National College of Ireland

Bachelor of Science (Honors) in Computing

Software Development

2022

Josh Rothwell

19237081

x19237081@student.ncirl.ie

Interstellar Marauder: A scrolling space Shoot’emup

Technical Report

Contents

Executive Summary

1.0 Introduction

1.1. Background

Game development is a sector of Software Development that I am quite interested in, it's a career I am currently aiming towards entering someday. For a previous NCI project, I worked on a group assignment that was geared towards game development. We developed a short proof of concept game titled 'Dark Dug', a top-down 2D horror game set in an underground cavern.

Ever since working on Dark Dug, I've wanted to revisit the idea of creating a game, and using this Software Development project as an ideal opportunity, I want to take the opportunity to develop my skillset as a developer, and improve in areas where Dark Dug either failed or didn't do as well as I would've wanted it to. Specifically creating a game that utilized the same development engine, but focusing on a new genre.

1.2. Aims

This project aims to develop a Unity game in the genre of side-scrolling space shoot’em up (or commonly known as a “SHMUP”), the base concept of the game will involve taking control of a spacecraft, shooting enemy ships, dodging obstacles and surviving until the end of the level, where the player is expected to fight a boss enemy.

1.3. Technology

The game has been developed using the Unity engine, which supports both 2D and 3D game projects, the scripts used for the Unity gameObjects will be written in C#.

Another technology used during development was the voice-emulator program SAM (Software Automatic Mouth), originally developed in 1982 by Don’t Ask Software, for systems such as the Atari systems, Apple 2 and the Commodore 64. In recent years this software has become known as ‘Abandonware’, in which a program is no longer distributed or supported by its developer or copyright holder.

This technology was used to create the text-to-speech voice of the “Master” character featured in the games first cutscene, purposefully chosen to come off as inhuman or unsettling for thematic purposes.

1.4. Structure

This document will discuss the system requirements of the project, going over various use cases, examples of the game’s functionality and codework, aswell as discussing some of the methodologies behind them.

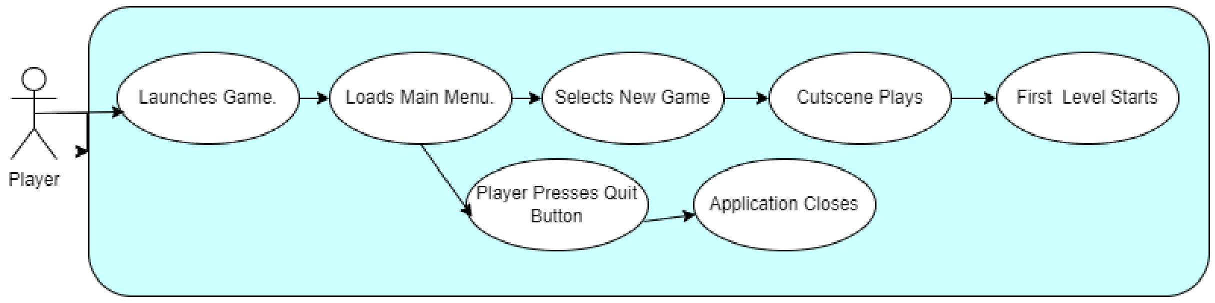
2.0 System

2.1. Requirements

Users/Controllers should be able to pick up and learn the game in around 5-10 minutes or less, if provided a proper form of tutorial or documentation on how to play the game, covering the basics such as player controls, shooting controls and mechanics and enemy mechanics.

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1 : Menu Navigation

2.1.1.3. Description and Priority

A basic expectation for application functionality, a main menu is needed to properly segway into the gameplay, rather than just dropping the player into the action abruptly.

2.1.1.4. Use Case

Scope

The use case covers arguably the first thing a user will see when launching the game, the main menu alone needs to be observed in order to begin playing the game, along with any other options included with the final product, including options, credits and more.

Description

Upon launching the game, the player will be greeted with the main menu screen, here the player can take a number of paths, they can either start a new game, navigate into the options menu, credits or close down the application. The Use Case also covers returning to the main menu via dying or through the pause menu.

Flow Description

Precondition

Player must have access to the game's executable file or have access to the Unity Project file.

Activation

The use case starts when the game has been launched, and the player loads into the main menu, and is free to choose GUI buttons.

Main flow

1. Player launches game
2. Player loads into the main menu, here there is options for new game and a button to exit the game.
3. Player selects new game.
4. Game loads the intro cutscene.
5. Upon finishing the cutscene, the first level will load.

Alternate flow

A1 : Exiting the game at the main menu

1. Player launches game
2. Player loads into the main menu, here there is options for new game and a button to exit the game.

3. Player selects the quit game button, closing the executable and returning to Desktop.

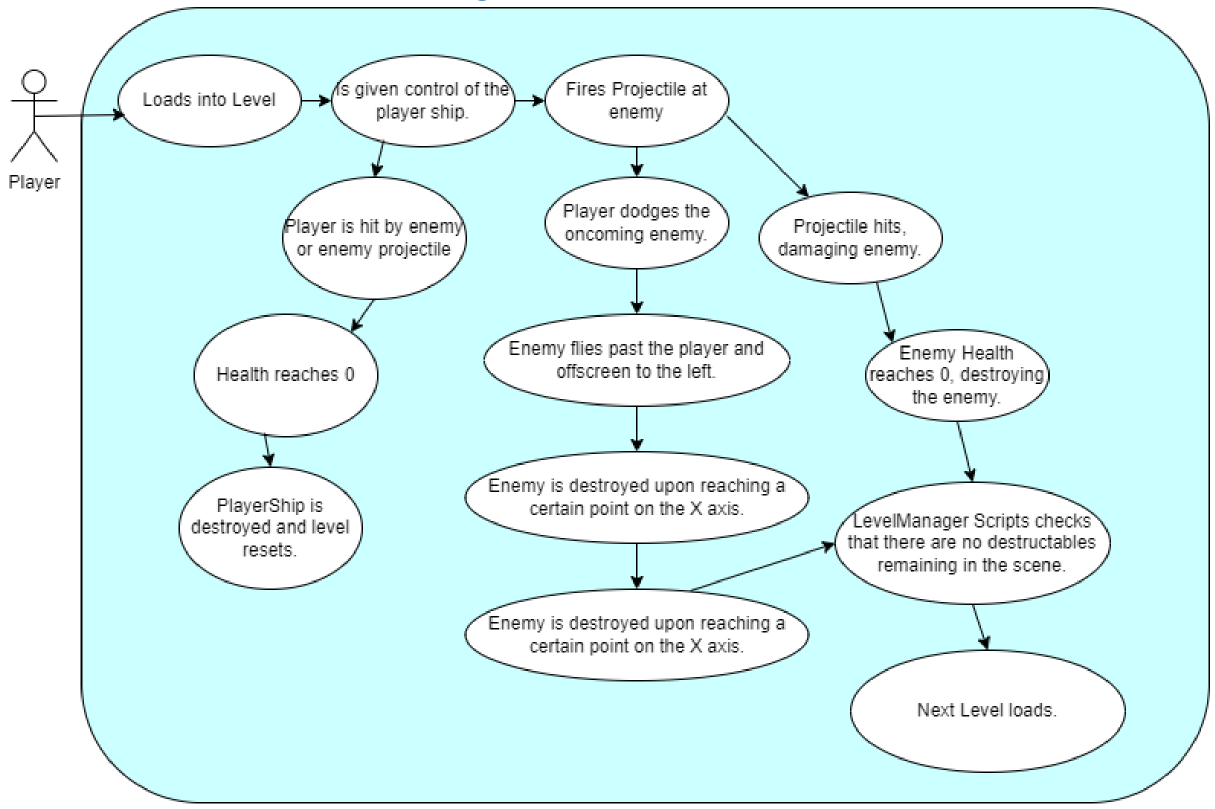
Termination

Terminated upon the alternate flow being taken, in which the game is shut down via the main menu's 'exit' option, shutting down the executable.

Post condition

The main menu is unused until the player navigates to it again, or relaunches the game

2.1.1.5. Use Case Diagram


Player
Loads into Level
Is given control of the player ship.
Fires Projectile at enemy
Player is hit by enemy or enemy projectile
Player dodges the oncoming enemy.
Projectile hits, damaging enemy.
Health reaches 0
Enemy flies past the player and offscreen to the left.
Enemy Health reaches 0, destroying the enemy.
PlayerShip is destroyed and level resets.
Enemy is destroyed upon reaching a certain point on the X axis.
LevelManager Scripts checks that there are no destructables remaining in the scene.
Enemy is destroyed upon reaching a certain point on the X axis.
Next Level loads.

2.1.1.6. Requirement 2: Combat and Level Flow

2.1.1.7. Description and Priority

Part of the primary gameplay loop featured in Interstellar Marauder, this use case describes the interactions between the player and the enemies they are required to fight against, describing two major flows, one in which the player successfully destroys an enemy, and one where the enemy inflicts damage on the player.

2.1.1.8. Use Case

Scope

This scenario will happen on almost a constant basis, the player being tested on their ability to dodge and destroy a mass influx of enemies, ranging from hostile ships to asteroids that required special ammunition to destroy. It is meant to be the most engaging part of the game.

Description

Upon loading into a level, the player is given immediate omnidirectional control over a ship, the screen will begin to automatically scroll from left to right.

The player can fire two types of projectiles, one designed for killing enemy ships, and one that is capable of killing enemy ships, and destroying harmful asteroids. These enemies and obstacles will fly from the right side of the screen to the left, in varying movements and formations.

The player can fire a projectile to destroy either of these dangers, if the player lets one of these entities collide with their ship, they will receive damage, take too much damage and the player will die, resulting in a game over.

Flow Description

Precondition

Player must have already selected to start a new game in order to load into the level.

Activation

Upon starting a new game, the user will load into the level, almost immediately being given control over their ship.

Main flow

1. Player launches new game
2. Player loads into the level.
3. Player is given omnidirectional control over the ship within the 2D space.
4. Player is able to fire two types of projectiles using the Z and X keys.
5. As the screen automatically scrolls, enemies will fly from the right to the left side of the screen.
6. If the player shoots a projectile that collides with an enemy, the enemy will be destroyed.
7. Once all enemies have been destroyed by any means, the next level will load.

Alternate flow

A1 : Receiving damage from an enemy

1. Player launches new game
2. Player loads into the level.
3. Player is given omnidirectional control over the ship within the 2D space.
4. Player is able to fire two types of projectiles using the Z and X keys.
5. As the screen automatically scrolls, enemies will fly from the right to the left side of the screen.
6. If the enemy collides with the player, the player will receive damage (Usually in the form of losing a "health state", ie 3 hits equals a game over.

A2 : Automatic destroy

1. Player launches new game
2. Player loads into the level.
3. Player is given omnidirectional control over the ship within the 2D space.
4. As the screen automatically scrolls, enemies will fly from the right to the left side of the screen.
5. Players can steer their ship to avoid enemies and their projectiles, enemies that fly off screen will be automatically destroyed.

6. Once all enemies have been destroyed by any means, the next level will load.

Termination

Terminated upon the alternate flow being taken, in which the game will end when the player receives too much damage and receives a game over, as discussed in the previous use case, the player can either restart the level or exit the game.

Post condition

Enemies are also destroyed if they are allowed to fly beyond the left side of the screen, allowing the player to simply avoid enemies without having to shoot them.

2.1.2. Data Requirements

Data requirements for the project will involve the tracking of player data accumulated throughout the game, this includes tracking player health, progress through the level(s), and potentially other factors such as player ammunition.

2.1.3. User Requirements

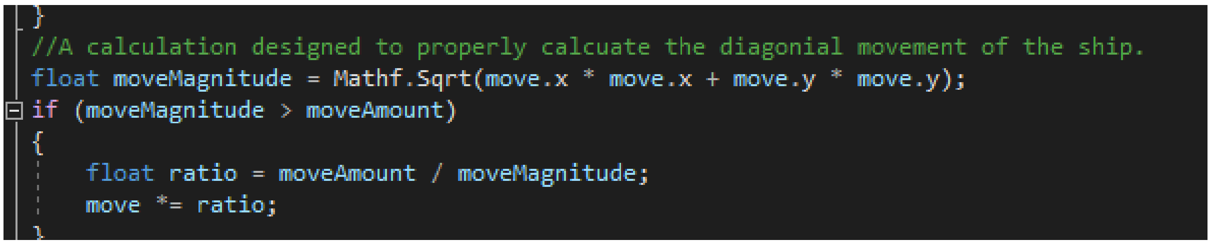
The user will require proper documentation or tutorial material on how to play the game, this will be necessary if the player is expected to quickly adjust to the mechanics controls of the game.

2.1.4. Usability Requirements

The game needs to be designed to be accessible to as many potential players as possible, a number of considerations come to mind to mind such as age rating or content accessibility, ensuring that content is appropriate for certain age groups. Hardware accessibility requires the game to be optimized so that it can run on a wide array of hardware with varying specifications. In this case the current scope for hardware is personal computers.

2.2. Design and Architecture

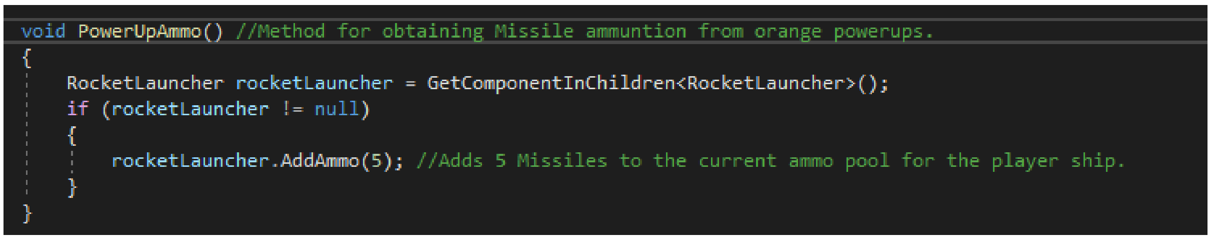
The project utilizes a number of Unity gameObjects stored inside of 'Scenes' to make up the various components of the game, for example the ship that the player controls consists of an invisible entity that houses the C# script used for player interaction, attached to this blank object is the 2D Sprite that the player can see, along with separate game objects for the ship's various weapons and shield ability. An example of the coding involved with the ship involves a calculation used for movement, to prevent the player from moving faster than intended by moving diagonally, in either direction while moving up or down.



```
}
//A calculation designed to properly calcuate the diagonial movement of the ship.
float moveMagnitude = Mathf.Sqrt(move.x * move.x + move.y * move.y);
if (moveMagnitude > moveAmount)
{
    float ratio = moveAmount / moveMagnitude;
    move *= ratio;
}
```

Figure 1: Code implemented that calculates the diagonal movement of the player ship.

Another example of the PlayerShip script is a snippet from the code used to obtain ammunition from powerups. This method is called upon when the player collides with an object with the PowerUp script attached, and will check to see what kind of effects it will grant the player.

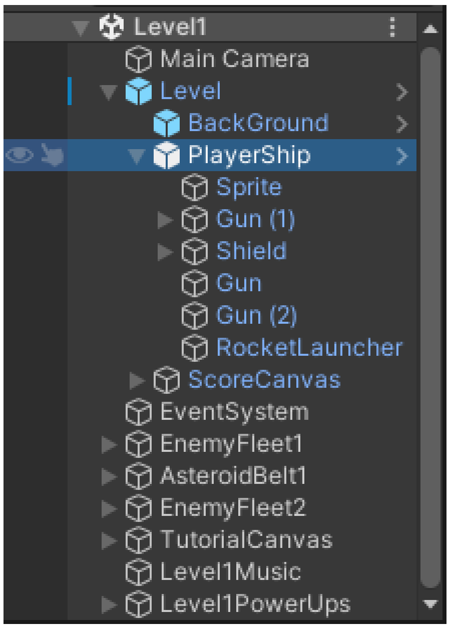


```
void PowerUpAmmo() //Method for obtaining Missile ammuntion from orange powerups.
{
    RocketLauncher rocketLauncher = GetComponentInChildren<RocketLauncher>();
    if (rocketLauncher != null)
    {
        rocketLauncher.AddAmmo(5); //Adds 5 Missiles to the current ammo pool for the player ship.
    }
}
```

Figure 2: A method for adding 5 Missiles to the players rocket ammo, this method is called upon later in the code. Code also references another game component known as 'RocketLauncher'.

2.3. Implementation

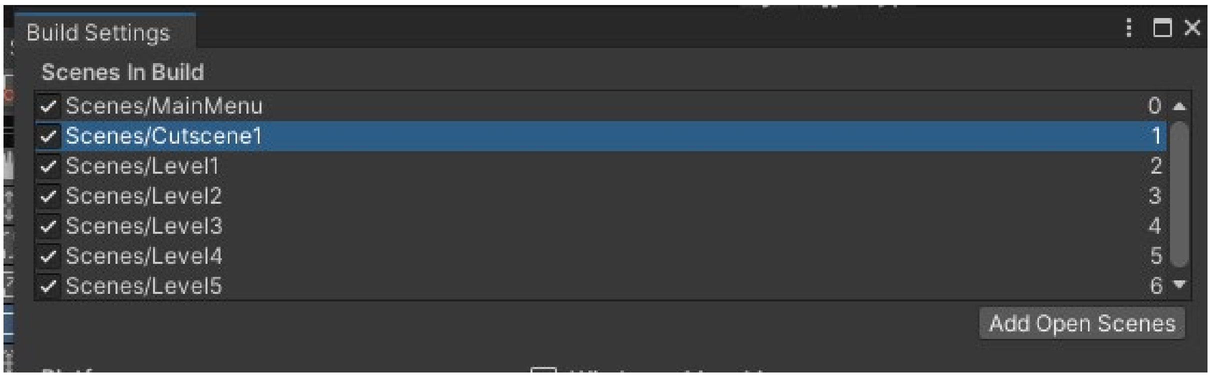
As mentioned earlier, the game is made up of 'scenes' that contain all the necessary components to make the game function, all gameplay sections of the game utilize similar if not the same layout and implementation of components, A level manager which contains the Player Object, scrolling background and score. The level manager is responsible for keeping track of these objectives and carrying them over into the next loadable scene.



```
Level1
  Main Camera
  Level
    BackGround
    PlayerShip
      Sprite
      Gun (1)
      Shield
      Gun
      Gun (2)
      RocketLauncher
    ScoreCanvas
  EventSystem
  EnemyFleet1
  AsteroidBelt1
  EnemyFleet2
  TutorialCanvas
  Level1Music
  Level1PowerUps
```

Figure 3: Hierarchy view of the game objectives contained within level 1.

Levels themselves are connected in a few ways, the first is setting each level to load in the game's build settings, this is mandatory in order to access these scenes outside of the editor.



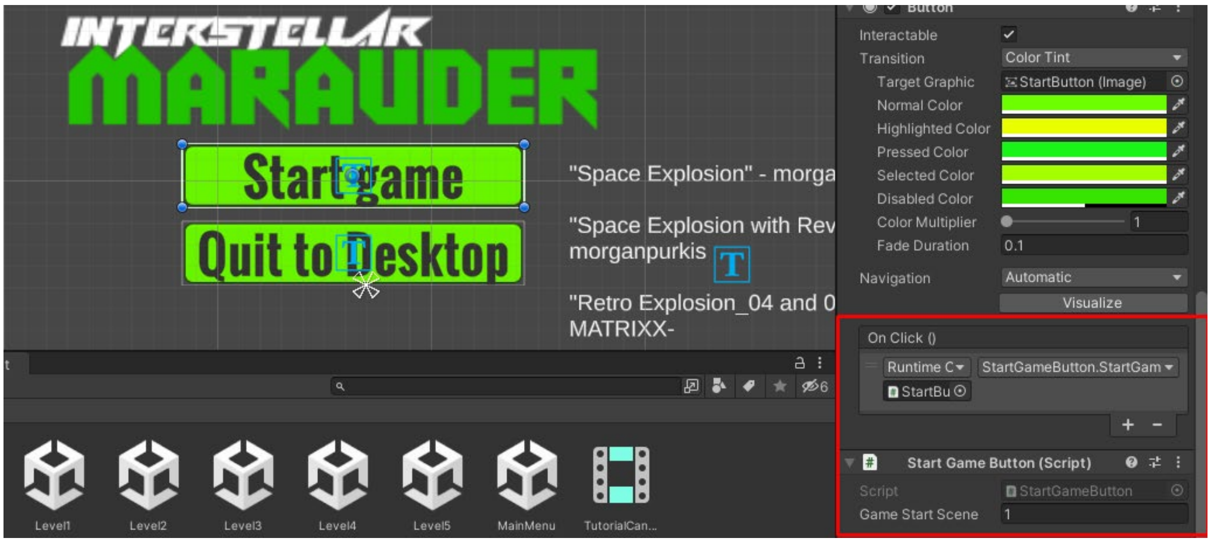
Build Settings

Scenes In Build

✓ Scenes/MainMenu	0
✓ Scenes/Cutscene1	1
✓ Scenes/Level1	2
✓ Scenes/Level2	3
✓ Scenes/Level3	4
✓ Scenes/Level4	5
✓ Scenes/Level5	6

Figure 4: View of the Build Settings, displaying all of the games' scenes set to load upon building.

Next, I used a mixture of different methods for loading certain scenes, with the main menu being set to the first scene that loads, I implemented what is known as a "OnClickEvent" on the button that starts the game, this works by creating a script that will load a scene by its number (Refer to the image above, which displays all scene order numbers on the right side.), this script is then applied to the ClickEvent as seen in the image below.



Button

Interactable ✓
Transition Color Tint
Target Graphic StartButton (Image)
Normal Color 🟩
Highlighted Color 🟨
Pressed Color 🟩
Selected Color 🟨
Disabled Color 🟩
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
Visualize

On Click ()
Runtime C StartGameButton.StartGam
StartBu

Start Game Button (Script)
Script StartGameButton
Game Start Scene 1

Figure 5: An example of a OnClick() event used in the main menu.

Another example of scene management is through the Timeline component, which is featured in the game's introductory cutscene. The timeline component functions much like most video editing programs, featuring a timeline that can contain video and audio inputs. These inputs can be given keyframes and in Unity's case, carry out specific scripts upon initialisation.

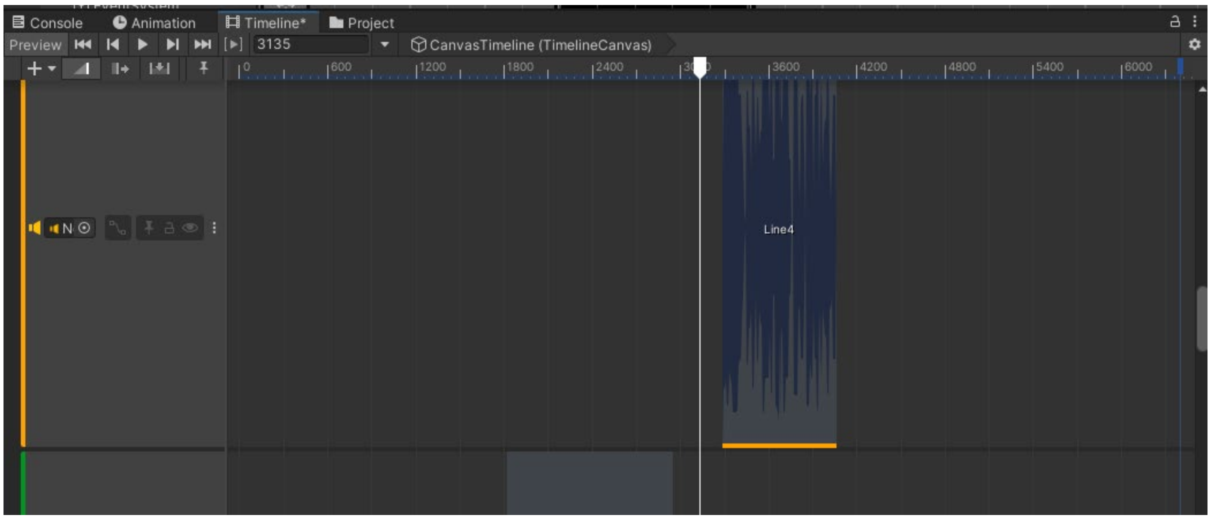
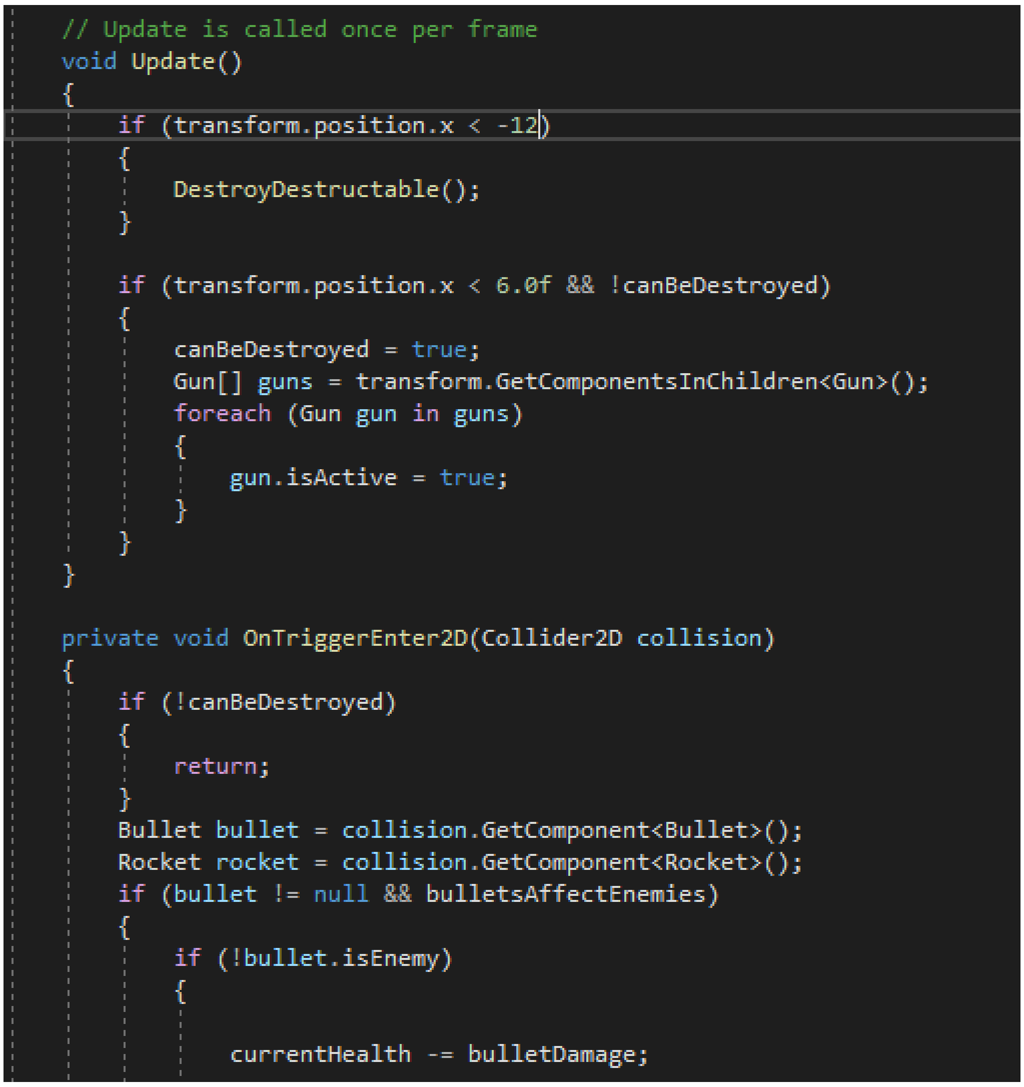


Figure 6: The timeline componenet used in the game's intro cutscene.

The next level is initialized through a gameobject that when initialized by the timeline, ends the current scene and loads the next one.

All other scenes run off of a script that checks for the number of "Destructable" objects, these includes enemies, bosses and power ups. Once the script confirms that there are zero destructables left on the scene, the next scene will load.

A scene can end two ways, when every Destructable has been destroyed by the players weapons, or if they fly off the screen and hit a specific point on the X axis, as seen in this code.



```
    // Update is called once per frame
    void Update()
    {
        if (transform.position.x < -12)
        {
            DestroyDestructable();
        }

        if (transform.position.x < 6.0f && !canBeDestroyed)
        {
            canBeDestroyed = true;
            Gun[] guns = transform.GetComponentsInChildren<Gun>();
            foreach (Gun gun in guns)
            {
                gun.isActive = true;
            }
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (!canBeDestroyed)
        {
            return;
        }
        Bullet bullet = collision.GetComponent<Bullet>();
        Rocket rocket = collision.GetComponent<Rocket>();
        if (bullet != null && bulletsAffectEnemies)
        {
            if (!bullet.isEnemy)
            {

                currentHealth -= bulletDamage;
```

Figure 7: Snippet of DestroyDestructable Method and 2D onTrigger method.

2.4. Graphical User Interface (GUI)

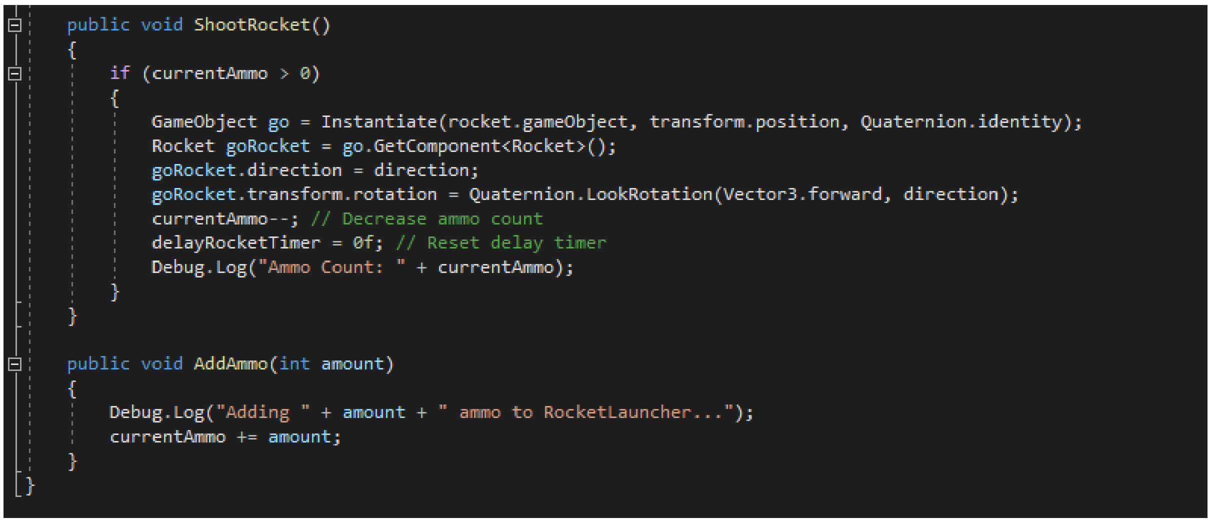
The presence of GUI within Interstellar Marauder is rather minimal, the main two examples being the previously discussed main menu, and a score tally that keeps track of the points a player amasses whenever they destroy an enemy with any of their weapons.

2.5. Testing

During development, Unity's various forms of debugging tools were a great help in tracking down issues, one particular example of debugging that was utilized was during the development of a mechanic that gives players limited ammunition and allows them to reload part of their ammo with a power up.

Making the ammunition was relatively easy, the difficulty came in getting the power up to work with the ammo system, since the missiles and the power ups did not share the same script.

While attempting various potential fixes, I modified the rocket launcher code to print out a debug statement, which would show the current ammo count, and a statement for whenever the power up was picked up, displaying the number of missiles added to the ammo pool.



```
    public void ShootRocket()
    {
        if (currentAmmo > 0)
        {
            GameObject go = Instantiate(rocket.gameObject, transform.position, Quaternion.identity);
            Rocket goRocket = go.GetComponent<Rocket>();
            goRocket.direction = direction;
            goRocket.transform.rotation = Quaternion.LookRotation(Vector3.forward, direction);
            currentAmmo--; // Decrease ammo count
            delayRocketTimer = 0f; // Reset delay timer
            Debug.Log("Ammo Count: " + currentAmmo);
        }
    }

    public void AddAmmo(int amount)
    {
        Debug.Log("Adding " + amount + " ammo to RocketLauncher...");
        currentAmmo += amount;
    }
}
```

Figure 8: Code depicting the method used for firing a missile, note the debug messages that print out the current number of ammo after the missile has been launched, and the debug message for whenever ammo is picked up.

Another example of debug testing was when developing the aiming mechanic for Turret enemies, a debug message was created that would print out the rotation value of a turret object, helping me determine whether the gun was actually turning in place to aim at the player.

2.6. Evaluation

While I feel the game is too early to be truly evaluated on its playability, I feel I can get a sense of where I want to take the game in terms of functionality, and so far I'm liking it. Experimenting with this crude prototype has given me ideas that I did not initially have when proposing the project.

3.0 Conclusions

The game is being based on a tried and tested genre, these kinds of games have been around since the height of arcade popularity and still continue to thrive in varying forms, so theoretically the majority of the conceptual work has been done for me, I am not taking any major risk when choosing the genre, but Moreso when it comes to developing and adding onto the concept of a shoot'em up. These kinds of games are also very approachable to the average player, they are so accessible that they are often advertised as mobile or browser games, something that most people can pick up and learn to play within minutes.

However, the simplicity of these games can also be a deficit, without in-depth experience, time and resources there is only so much you can do with the concept of a 2D space-themed shooter, the concept of Interstellar Marauder most likely isn't going to be revolutionary, but it at the very least be engaging to play.

4.0 Further Development or Research

After playing around with the concept, there are some ideas that have sprung to mind that were not part of the initial proposal, while working on the shooting and enemy features, an idea sprung to mind. The game currently handles weapon as an array, meaning it can handle multiple forms of weapons for the one player character, and with the methods used to create enemy, there can exist different types, each with their own rules or scripting.

I plan to pursue this concept by having Asteroids appear throughout the level as an obstacle that either cannot be destroyed by any means and must dodged with the player controls, or to introduce a special weapon type that can destroy asteroids, but only exists in a limited quantity. This should add a new level of a challenge and engagement to the concept of flying through space shooting enemies. This type of challenge is reminiscent of a recently emerged genre of shooter games known as "Bullet Hell", this refers to games that involve dodging ludicrous amounts of projectiles while shooting at enemies, the "hell" part of the name refers to their often notorious levels of difficulty.

5.0 Appendices

5.1. Project Proposal

National College of Ireland

Project Proposal

“Interstellar Marauder: A side-scrolling space shoot’em up”

30/10/2022

Bachelor of Science (Honors) in Computing

Software Development

2022

Josh Rothwell

19237081

x19237081@student.ncirl.ie

Contents

6.0 Objectives

The primary objective of this project is to develop a space/science fiction themed side-scrolling 'shoot'em up' game. End result should resemble a game in which player takes control of a character/spacecraft capable of moving across the screen as it automatically scrolls to the side, the primary gameplay loop involving shooting at oncoming enemies and obstacles.

This project also aims to expand on the complexity of the game created by the end of development. Unlike previous game development project(s), this game focuses on being more action-oriented, with more in-depth mechanics. Goals for gameplay include more than one enemy types, multiple levels and atleast one boss-fight encounter. Other possible gameplay features I want to consider include character/ship customization, either purely cosmetic or including different ships that function slightly differently, letting players choose a preferred playstyle.

Some extra objectives I want to accomplish with this project include implementing Unity controller support, since my previous game project was only controlled through the mouse and keyboard. I want to experiment with the concept of controller support, since the last project I am now in possession of a controller I can use for developing and testing.

7.0 Background

Game development is a sector of software development that I am quite interested in, it's a career I am currently aiming towards entering someday. For a previous NCI project, I worked on a group assignment that was geared towards game development. We developed a short proof of concept game titled 'Dark Dug', a top-down 2D horror game set in an underground cavern.

Ever since working on Dark Dug, I've wanted to revisit the idea of creating a game, and using this Software Development project as an ideal opportunity, I want to take the opportunity to develop my skillset as a developer, and improve in areas where Dark Dug either failed or didn't do as well as I would've wanted it to. Specifically creating a game that utilized the same development engine, but focusing on a new genre.

Like before, the game will be developed in the Unity engine, with the nature of the game being a horizontal scrolling shooter, most likely developed as 2D game once again, featuring sprites and tilesets. The plan is to establish the basic mechanics such as flying, shooting, screen scrolling and basic enemy interactions before expanding onto the more complex mechanics, such as multiple enemies, weapons, levels and boss encounter(s).

8.0 State of the Art

The idea of 'Interstellar Marauder' is inspired off classic scrolling shooters such as Gradius or R-Type. These types of games were abundant in arcades and classic consoles, in recent years the genre has evolved over time and moved onto platforms such as mobile, taking on different forms such as 'Bullet Hell' games, in which players are subjected to highly difficult gameplay involving dodging high quantities of projectiles. The main differences these games possess is usually down to controls, visuals, or difficulty. Some games of this genre rely on difficult challenges to stand out from the competition, others deliver appealing graphics and a solid soundtrack to become memorable.

A potential way to differentiate this project from others in the genre is through the story of the game. A common pattern I have noticed is that the story involves an underdog protagonist going up against a greater enemy, effectively an underdog serving as the last line of defence against a hostile power. I propose doing a role-reversal, in which the players takes control of a ship piloted by one of the antagonists, acting as a hostile invader with the explicit purpose of securing objectives for the villains of the story. This allows for unique storytelling potential, through cutscenes, dialogue or environmental storytelling as the game can take on darker tone as your character effectively becomes the villain of the story, even the name of the project 'Interstellar Marauder' refers to the characters main title as an invading force.

9.0 Technical Approach

Development will most likely take on a linear development life cycle, requirements have been identified based off of expectations of the genre, what I am interested in accomplishing with the project and what I feel I can reasonably achieve but still aim to challenge myself with.

Since this is an individual project, I will be taking on the sole responsibility for all aspects of development, and have thus categorized project tasks and requirements into areas such as gameplay mechanics and Quality of Life additions. Gameplay comes first and will thus have priority over visuals, audio and storyline aspects. Milestones being dictated by the state of individual features, such as the completion of a mechanic, or the early implementation of a new feature.

Much like my previous game project, Interstellar Marauder will receive a dedicated GitHub repository, this can be used for sharing the game build among individuals I recruit for game testing, the GitHub repository can also be used for portfolio purposes and housing testing repositories through GitLab.

As will be elaborated on in the project plan, I plan to conduct weekly and monthly reviews of my progress on development with the aim of tracking the rate at which I achieve objectives or milestones, allowing me to adjust my priorities and development philosophy as development progresses.

For any difficulty that may arise during development, which most likely will as I will be treading into mechanics I have less experience with. I know from previous experience what information sources I can go to to seek advice and guidance from other Unity developers on development forums or Unity's official tutorial and informational material.

10.0 Technical Details

Going into the project with previous experience, Unity will be the chosen engine for developing the bulk of the project, it will contain the game's coding and executable. The primary coding language that will be implemented for this project will be C#, classes being edited using Visual Code Studio.

Much like my previous project, Unity plugins will be implemented to help streamline the process of adding multiple features, aswell as for modifying numerous game elements quickly and efficiently.

Despite the chosen style for the game being 2D pixelart, the versatility of the Unity 2D engine and the wide variety of space-themed assets, there is potential for implementation of basic lighting and particle effects which can be implemented as a quality of life feature.

Unity games often function as "scenes", separate screens encompassing separate parts of the game such as menus, cutscenes and the levels that encompass gameplay. Each screen containing separate properties, being tied together through scripting that allows players to navigate to a different scene.

11.0 Special Resources Required

Visual assets will have to be either taken from the Unity asset store, or created using art program outside of Unity, previously when creating new sprite work, I worked inside the art program 'Gimp' to create and edit tilesets and sprites.

Sound and music material will also need to be accessible, most likely also through the Unity asset or other royalty-free or public domain sources.

Work has already begun on researching information on the required assets, including browsing through the Unity store and other asset repositories for material that can be used easily in the project.

12.0 Project Plan

The project plan began around early October, with the original conception of the project idea, multiple ideas were considered before the final idea of Interstellar Marauder was chosen. The rest of October was spent working on the requirements defining, aswell as the initial groundwork such as beginning to setup the Unity workspace.

The goal is to have the basic core features implemented in some capacity by the end of November. Core features are defined as basic character movement, shooting mechanics, screen scrolling and atleast 1 type of AI or obstacle for the player to avoid/shoot. Once these basic mechanics have been implemented, I plan to begin refining them aswell as adding more complex features such as multiple levels or boss encounters.

I plan to have settled on a visual look for the game by the end of November aswell, if necessary, the game will use basic 2d shapes to represent the in-game objects until a look has been decided. Ideally assets will either come from the Unity Asset Store with proper credit.

By December I intend to start work on some of the extra, quality of life features such as the game's plot, which will require cutscenes of some kind, even in the form of narration or text. Next I want to begin on the implementation of controller support, since I should have the basic flight controls already implemented by that point in development.

By January/February I aim to have a solid grasp on the basic features and achieved a decent amount of progress if not better on the more complex features, it is between December and April that I aim to have enough material completed to comply with any academic inspections of progress or demonstrations, aiming to complete the project by the May deadline.

I intend to keep track of my progress through weekly and monthly self-reviews. With my monthly goals set up I plan to use weekly objectives, small achievements to keep myself from being intimidated by the work ahead or burnt out by any lack of progress. By the end of each month I plan to carefully review each week in hindsight, reviewing the objectives I achieved and the ones I did not. Objectives that were not achieved will either be reconsidered or receive priority attention for their completion.

The bulk of development will most likely go into the core features, with the extra details such as music, story and quality of life features will come second. But if development goes smoothly there should be enough time to implement said quality of life features.

13.0 Testing

With experience in testing my previous project, Dark Dug and my time as a volunteer Quality-Assurance tester for an online video game, I can say with confidence that I will be able to adequately test my project. The primary form of testing will come in the form of functionality testing, simply ensuring that the game itself is playing as intended, this form of testing I should be able to accomplish on my own.

Where I may require a second person however is the task of playtesting the game, seeing how others perceive the experience of my side-scrolling shoot'em up. Once I have established a basic, solid gameplay loop with enough features for other players to engage in. I will seek out play testers who would be willing to try the game out and offer feedback on the game's quality, suggestions to improve gameplay and balance on the game's difficulty.

13.1. Reflective Journals

14.0 Supervision & Reflection Template

Student Name	Josh Rothwell
Student Number	19237081
Course	Bachelor of Science (Honors) in Computing
Supervisor	Lisa Murphy

15.0

16.0 Month:

What? In October, I worked on the initial documentation of the project, while exploring what assets I could implement into a Unity project on a limited budget. November was spent working on the groundwork of the project, setting up the Unity Project workspace and the earliest form of scripting. December was when the major milestones were reached, such as Github implementation and the early forms of shooting and enemy mechanics were implemented.
So What? The progress of the project is going good so far, though proportionally the major strides were done running up to December, an admittedly slow and uneventful start in October. But with the current demo prototype, I'm looking to have a functioning level demonstration by January.
Now What? The goal for the new year is to have a level with atleast 2 differenet types of enemies, capable of varied movement patterns, a functioning shooting mechanic for destroying enemies, and more visual Quality-of-life improvements.

Student Signature	Josh Rothwell

Supervision & Reflection Template

Student Name	Josh Rothwell
Student Number	19237081
Course	Bachelor of Science (Honours) in Computing
Supervisor	Lisa Murphy

Month: January

What?   January started by assessing the feedback gained from the mid-point presentation, notably the areas that need to be expanded on for an actual gameplay loop to be created. Thus going forward I went ahead and setup enemy collisions and combat mechanics, allowing the player to destroy enemies, another advancement I made at this point was the ability for enemies to move vertically, this is done by incrementally moving the enemies Y position on a strict pattern, the script created allows me to set the frequency and speed which enemies move. This can be useful later on for creating enemies with different varieties of movement.
So What?   I would say the implementation of the first major strides in a gameplay loop, even just being able to fly around and fire projectiles without risk of dying helped give me a sense of what I wanted the game to "feel" like playing. Here I made some tweaks to hitboxes, sprite size and game speed.
Now What?   The challenges going forward will be implementing a form of challenge via enemy types that are capable of firing their own form of projectiles at the player, and having player be destroyed or damaged by said projectiles. Later on this can be used for other forms of danger such as the previously mentioned Asteroids and for boss fights.   Once someform of gameplay loop has been setup, I can work on other aspects of the game, such as the UI and menu interactions.

Student Signature	Josh Rothwell

Student Name	Josh Rothwell
Student Number	19237081
Course	Bachelor of Science (Honours) in Computing
Supervisor	Lisa Murphy

Month: February

What?

Early in February, I had successfully completed the goals from January, creating enemy collisions, enemy death scripts and hitboxes, I set to work on creating an ability for enemies to shoot their own projectiles. However I ran into an issue where because of how I designed the script that handles projectiles, the bullets appear to always go from left to right, even when they are scripted to go left via a Unity object parameter.

Along with this, a meeting I had with my supervisor in February also brought up a few Quality of Life issues that the game was lacking, such as animations for enemies being destroyed, sound effects and music. Bringing up how the game felt very "empty" and engaging without these.

So What?

The implementation of the basic gameplay loop in, I was able to start designing enemies that are able to shoot and enemies that merely fly towards the left, making some enemies easier to deal with than others. I also plan to try experiment with different bullet types, which can allow for different projectile speeds, sizes, and possibly even projectiles that can be destroyed, such as mines.

While the game was always intended to have sounds and effects, the meeting brought up the point that it should be made a priority, even if the sounds are placeholder until leading up to the game's completion.

Now What?

Shortly after the meeting with my supervisor, I managed create a solution to the gun problem, that being the gun is technically flipped upside down. A unconventional solution, but one that shall suffice for now.

As for the sounds and music, I will need to take the time to browse for royalty-free libraries, and find sounds that I think work best, as for explosion effects, the assets I have been using currently contain a few assets that I believe I can repurpose for an explosion.	
Student Signature	Josh Rothwell

Student Name	Josh Rothwell
Student Number	19237081
Course	Bachelor of Science (Honours) in Computing
Supervisor	Lisa Murphy

Month: March

What?

Due to the obligations of other modules, I had to dial back the work on Interstellar Marauder, but in order to make sure I wasn't making any less progress, I decided to work on less demanding aspects, such as menu GUI, sounds and music.

For music, I browsed through a number of tracks that would fit the general science-fiction theme of the game while also fitting the dark tone of the game, for sound effects I decided to go for a slightly minimalistic angle, deciding to try and go for a bit of realism by using muffled sounds for gunshots and explosions, since the game takes place in space which a vacuum. The intention is to also help bring a focus to the music, though this can always change.

I began the first implementation of the menu GUI, which will allow players to start up the game via the main menu, along with the implementation of a credits and help screen, later on I I want to try and implement an options screen, where the player can alter settings such as controls or sound volume, though this is further down the priority list.

So What?

While not necessary to the main gameplay loop, these are essential elements that help make the game feel more 'alive', what gives the game its identity in terms of aesthetic and personality. I plan to return to these aspects and give them the finishing touches once the more important factors of the game have been completed.

Now What?

My next priority is the implementation of a level system that will be integral for loading in more waves of enemies and boss encounters, boss encounters will especially be the toughest challenge yet, as they will have to be functionally different from enemies.

Once boss functions have been implemented, I will work on an implementation of powers up, introducing multiple levels before introducing a form of story via cutscenes, these cutscenes will most likely be very basic, functioning more as text scrolls or text crawls with basic imagery to convey the plot of the game.	
Student Signature	Josh Rothwell

Student Name	Josh Rothwell
Student Number	19237081
Course	Bachelor of Science (Honours) in Computing
Supervisor	Lisa Murphy

Month: April

What?

I began the stretch to the deadline by working on the script for loading multiple levels, the initial implementation was designed to be seamless, allowing the player to technically play “multiple” levels, while also carrying over the players current health, the script proceeds to the next level once all enemies have been destroyed, either by the player or when they are automatically destroyed after leaving the screen.

Next I began work on the boss mechanics, my goal is to have atleast one form of boss enemy, a large ship that will move up and down vertically, firing a variety of projectiles at the player, requiring multiple hits to take down. With multiple levels, I can tweak this boss encounter by increasing health, projectile difficulty or spawns enemies/obstacles to assist the boss, it is likely work on this will continue up until the deadline, as it will be arguably the most difficult part of the game’s development.

Next came powerups, I had two powerups in mind for the player, an attack boost that temporarily turns the players attack into a triple attack, firing 2 extra projectiles at an angle, allowing them to destroy multiple targets at once. Then a shield powerup, which will grant the player temporary invulnerability.

Then the cutscenes, cutscenes have been toned down since their original inception, as stated previously they function has text drops accompanied with music and dialogue, dialogue is currently silent, though I have the goal of implementing sound effects or text to speech, in order to bring the characters to life. The plot of the game will be conveyed through these cutscenes, following the council of overlords that the player works for, who will describe the events of the game as they unfold.

So What?

The game is definitely starting to shape up at this point, with the majority of the game’s primary challenge and mechanics implemented at this point, aside from the final major hurdle of boss fight encounters, the rest of the work will be focused on features that can be considered quality of life updates, such as the music, sound effects and visuals.

Now What?

As previously stated, the priority will be on completing the boss mechanics, as they will be the most challenging part of the games development yet, I have already begun work on their earliest implementation, but as it stands there will most likely be less variety in boss encounters than originally envisioned. Afterwards,

all remaining work will be put on implementing quality of life changes, going back and revising previous additions to the game.	
Student Signature	Josh Rothwell