueprints. Each element can then access and call any child functions to get access to the player's data required to update the UI. The current score and timer widget elements should also inherit from the same class and have helper functions that interface with the class that stores score and time.

The high score UI element can be reused between the main menu and the game ending menu. This element should inherit from a separate widget class that is implemented in C++ that contains helper functions that access any previous high score data.

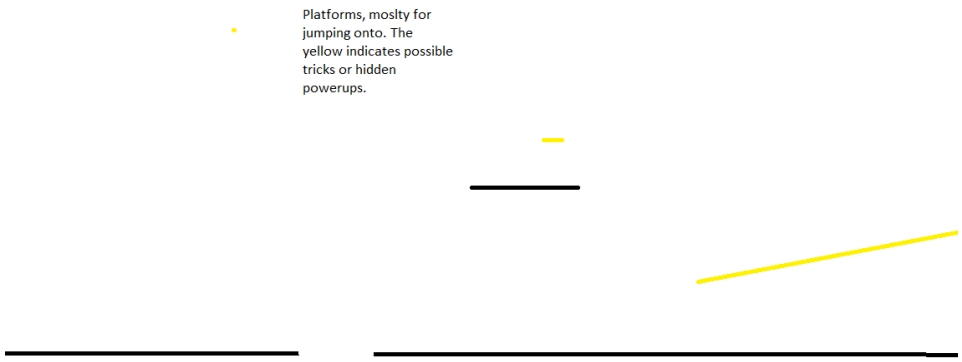
Section VI –Level

(By Savannah Coward)

Overview

We want the player to achieve completion of the level, AKA, "escape". There will be some puzzles attempting to ease the user into how to discover items. These puzzles will include mostly hitting blocks, which will shoot out an item. As stated, there will also be puzzles around figuring out what the player has to do to defeat enemies. The primary focus will be puzzles on working out the enemies. The player will also have to figure out several jumping situations and some climbing ones, aside from the box.

Level Design/Features



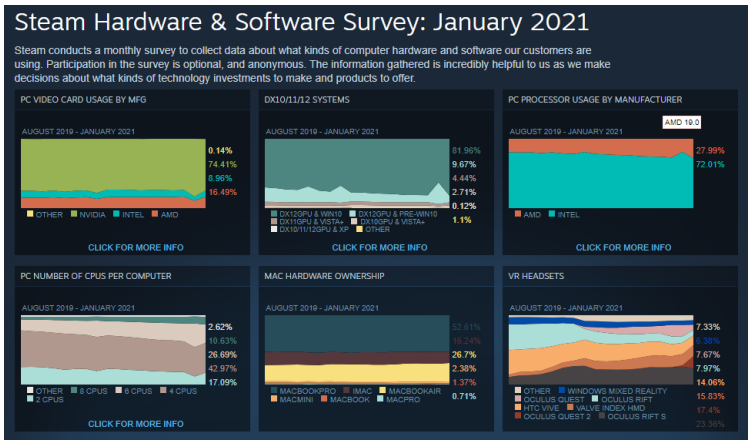
A basic look at the ideas, on a 2D look. There will be jumps, where the AI will register and not fall into. The check point will be towards the end, where the flag will spawn the boss. There is one indication of a powerup box, that the player will go under and jump up to get a random generated power up.

Section VII – Technical Specification

(By Josh Shepherd)

Target Hardware

As the PC market is vast and contains many gamers, the hardware for each person is never the same and can contain a massive variation. The decision on which hardware to aim for is based on the most popular games distribution service on PC, Steam, as they have an optional survey which gathers each person's gaming configuration and publishes it for all to see.



Steam's Hardware & Software survey results for January 2021

Because of the survey's outcomes, we aim to deliver the game using a low to mid-range specified PC, with an Intel CPU with 4-6 cores, one GPU and is using Windows 10 & DirectX 12. By limiting ourselves to just the PC market for now, it will be easier for us for debugging and solving any user problems and also simplify delivery since Unreal Engine is developed using a PC.

Game Engine & Language

The game engine we will use will be Unreal Engine (version 4.22.3) which utilizes Blueprints and C++ for coding. We will primarily use C++ for coding any elements of the game, like gameplay elements such as the player, enemies, projectiles, and any core functionality. Blueprints will only be utilized and contain logic for small and unrelated core gameplay such as UI menus. However, Blueprints will be used to set variables that will be customized and changed often to fine tune and tweak behaviour.

Game Delivery

We aim to develop the game solely for the PC market and will utilize game delivery services, such as Steam, to begin with and gradually adding more services such as Origin, Epic Games Store, Uplay and GOG. We can also distribute the game DRM-free such as handing out the game, packaged as a zip file, for free to players we choose.

Section VIII – Appendix

(By Savannah Coward)

Josh Shepherd 1700471
Savannah Coward 1801678

Models & Art Assets

As stated in the Player section, we will use the base Unreal Engine mannequin model for Quinn, the main character. We also will aim to use:

- Checkpoint Flag
- NPC's models
- Circle projectile

Music & Sound Assets

We aim to use any royalty free music and sound effects in the game to avoid any copyright issues.

Version Control

(By Josh Shepherd)

For our project, we should use Git for version tracking and be used for development since it is the most flexible. We can use feature branches and pull requests to merge in new features, once developed. Using pull requests will also allow us to validate and check our code to make sure it is of the highest quality.

Coding Practices & Conventions

For coding, we will use a simple coding convention that is easy to follow and makes variables distinct from each other, regardless of their location.

Class Names

If the class is a blueprint, then the prefix "BP_" should be used before name to indicate that it is a blueprint. For C++ classes, we will use the Unreal Engine convention of using the "A" prefix for actors, or "U" prefix for Unreal objects.

Variables

Public and protected variables should be declared using PascalCase, which make them distinct and similar to other conventions, such as "FireSpeed". Local variables, ones that exist within the scope of a constructor or function, should be using "camelCase", for example "fireSpeed". Any private variables should also use "camelCase" style, with the prefix "m_" to indicate that it is a class-wide private variable, such as "m_fireSpeed". Any static or constant variables should use a "CONSTANT_CASE" style, where all of the characters are upper case, separated by an underscore. For example, "FIRE_SPEED".