FOX 2D

Gamedev Group

Ian Peña, Madhava Muli, Manasa Sangana, Ashish Goli, Audrey Warrene

Contents

1. Introduction

1.1 Project Statement and Functionality

The video game will be a 2D platformer, which is a genre of video game that combines horizontal movement mechanics with obstacles that challenge the player's movements. The objective is to move from left to right until the level is completed. There will be multiple levels, with the user being able to select a level from a main menu screen. The final level should sufficiently test the player's skills acquired from previous levels. To progress throughout the game, the player will have a variety of abilities that make the challenges possible. These abilities include walking forwards and backwards, jumping, and crouching. The control keys are bound to both the "WASD" keys and the directional arrow keys. The camera will follow the player character horizontally and vertically as they move throughout the level. The level is cleared when the player reaches the end gate, marked by the checkered flag. If the player dies, at any point within the level, they are sent back to the beginning of the level. Otherwise, they can advance to the next level. The player character can be hit up to 3 times, with the third hit killing the player character. Some obstacles kill the player instantly. A health counter system is in place to show the current health of the player. The game will contain 16-bit style graphics and audio. The game will be developed with the Unity game engine using the built-in C# implementation.

1.2 Performance Issues:

Because this game will make use of pixelated 16-bit style textures, it should not be too demanding in terms of performance. Any computer made within the last few years that is running a modern operating system should be capable and compatible. That is not to say that performance will be unimportant – if multiple objects appear on screen at once, such as high quantities of enemies or projectiles, then the game has the potential to slow down, especially on systems that lack a dedicated graphics processing chip. Additionally, we will need to consider how we want to handle network performance, specifically relating to the multiplayer component of the game. If we decide to implement peer-to-peer networking, then latency becomes a concern. In a situation where one player has a significantly slow connection speed, we will need to ensure that their delayed/inconsistent inputs are processed in a way that is fair to the other player. If a player’s connection is dropped, then the other remaining player should be able to continue playing seamlessly. The disconnected player should be able to reconnect either at the remaining player’s position, or at least, at the end of the level. It is extremely likely that we will need to actively benchmark performance and network usage throughout the project to ensure that the game plays at an enjoyable speed. Luckily, many game rendering engines, including Unity, offer performance benchmarking software solutions for this very purpose. Most modern operating systems also have built-in performance metrics that assist with benchmarking.

2. Project Estimation

2.1 Background

This game is one of the first end-to-end software projects for many of the members of this group. Coordinating development efforts and meeting times between four different people also adds to the complexity of the project. Considering these factors, we roughly estimate that it will take 190 hours to complete this project. The optimistic estimate is 160 hours, and the pessimistic estimate is 250 hours for total development time. The mode estimate is also 190 hours. This number is an aggregation of the different amounts of time that each group member estimated the project would take. As a result, the estimated amount of work per individual is six hours per week.

2.2 Estimation Techniques

Three-point estimation is the technique that was applied. This technique uses a mathematical approach as the weighted average of three estimates of the work package: optimistic, most likely, and pessimistic. This is often known as the PERT. As such, the average time of optimistic, most likely, and pessimistic estimate is 200 hours.

3. Project Risks

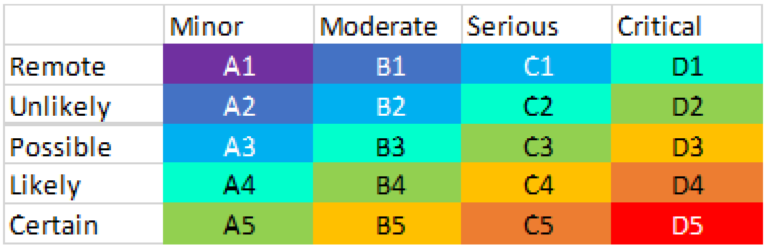
3.1 Identification and Estimation

Game development is a very complicated task, and the entire process can fall apart quickly if even one component fails. The video game industry has no shortage of projects that have financially or critically failed upon launch, let alone projects that are never even completed. As such, there are many risks involved with creating a video game. One of these risks is not being able to deliver on promised functionality. This can be a result of unrealistic expectations as to how long certain aspects of the game will take to develop. It can also result from a team not having enough development experience to accomplish their goals. There are many other risks, such as encountering mission-critical "bugs" during development. While some bugs are at worst a minor inconvenience, other times, a bug can prevent the player from completing the game entirely. Bugs can occur in the rendering of graphics, the processing of physics, the assignment of controls, and the loading of files, just to name a few examples. Creative differences between team members can also impact the direction the game takes. Story, setting, characters, graphics, gameplay features, and more are all subject to the creative choices made by the members of the team. If the members of the development team do not agree on the direction the game should take, inconsistencies in the project's design will present themselves.

3.2 Elimination

In order to reduce the risk of overpromising, we have drafted a development schedule that spans from this week until the end of the semester. We have allocated a generous amount of time to working out the more time-intensive components like networking and asset design. Our planning efforts should ensure that we have the time needed to implement nearly all if not all of the features discussed in this document. To reduce the risk of finding any serious bugs, we will test the game frequently and rigorously. At every point in the game's development, it should be played and replayed many times by different people in order to find any bugs present in the code. Bug testing is a vital part of the video game industry, with many different techniques that can be employed to find and fix problems before they become buried by newer code.

3.3 Risk Table:

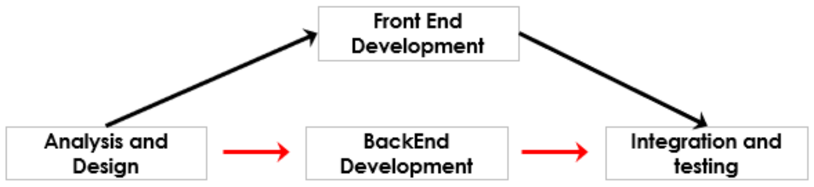


	Minor	Moderate	Serious	Critical
Remote	A1	B1	C1	D1
Unlikely	A2	B2	C2	D2
Possible	A3	B3	C3	D3
Likely	A4	B4	C4	D4
Certain	A5	B5	C5	D5

Placement	Event
D4	Running out of time to develop necessary features
B3	Technical difficulties and lack of experience in Unity
B4	Group scheduling conflicts
A3	Group members getting sick
C2	Computer crash resulting in data loss
C4	Encountering critical bugs that inhibit progress
B4	Failing to meet a project deadline
A4	Setting expectations too high
B2	Scope creep
C3	Failure to find a serious problem while testing

4. Schedule

4.1 Task Network (CPM)



4.2 Timeline Chart (Gantt Chart)

The full chart is available at the link below. Please note that it may not display correctly on the web version of Excel and should be downloaded instead.

https://uncw4-my.sharepoint.com/:x:/g/personal/ip5880_uncw_edu/
EQPSv8JJoyFCsUIqNBYKSNUBbS5awoxoLnJptQ9HONcpJQ?e=cD4c8e
[hyperlink]

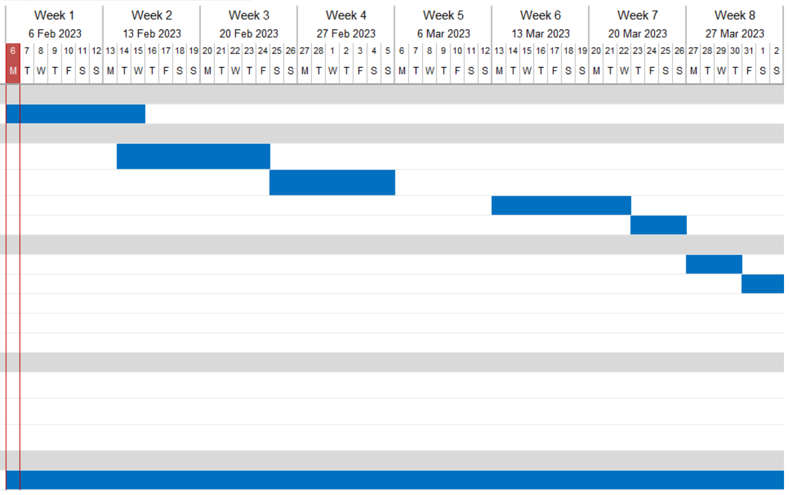
Project Schedule

[CSC550-UNCW]

Gantt Chart Template © 2006-2018 by Vertex42.com.

Project Start Date 2/6/2023 (Monday)   Display Week 1

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Analysis and Design**			-			-
1.1	Analysis and Design	[Name]	Mon 2/06/23	Wed 2/15/23	10	0%	8
2	**Front End**			-			-
2.1	Graphics,audio,other assests		Tue 2/14/23	Fri 2/24/23	11	0%	9
2.2	UI and level selection menu		Sat 2/25/23	Sun 3/05/23	9	0%	5
2.3	Movement mechanics		Mon 3/13/23	Wed 3/22/23	10	0%	8
2.4	player input		Thu 3/23/23	Sun 3/26/23	4	0%	2
3	**Back End**			-			-
3.1	networking code		Mon 3/27/23	Thu 3/30/23	4	0%	4
3.2	create class		Fri 3/31/23	Wed 4/05/23	6	0%	4
3.3	unity 2d		Thu 4/06/23	Tue 4/11/23	6	0%	4
3.4	writing unit tests		Wed 4/12/23	Fri 4/14/23	3	0%	3
3.5	valid inputs		Sat 4/15/23	Sat 4/15/23	1	0%	0
4	**Integration**			-			-
4.1	integrate front end and back end		Sun 4/16/23	Thu 4/20/23	5	0%	4
4.2	perform integration testing		Fri 4/21/23	Mon 4/24/23	4	0%	2
4.3	Perform User Acceptance testing		Tue 4/25/23	Fri 4/28/23	4	0%	4
5	**Doccumentation**			-			-
5.1	*Doccumentation*		Mon 2/06/23	Sun 4/30/23			60



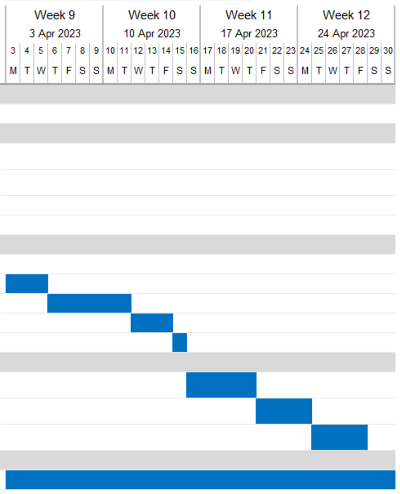
Project Schedule

[CSC550-UNCW]

Gantt Chart Template © 2006-2018 by Vertex42.com.

Project Start Date 2/6/2023 (Monday)   Display Week 9

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Analysis and Design**			-			-
1.1	Analysis and Design	[Name]	Mon 2/06/23	Wed 2/15/23	10	0%	8
2	**Front End**			-			-
2.1	Graphics,audio,other assests		Tue 2/14/23	Fri 2/24/23	11	0%	9
2.2	UI and level selection menu		Sat 2/25/23	Sun 3/05/23	9	0%	5
2.3	Movement mechanics		Mon 3/13/23	Wed 3/22/23	10	0%	8
2.4	player input		Thu 3/23/23	Sun 3/26/23	4	0%	2
3	**Back End**			-			-
3.1	networking code		Mon 3/27/23	Thu 3/30/23	4	0%	4
3.2	create class		Fri 3/31/23	Wed 4/05/23	6	0%	4
3.3	unity 2d		Thu 4/06/23	Tue 4/11/23	6	0%	4
3.4	writing unit tests		Wed 4/12/23	Fri 4/14/23	3	0%	3
3.5	valid inputs		Sat 4/15/23	Sat 4/15/23	1	0%	0
4	**Integration**			-			-
4.1	integrate front end and back end		Sun 4/16/23	Thu 4/20/23	5	0%	4
4.2	perform integration testing		Fri 4/21/23	Mon 4/24/23	4	0%	2
4.3	Perform User Acceptance testing		Tue 4/25/23	Fri 4/28/23	4	0%	4
5	**Doccumentation**			-			-
5.1	*Doccumentation*		Mon 2/06/23	Sun 4/30/23			60



5. Project Resources

5.1 People

1. Ian Peña
2. Madhava Reddy Muli
3. Ashish Goli
4. Manasa Durga Sangana
5. Audrey Warrene

5.2 Hardware

All team members will use their personal laptops to design, develop and test the various aspects of the project. Any creation or processing of game assets, such as image and music files, will most likely also be performed on the group members' personal computers. Performance testing and checking for bugs in the software will take place on two dedicated machines, one running Windows and one running MacOS. These machines and their specifications are described in greater detail in sections 16 and 17.

5.3 Software

a. **Unity:** This is the game engine platform that will utilize our code and assets. This will simplify the development process greatly by giving us access to industry-standard game development features. It will also automatically handle certain aspects of the game, such as physics and lighting.

b. **Lucidchart/Visio:** Graphical tools that were used to visualize the ideas put forth by the team members. It also aids in the process of creating project management documentation like the Gantt charts, state transition diagrams, and more.

c. **SharePoint:** File hosting and sharing service that is offered through the university's student Microsoft accounts. It offers large volumes of storage and easy access to files from any device, making it an excellent choice for team collaboration.

d. **Discord:** Instant messaging social platform for team members to communicate with when in-person meetings are not possible. Offers support for video camera and screen-sharing support, making it an ideal way for us to share ideas and assist each other.

e. **Microsoft Word:** Used to create all documentation and reports throughout the project, including all deliverables.

f. **Performance Monitor:** Used to generate the system resource usage graphs during performance testing on the Windows benchmark machine. Capable of monitoring processor, graphics card, memory, and disk usage.

g. **Visual Studio:** The IDE of choice for the creation and modification of all C# scripts in the game. This includes the handlers for movement, camera control, health management, etc. Please note that Visual Studio is a different product than Visual Studio Code.

6. Requirements Specification

6.1 System Requirements

Video games often have very particular system requirements as a result of their complexity. Multiple different software products need to be installed and configured correctly for the game to function. Our game will be developed with the Unity engine, and as such, any computer that will run our game needs to meet the minimum requirements of the Unity engine. Our game targets both Windows and MacOS platforms. Unity version 2021.3 requires at least: Windows 7 SP1 / MacOS 10.13 or higher, Direct X 10 / Metal or higher, and a 64-bit CPU with at least SSE2. Our game's handler scripts will be programmed in C# – The user will not need to have any additional software installed on their computer, as the scripts are compiled by the engine itself. The user will of course need a keyboard to control the character.

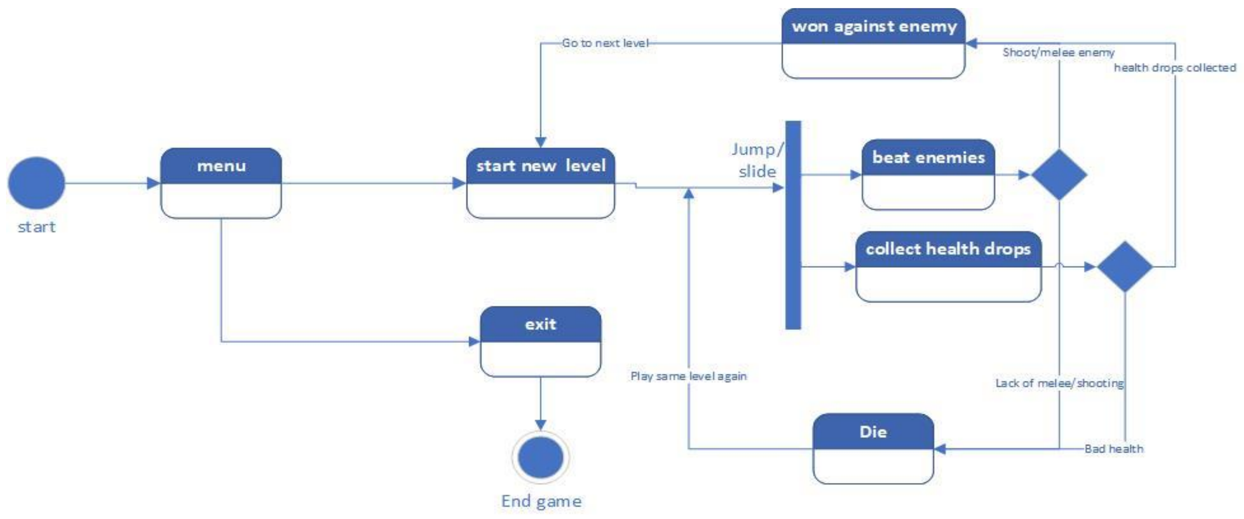
6.2 Project Constraints

One of the most significant constraints on this project is the amount of time our team will have to complete it. The scope of this project is bounded, but we plan to be flexible with the features contained in the final product. There are certain components of our plan which are not necessary for the game to function but will add great value to the project if they can be completed within the project timeframe. These additional features include networked multiplayer functionality, multiple "levels" or "stages" for the player to complete, projectile weapons that can be used by both the player and enemies, and vertical movement with ladders or "jump pads". These functions are consistent with the scope of the project, as many of them are fairly common mechanics in other platformer style games. Adding them to this project would not significantly alter the purpose that the software serves. The software should perform very well on modern computers. The biggest performance constraint would have come from any networked gameplay, as that would rely on the speed of both players' connection.

7. Functional Description

The game will require the use of the host computer’s local persistent storage for only one purpose, and that is to save the game state. When a player is done with a level, they may want to save their progress up until the current time so that they can return to playing the game at a later time without losing their progress. The game should save the player’s progress at the end of every level. There may also be a “save game” located somewhere on the level selection screen. Save files should contain a list of all levels that the player has completed so far, as well as any persistent weapons or abilities obtained throughout the course of those completed levels. When loading back into the game, the player should be able to choose their save file to restore their data. Save data will most likely be placed in a subfolder within the current user’s documents folder, as is common practice with many video games that offer a direct “save-to-file" approach of saving. This is to prevent permission/privilege/UAC issues with saving to a protected directory like “Program Files”. While saving directly to the Windows Registry is a valid option, it would make it harder for the player to access or modify their saves and would also be much more difficult to implement.

8. Behavioral Description (State Transition Diagram)



8.1 System States

The State Machine diagram is required to create a generalized map for the flow of data throughout the game design. This is subject to changes throughout the project development based on change in requirements or issues during the development process.

1. Menu selection at start of game or when exiting from game
2. Start new level – Player can begin playing by initiating any movement
3. Defeating enemies – Player attacks
4. Collecting health – Placed on map and drops from enemies
5. Won against enemy (reached end of stage) - Continue to next level
6. Lost against enemy – Player dies, retries current level
7. Exit from game

8.2 Actions

1. Sideways movement

2. Jumping and crouching

3. Collection of health drops or other items

8.3 Events

1. Repeat the same level if the player dies

2. Start the next level if the player completes the previous one

8.4 User Stories

Person A: As a person who plays video games often, I want a game that is challenging enough to keep me interested so that I can have fun while playing the game. I am familiar with basic PC control schemes in other video games and feel confident that I can learn the mechanics that this game will introduce. I would like a fair challenge so that I do not lose interest in the game.

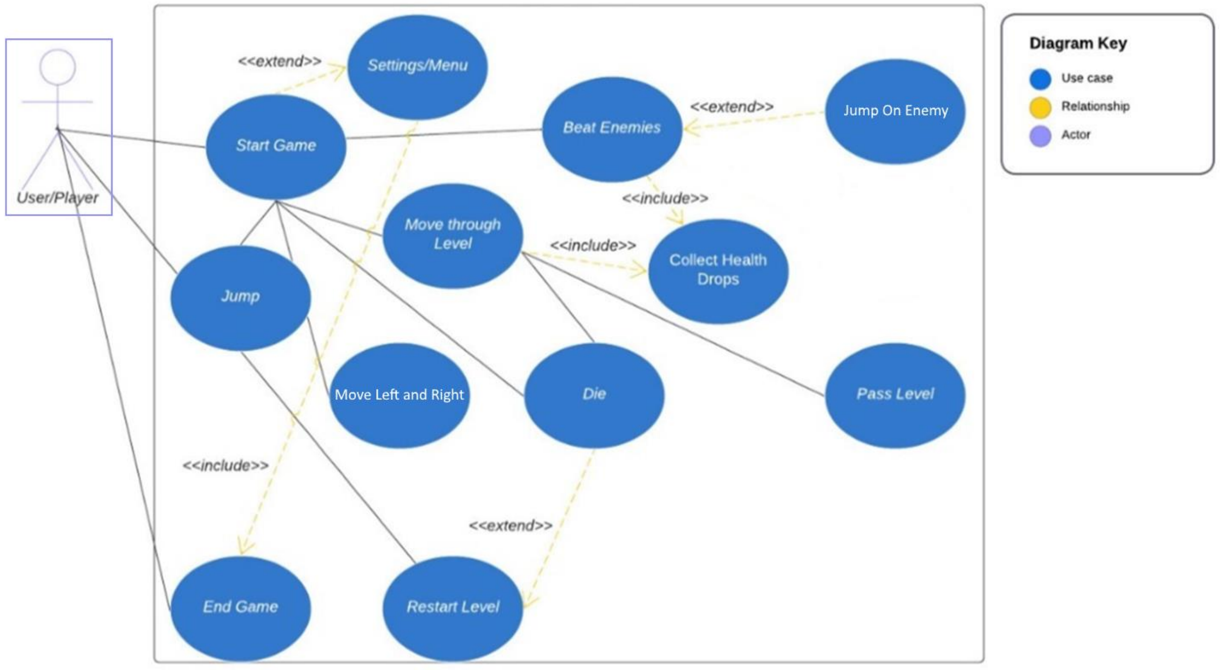
Person B: As someone who hasn't played very many video games, I want a game that is welcoming and forgiving enough for me so that I can stay engaged and have an enjoyable experience playing the game. I prefer that the controls and movement are explained and demonstrated at some point early in the game so that I can learn how to play the game at my own pace.

Person C: As someone who does not have a fast computer, I want a game that is fast and optimized so that I am able to run the game without any performance issues. The minimum specs listed in the game's description should be accurate so that people with older or slower hardware can be certain whether or not we are capable of running the game on our computer.

9. Use Cases

9.1 Use Case Diagram

The Use Case diagram is shown below to express the important operations and actors involved in the game. The figure includes an actor as the player of the game. The use case description, described in the section below, provides the details of all the use cases and flow of activities.



9.2 Use Case Description

Actors:

- User/Player
- The host device running the game (secondary actor)

Stakeholders:

- Almost all video game development projects have stakeholders. This could include publishers, intellectual property holders, or project financers. Our project, however, is an exception – We are not working on this project for a company, and as such, we do not have any traditional stakeholders.

1. Start the Game

Description:	Start the game
Actors:	Player
Preconditions:	Game is started, system is in operation.
Postconditions:	Game should finish and exit to desktop properly.
Flow:	1. Enter settings screen: You may resume gameplay or quit the game/level. 2. Move through level (either forward or backwards) 3. Jump 4. Crouch 5. Attack an enemy by jumping on them 6. Collect health from designated parts of the map 7. Die, if too much damage is taken

2. Restart the Level

Description:	Restart a level
Actors:	Player
Preconditions:	Play a level and die in the middle of the level
Postconditions:	Play the same level again till the end or die in again
Flow:	1. Enter settings screen: You may resume gameplay or quit the game/level. 2. Progress through level (either forward or backwards) 3. Die on the same level 4. Restart the same level instead of any other level

3. End the Game

Description:	End the Game
Actors:	Player
Preconditions:	Play all levels of the game or go to menu by pause button while playing
Postconditions:	Exit and go back to desktop
Flow:	1. Play through level without reaching the end 2. Pause and go back to start menu 3. Click on exit game 4. Complete all levels, then return to main menu and exit from game

<u>10. Scope</u>

10.1 System Objectives

The primary objective of our system [game] is to provide a goal for the users [players] to attempt to reach in a way that is fun and entertaining. A set of interactive challenges are provided to make it difficult for the player to reach that goal. The challenges are designed to retain the player's interest, such that the player actively seeks out more challenges throughout the duration of their playtime. If a player fails a challenge, the character onscreen will die, and a prompt will appear for the player to make another attempt.

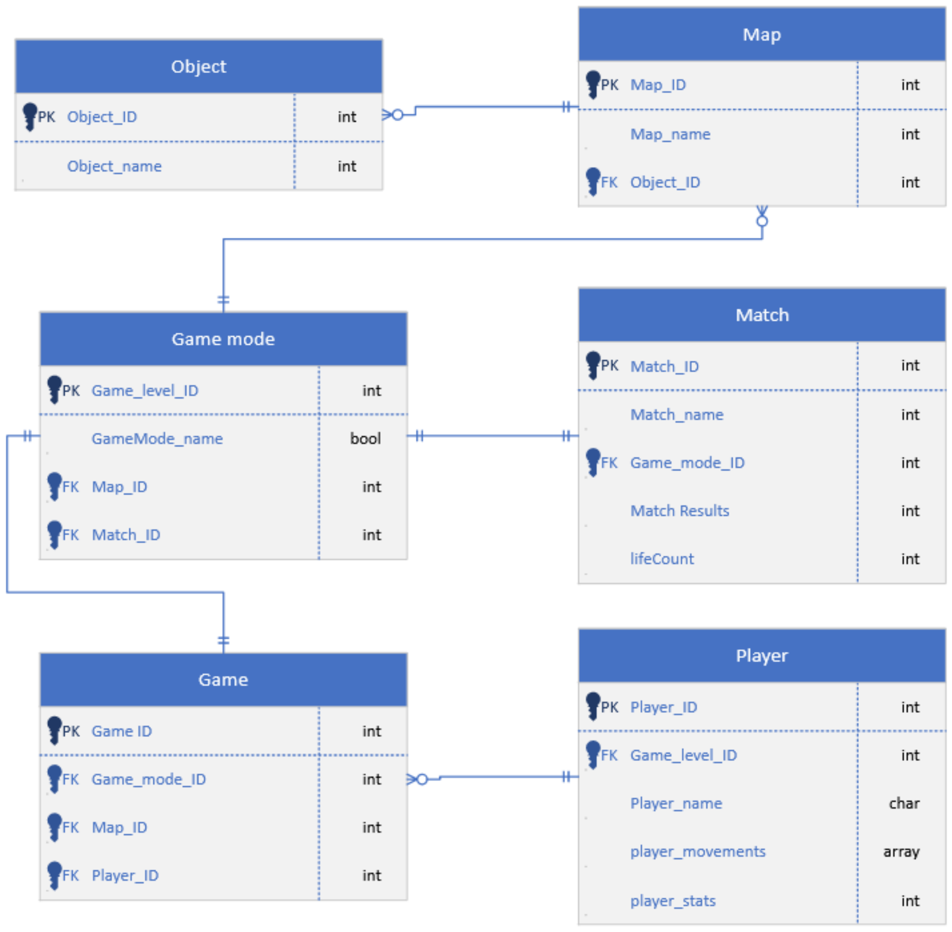
10.2 Major Requirements

1. The player can start the game and choose a level to play.
2. The player can move the on-screen character using the computer's peripherals.
3. The player is able to complete the challenges presented to them in the form of enemies, pitfalls, obstacles, etc.
4. The player can reach the end of a level by clearing a set of conditions.
5. The player, after clearing all levels, can exit the game and return to their desktop.

10.3 Design Constraints and Limitations

1. One of the more significant constraints is that to implement more refined movement mechanics in the game, we will need to design and build a custom movement handler, which includes the use of many external libraries. Consistent and satisfying movement is essential to a platformer game, and we will need to consider this challenge while designing the movement system and the levels themselves.
2. Another limitation is that the Unity engine does not have a built-in designer to create game menus with. The player will need to interact with a series of menus to start and finish playing, and as such, we will need to build these menus ourselves with the use of external tools.
3. Lastly, we will need to write our game's control scripts with extensibility in mind. Players should be able to set their own controls for the game so that the experience is comfortable, easy, and personalized. This means that we must be very careful not to hard-code any values into the scripts, even more so than would already be expected as a "best practice".

<u>11. Data Design (Entity Relationship Diagram)</u>



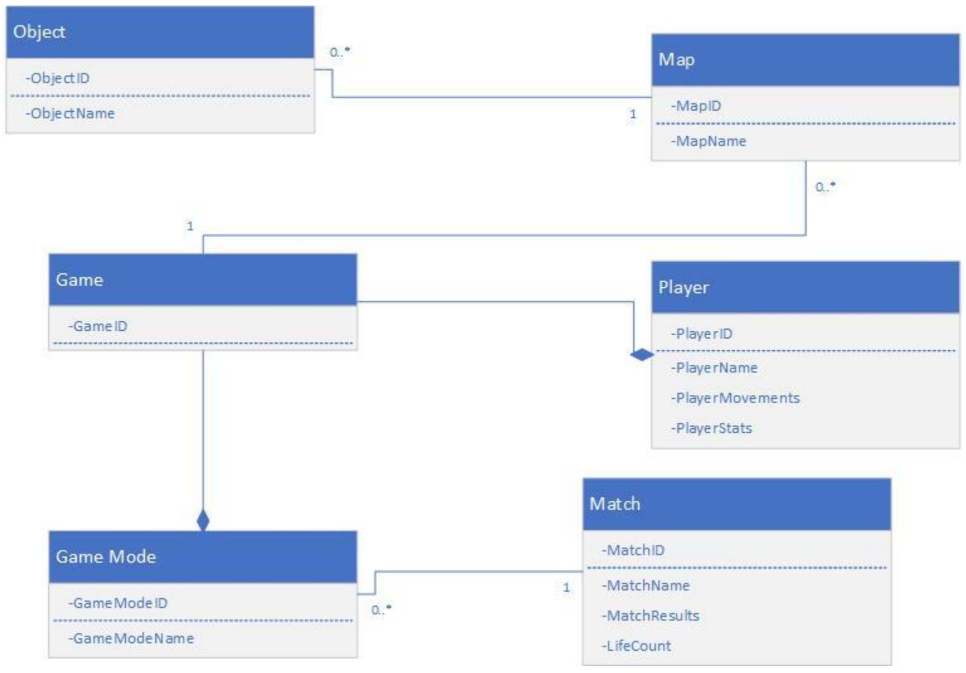
11.1 Data Objects

The ER diagram is a graphical representation of how different entities or objects in a project are related to each other. ER diagrams are most used to design or debug relational databases in software engineering, business information systems, education, and research. Also known as ERD or ER models, they use a defined set of symbols, such as rectangles, diamonds, ovals, and connecting lines, to describe the interconnectedness of entities, relationships, and their attributes. The “crow’s foot” notation in Microsoft Visio was used for this project.

11.2 Relationships

For one-to-one relationship like the relation between game and game mode the data is shared only once whereas for one-to-many relationship like the one between game and map, it indicates that each game can have multiple maps. In this game we are planning to have a single game with different maps. Also, each map can have different objects in a level of the game hence there is a one-to-many relationship between maps and objects.

12. Architectural Design (Modules/Class Diagram)



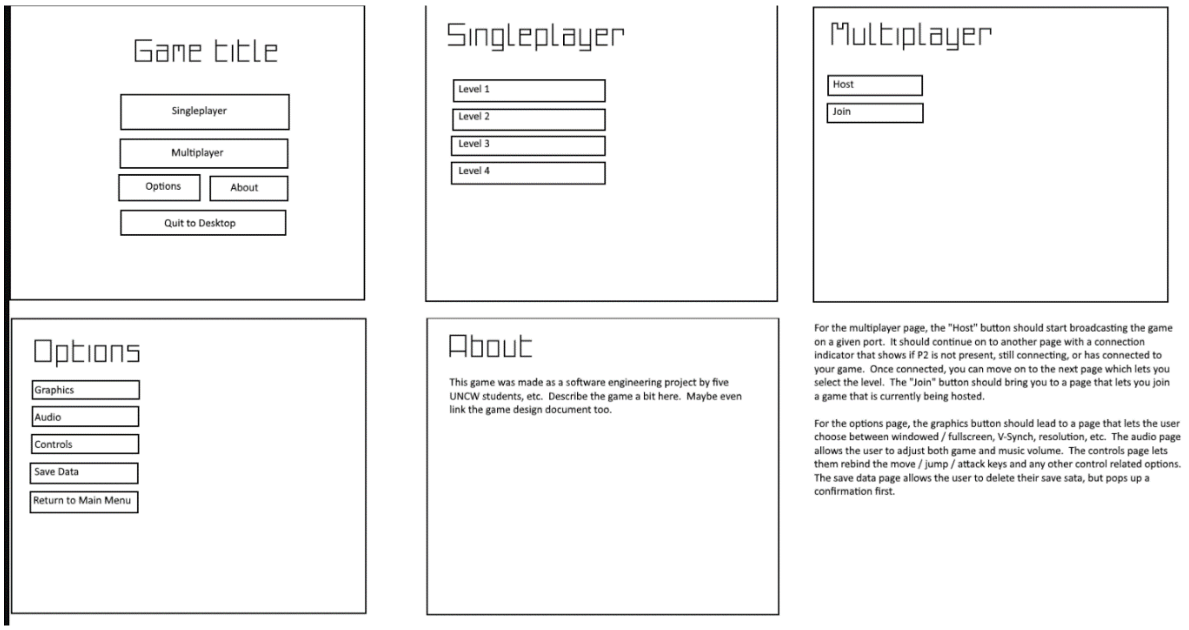
A class diagram in the Unified Modeling Language (UML) shows the connections and dependencies between classes in the source code. A class, which in this case refers to a particular item in a program or the chunk of code that corresponds to that thing, defines the methods and variables in an object.

13. Interface Design

The technique that designers use to create user interfaces in software or electronic devices with a focus on aesthetics or style is known as user interface (UI) design. Designers strive to produce user-friendly and enjoyable interfaces. Graphical user interfaces and other types, such as voice-controlled interfaces, are referred to as UI design. Character design and animation are typically the first aspects of video games that come to mind. Simply expressed, the goal of game interface design is to produce visual signals that guide players through the planned path of action in a particular video game. We are designing a non-diegetic user interface for this project.

13.1 Graphical User Interface Design

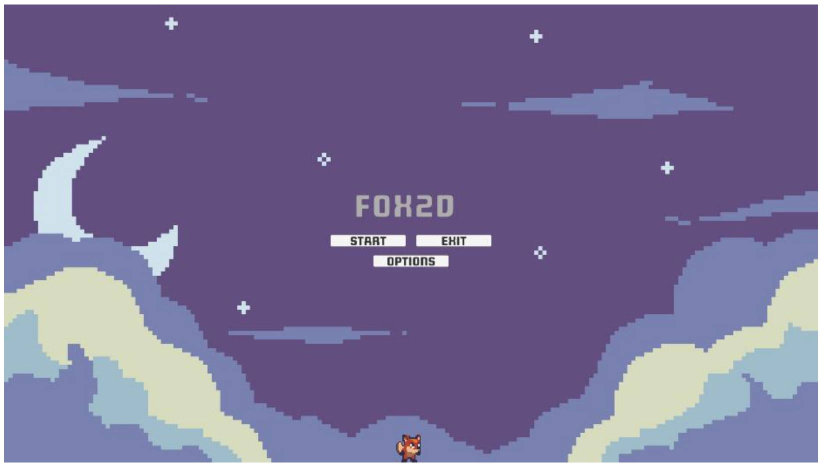
The images provided below was the initial design for the 2D game. This was based on the PC prototype. The design elements include graphics, control layout, audio levels, and save management. The logo, character, and icons were added to the game during development. This can be seen on the next page, which includes the final designs for the UI.



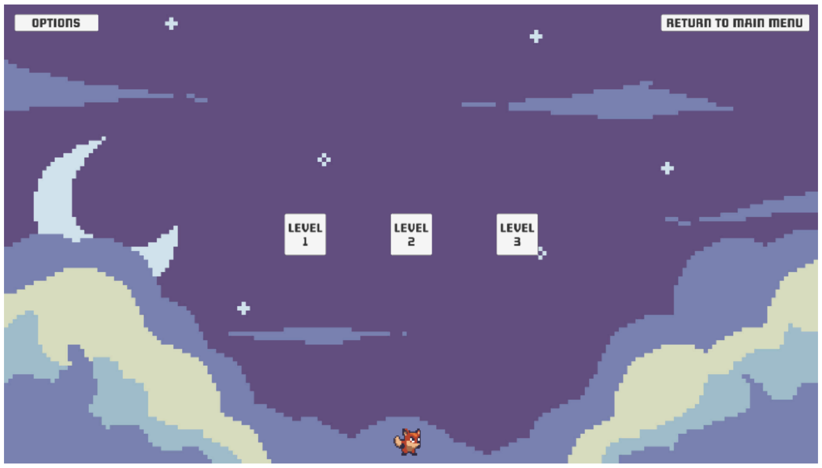
This game was made as a software engineering project by five UNCW students, etc. Describe the game a bit here. Maybe even link the game design document too.

For the multiplayer page, the "Host" button should start broadcasting the game on a given port. It should continue on to another page with a connection indicator that shows if P2 is not present, still connecting, or has connected to your game. Once connected, you can move on to the next page which lets you select the level. The "Join" button should bring you to a page that lets you join a game that is currently being hosted.

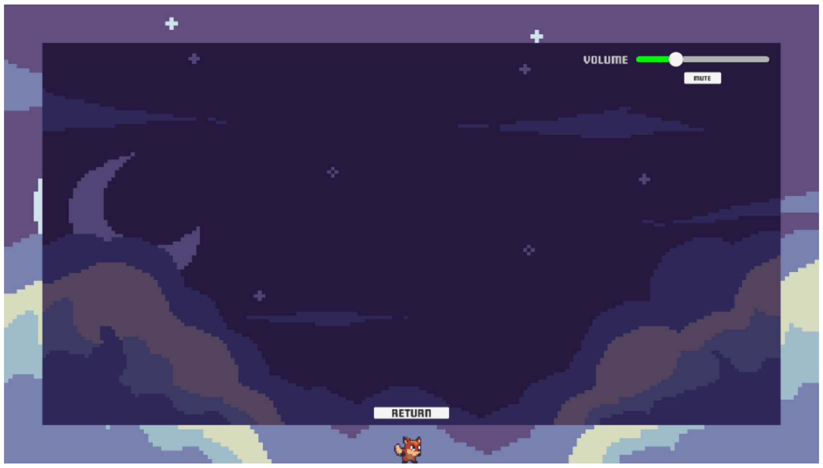
For the options page, the graphics button should lead to a page that lets the user choose between windowed / fullscreen, V-Synch, resolution, etc. The audio page allows the user to adjust both game and music volume. The controls page lets them rebind the move / jump / attack keys and any other control related options. The save data page allows the user to delete their save sata, but pops up a confirmation first.



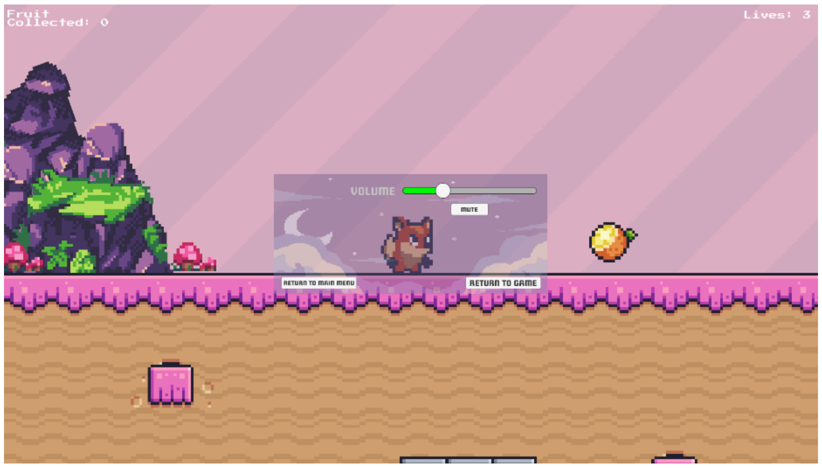
Main title screen



Level selection screen



Options screen



In-game pause screen

13.2 Graphical User Interface Specification

1. ***Start button:*** Shown on the main title screen. Takes the user to the level selection screen.
2. ***Options button***: Displays the options menu. The audio slider allows the user to adjust the game volume, with an additional button to fully mute it. The return button takes the player back to the main title screen.
3. ***Level 1/2/3 buttons:*** Begins the selected level.
4. ***Exit button:*** Used when the player wants to quit the game and return to the desktop.

14. Test Plan

14.1 What to test?

Testing entails methodically play-testing the game, comparing how it performed to the designer's aims, finding issues, and making suggestions for changes. In testing phase, we check for software bugs, from complete crashes to minor glitches in the program. Also, testers serve as the game's initial audience, providing feedback on its usability and pointing out any areas that may be improved.

14.2 When to test?

In video game testing, there are two main types of game testing, “in-progress” testing and “in-situ” testing. “In-progress” testing should be conducted throughout the programming stage while the development of the game is still underway. “In-situ” testing should be conducted once the game is deemed complete and has been published in its final form.

14.2 How to test?

Testers work in teams, sometimes playing together in the case of a multi-player game. Testing involves repeatedly playing a game, testing all of its components. This even includes trying incomplete “development builds”, which are frequently lacking critical features. Testers must continue testing long after the fun of playing the game has faded, even though the work can be monotonous and repetitive at times.

15. Functional Testing

15.1 Test data design technique

This involves finding bugs and other game issues that can impair the user experience is the goal of this testing mode. This testing includes assessing whether the application is performing in accordance with the requirements and if it falls within the category of complicated black-box testing methods. This takes longer to complete, as testers search for gameplay, graphical, and audio related flaws. Game testers watch carefully for any graphical issues with the interface like missing elements, incorrect colors, misalignment, or issues with animation and clippings. This process ensures that the user is able to navigate the game correctly. Via the use of this technique, problems such as freezing, crashing, and progression blockages can be avoided before they reach production/release.

15.2 Test cases:

Functional test cases	Expected results
Complete all levels.	Player must be able to reach end of game
Check for object placement	All levels should have every object that was implemented during development of the map
Check for level-specific stats	Player and enemy health points should be dependent upon the current level
Check for player health amount	Player health should be visible at all times and should decrease when damage is taken
Check for character movements	Character should turn left/right as expected
Check menu options functionality	Menu options should display volume, control, screen settings
Check pause/resume functionality	Player should be able to pause and resume game at all times

16. Performance Testing

16.1 Test data design technique(s)

For the purpose of creating a consistent and reliable game, it is crucial to assess the performance of the game on many criteria, such as scalability, stability, speed, and responsiveness. Low system resource usage and long battery life should be achieved, and game responses should remain constant under different types of workloads. Performance was tested on two different systems, each with very different specifications. The results of these tests are shown on the next page.

16.2 Test cases:

Performance Test Cases	Expected results
Processor usage	Usage should stay within reasonable levels
Graphics card usage	Usage should stay within reasonable levels and should be somewhat balanced when the game is running on multi-card systems
Memory usage	Memory usage should not exceed the predicted usage amounts
Disk usage	Disk usage should only ever be noticeable when the game is loading for the first time or when a level is being loaded into view
Load times	Game startup time should be consistent with the amount of content that needs to be loaded in at every startup

System 1 Performance

System 1 Specifications:

Operating System: Windows Server 2019

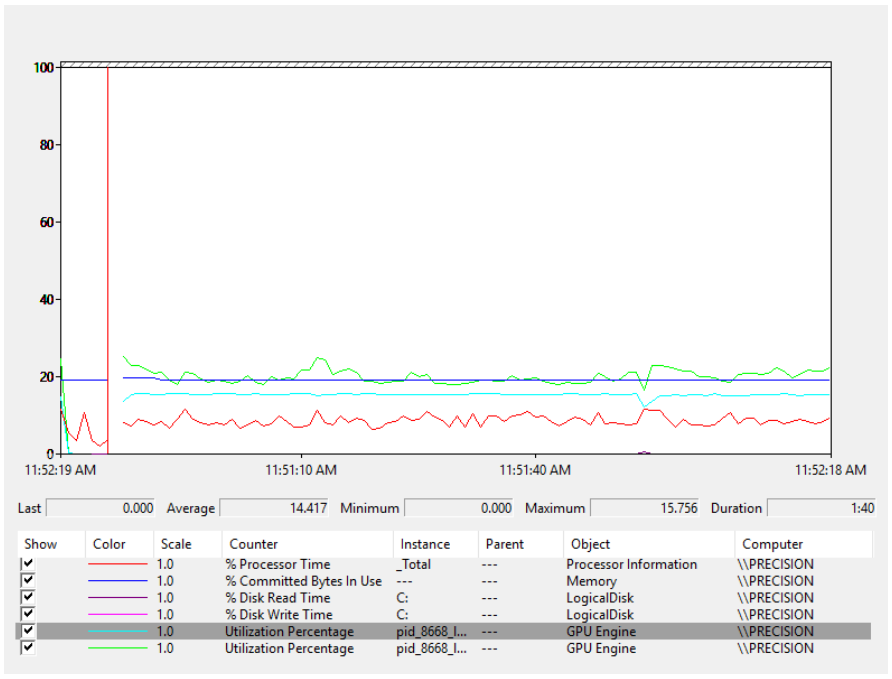
Processor: Intel Xeon E3-1505M v6 @ 3.00GHz, 4 Core(s)

Memory: 16GB DDR4 ECC

GPU0: Intel HD Graphics P630

GPU1: NVIDIA Quadro M1200

Disk: Kingston M.2 NVME SSD SNVS1000G



Last 0.000 Average 14.417 Minimum 0.000 Maximum 15.756 Duration 1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	—	1.0	% Processor Time	_Total	---	Processor Information	\\PRECISION
✓	—	1.0	% Committed Bytes In Use	---	---	Memory	\\PRECISION
✓	—	1.0	% Disk Read Time	C:	---	LogicalDisk	\\PRECISION
✓	—	1.0	% Disk Write Time	C:	---	LogicalDisk	\\PRECISION
✓	—	1.0	Utilization Percentage	pid_8668_l...	---	GPU Engine	\\PRECISION
✓	—	1.0	Utilization Percentage	pid_8668_l...	---	GPU Engine	\\PRECISION

System 2 Performance

System 2 Specifications:

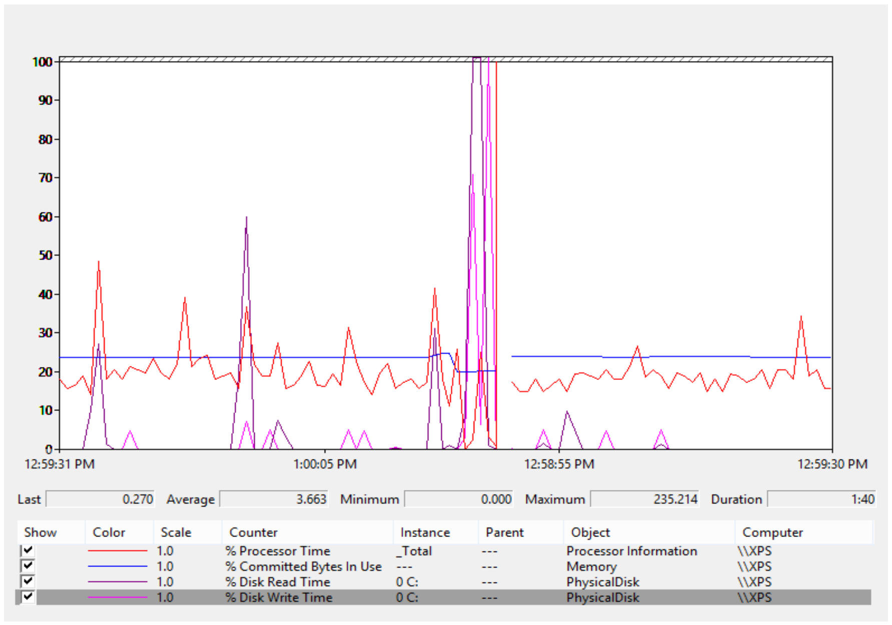
Operating System: Windows 8

Processor: Intel Pentium 4 @ 3.80GHz, 1 Core(s)

Memory: 4GB DDR2

GPU: ATI Radeon HD 7670

Disk: Seagate Hybrid HDD/SSD ST1000DX002



Last 0.270 Average 3.663 Minimum 0.000 Maximum 235.214 Duration 1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	—	1.0	% Processor Time	_Total	---	Processor Information	\\XPS
✓	—	1.0	% Committed Bytes In Use	---	---	Memory	\\XPS
✓	—	1.0	% Disk Read Time	0 C:	---	PhysicalDisk	\\XPS
✓	—	1.0	% Disk Write Time	0 C:	---	PhysicalDisk	\\XPS

*Note that the spike shown in this graph was captured when the program closed. The program seems to have a large overhead on older systems, but this does not affect runtime performance.

<u>Graphed Resources:</u>

Red: Processor Usage

Dark Blue: Memory Usage

Light Blue: Graphics Card 0 Usage

Light Green: Graphics Card 1 Usage

Purple: Disk Access Read

Pink: Disk Access Write

17. Compatibility Testing

17.1 Test data design technique(s)

This type of testing helps determine whether a game is working properly without regard to the hardware, software setup, and visuals used to create the device. This is one of the most crucial tests to determine whether a video game program operates correctly and does not degrade the user experience. Part of the compatibility testing was also handled during performance testing: By attempting to run the game on two different hardware configurations for the sake od performance differences, we have also tested the game's compatibility on two different computers. Each has vastly different hardware and software configurations, and are nearly 20 years apart in age.

17.2 Test cases

Performance Test Cases	**Expected results**
Check Windows OS versions	Game runs on Windows 7, 8, 8,1, 10, and 11.
Check MacOS versions	Game runs on MacOS 10.13 through 13.0.1.
Check supported screen sizes	Game should run at 1920x1080, 1366x768, and 4096x2160 resolutions
Check supported graphics types	Game should run on integrated and dedicated graphics solutions
Check form factor support	Laptop/desktop functionality does not impact game compatibility

18. Integration Testing

18.1 Integration strategies

There are multiple types of integration strategies that can be used for 2D game development: Top-down, Bottom-up, and Functional. Our primary way of integration testing will be bottom-up integration, as well as some functional integration testing. This will help to ensure that crucial game mechanics are working before the team continues development.

18.2 Test cases

Check sound effects	**Expected Results**
Increase or decrease volume	User should be able to adjust volume
Verify if sound effects are in sync with action	Player should not see any sound changes while playing game
ON/OFF device sound (native sound) and check	Players PC sound must be in sync with game
Check if music changes are as developed	Music must be same as developer expected (different music for different actions or levels)

User Interface	**Expected Results**
Check for animation, movement of character, graphics (all gestures) etc	Tester should be able to see changes in character movements with controls
There should not be any clipping	Tester should not find incomplete objects while playing
Test whether one object overlaps with another	Tester should not find any overlapping objects in all the levels
Character should not move out of the screen/specified area	While playing character should always be visible
Font displayed (color, size etc)	All the fonts, and color while playing game should be same as developed
Check other objects too (ex -if it's a car race- you need to look at road, people, other objects like buildings etc)	All the background objects should not collide with player or mask player
Check foe screen navigation	Screen change should be proper like menu screen to play or quit or pause
Check start and end of the game are error free	There should be screen changes from start and end of games while clicking different buttons

19. Regression Testing

After testing all the above four types of testing which are Functional, Performance, compatibility, and integration the errors will be reported to developer. Regression testing is a process where tester will test all the test cases gain to ensure that the updates in the game should not affect the old functionalities.

20. Test Evaluation

20.1 Test cases and actual test results

Shown below is the table that consists of whether or not the initial round of testing returned the expected set of results. Not that unless mentioned otherwise at the end of the paper, all of these issues were fixed in full.

Functional test cases	**Expected results**	**Actual Results**
play till last level	Player must be able to play all levels	Same as Expected
get the cheat codes from development team and check all the levels	each level should have all the objects that were implemented during development in the map	Same as Expected
Check for the features that will be unlocked level-wise	frog, health points must be unlocked in different levels	Same as Expected
Check for health score	health score should be visible	Same as Expected
Check the score hike when level gets increased	health must increment	Same as Expected
Check for game character movements	Character should turn to left or right as per control changes	Character not turning while jump
Menu options	Menu options should display volume, control, screen settings	Same as Expected
Check pause/resume functionality	Player should be able to pause and resume instead of restart	Unable to pause
Performance Testcases	**Expected Results**	**Actual Results**
Test game in both high and low performance systems	Player should be able to play in systems which have both high and low performances	Same as Expected
Check if that any action is not taking considerable time, game flow should be fast	Check if any glitches or crashes while playing game	Same as Expected
Compatibility testcases	**Expected Results**	**Actual Results**
Check in supported screen sizes and OS versions (basically depend upon requirement)	Load and play game in both windows and MAC	Same as Expected

Sometimes we need to check as per OS guidelines	Play game in different windows systems	Same as Expected
Check sound effects	**Expected Results**	**Actual Results**
Increase or decrease volume	User should be able to adjust volume	Unable to adjust volume
Verify if sound effects are in sync with action	Player should be able to check sounds while jumping or defeating objects	Music restarts when player dies
ON/OFF device sound (native sound) and check	Players PC sound must be in sync with game	Same as Expected
Check if music changes are as developed	Music must be same as developer expected (different music for different actions or levels)	Same as Expected
User Interface	**Expected Results**	**Actual Results**
Check for animation, movement of character, graphics (all gestures) etc	Tester should be able to see changes in character movements with controls	Same as Expected
There should not be any clipping	Tester should not find incomplete objects while playing	Same as Expected
Test whether one object overlaps with another	Tester should not find any overlapping objects in all the levels	Some objects were in front of character
Character should not move out of the screen/specified area	While playing character should always be visible	Same as Expected
Font displayed (color, size etc)	All the fonts, and color while playing game should be same as developed	Same as Expected
Check other objects too (ex -if it's a car race- you need to look at road, people, other objects like buildings etc)	All the background objects should not collide with player or mask player	Same as Expected
Check foe screen navigation	Screen change should be proper like menu screen to play or quit or pause	Menu/pause screens did not scale
Check start and end of the game are error free	There should be screen changes from start and end of games while clicking different buttons	End screen buttons don't work

20.2 Structural coverage measurement

Structural code coverage identifies the portions of the source code that are used by the application under test, software testing is complete. This offers a practical technique to guarantee that software is not made available with untested code. This game project had source code of player movements, camera controller, player life, menu, and end conditions. All the above source code components are tested in functional and integration testing.

20.3 Bugs that were detected and fixed

Based on above test cases and actual results below are the Errors detected.

a. High volume could not be adjusted.
b. Some objects were placed in front of the character instead of in the background.
c. Pause menu and pause functionality not working.
d. Character did not change direction (left/right) when jumping.
e. Music would restart when the player dies, or the level reloads.
f. Menu/Pause screens did not scale correctly in the application.
g. Buttons on the end screen don’t look and work as expected.

One bug was not fixed: After leaving pause menu player is getting stuck at pause state. User must click “Esc” to proceed further.

20.4 Summary of testing experience

The testing of this program included manual test cases. All of the testing was performed by loading, building, and compiling the game on multiple personal computers. The testing encompassed the following five types: Functional, performance, compatibility, integration, and regression. Based on the results of the initial run, the tester then received an updated version of the software with many of the bugs fixed, and the game was tested again. During the second set of tests, regression testing was performed. The program still contains two issues that were not fixed, as they were not critical, but would take a disproportionate amount of time to resolve. One of these issues is that the buttons on the end screen do not appear consistent with the rest of the menus. The other issue is that after leaving the pause menu, the player remains in a paused state and must press “Esc” to continue moving.

21. Overview

21.1 Overall description of the implemented system

The 2D Platformer Game that was implemented was later named to FOX 2D based on group feedback. The game contains a fox character that progresses through 3 levels. The fox can move forward, jump, or crouch based upon player inputs. Each level has different terrain and a unique placement of objects. The player must make the fox through all of the levels and across all of the obstacles. The difficulty of the gameplay increases with each level via the use of obstacles placements and the frequency of enemies, represented as frogs. The fox can defeat the enemies by jumping on them. Background music is played throughout the levels, which changes after each level is concluded. Players can pause, start, or quit the game by clicking the buttons that appear on the screen. Players can increase or decrease the volume of the music as well.

These features were stretch goals, and were not implemented due to time constraints:

The player should be able to define their own key bindings in the options menu.

a. Would require the entire input handler to be reworked.
b. The options menu would need to be recreated.
c. Many other aspects of the game would need to be regression tested.

There should be a boss fight in the final level.

a. Would require the creation of an enemy health tracking system.
b. A new level clear conditions would need to be set.
c. New sprites and animations would need to be added.

Being able to drop down through platforms.

a. Would need to change existing prefabs to account for crouching on a platform.

These are future improvements and additions to the game that we would like to add:

- Add additional levels to the game
- Allow saving data to social platforms
- Add a multiplayer mode
- Add a final boss fight
- Include variety of enemy types
- Add more objects to collect
- Add lava/water obstacles.
- Allow player lives and collected fruit counts to accumulate throughout gameplay, not just the level.

21.2 Summary of programming experience

Throughout the course of the game's development, we continued to add different modules and handlers to the game as necessary. This code consists almost entirely of C# scripts, which have placed in the "scripts" directory, a subfolder within the project's root. Unity is able to compile the code within these script files and execute them both within the main window (for testing purposes) and in the form of the deployable build (for production). Scripts need to be assigned to objects and sprites within the build window in order to have an effect on the desired components of the game. Some examples of the scripts that were necessary to achieve the intended level of functionality are as follows: Audio management, camera control, enemy logic, finish conditions, fruit pickups, gem pickups, handlers for the main menu / level selection / pause menu, player health management, player movement control, control bindings, sprite rotation, sound effect tracking, and text display. Many of these scripts need to be bound to the correct asset/element in the editor window in order to produce the intended functionality.

22. Codebase

The codebase is currently hosted on the team's SharePoint server in a shared folder. Migration of the codebase to GitHub is planned in the near future. Please note that the provided source code requires that the user have the Unity Editor program installed on their computer in order to view or alter the code. For now, the game is available in three forms:

Source Code:

https://uncw4-my.sharepoint.com/:u:/g/personal/ip5880_uncw_edu/
ESiVyoBahsdIl6IkPhN2Z-kBvp-qUpA_1c3NMkUc55TfIw?e=0TifWf

[hyperlink]

Windows Executable:

https://uncw4-my.sharepoint.com/:u:/g/personal/ip5880_uncw_edu/
EVqNdt4qn9VJuTzCbnPBuosB7Ci2wr44yMKR8c_INlNekQ?e=7fIe2z

[hyperlink]

MacOS Application:

https://uncw4-my.sharepoint.com/:f:/g/personal/ip5880_uncw_edu/
EmtPh4MhpRFHuWZL-F-rR70BGKmmQ8BQuxlR0Ds4XI7ZWw?e=FMWouC

[hyperlink]

23. Appendix

Meeting 1:

2nd February 2023: 6:15 PM – 6:45 PM

5th February 2023: 10:00 PM – 10:30 PM

Estimated that the total project time is around 190 hours of work.

Weekly meeting through zoom, every Saturday evening. (Timings flexible according to the availability of team members.)

Using Discord and GitHub to update each other about individual work and online communication.

Research on the platform game, details of how the game is to be developed and the number of levels included in the game.

Meeting 2

26th February 2023: 2:00 PM – 4:30 PM

Experimenting with the Unity engine and verifying system requirements: Versions are compatible with everyone's systems.

Inspiration from AI generated sprites and how to implement other game assets into project.

Discussion of core game mechanics and what features to expect: This was required for the creation of the diagrams and to consider the scope of the game given our constraints.

Investigation of client-server connection establishment methods in order implement multiplayer functionality. We will attempt this part of the project at a later time in development.

Added missing components to our original outline, improved components according to feedback.

Meeting 3

16th March 2023: 7:00 – 9:30 PM

Revised table of contents and made improvements to document formatting.

Considered which diagrams were most significant, given the nature of this project.

Discussed the process of creating all design diagrams.

Added sections that were missing or incomplete in previous versions of the document.

Meeting 4

24th April 2023: 10:00 AM – 3:00 PM

This was the final meeting to finalize the game and assemble the presentation.

Conducted performance testing.

Fixed bugs found in various different tests.

Transferred information from documentation files into the presentation.

Updated all graphs and diagrams to reflect the current state of the project.