Contra Development

CourseID: CS 360 - 001

Milestone 4 and Date: Milestone - 12/3/21

Project Members: Ethan Moore, Aman Patel, Will Craddock

Client: Dr. Galloway

Contents

[Introduction 2](#_Toc89791086)

[Project Scope 5](#_Toc89791087)

[System Boundaries 6](#_Toc89791088)

[Nonfunctional requirements 7](#_Toc89791089)

[Functional Requirements 7](#_Toc89791090)

[Structural UML Diagrams 7](#_Toc89791091)

[Organizational approach 13](#_Toc89791095)

[Technical feasibility 14](#_Toc89791096)

[Schedule feasibility 15](#_Toc89791097)

[Progress visibility 16](#_Toc89791098)

[Risk analysis. 20](#_Toc89791099)

[Wireframe diagrams/Storyboards 21](#_Toc89791100)

[OO Design Patterns 23](#_Toc89791101)

[Structural UML Diagram 26](#_Toc89791105)

[Data Dictionary 28](#_Toc89791109)

[Security 28](#_Toc89791110)

[Software/System Performance 29](#_Toc89791112)

[Software Testing 37](#_Toc89791121)

[Software Testing Checklist 37](#_Toc89791122)

[Unit Testing 38](#_Toc89791123)

[Flow graphs 40](#_Toc89791124)

[Integration Testing 44](#_Toc89791125)

[System Testing 45](#_Toc89791126)

[Validation/Acceptance Testing 46](#_Toc89791127)

[Concluding Remarks/Future work 53](#_Toc89791128)

[Appendix 53](#_Toc89791129)

[Final Source Code with Comments 53](#_Toc89791130)

# Introduction

Contra is a run and gun game originally created for arcades and other publicly available places. Original publisher of the game was Konami, and the game was published on February 20, 1987. And in 1988 home version was released for Nintendo Entertainment System, along with various computer formats. Different regions were calling this game with different names, for example the United States was calling it as Contra, PAL region or European were calling it Gryzor or probotector. The Japanese were calling it Kontora. And after the first success in 1988 they published several sequels of Contra since the original game, but none was more popular like the original game.

Let’s talk about the different characters in the game, contra had a lot of characters from different monsters to game heroes and aliens and more. First two are the face of the game, Bill Rizer and second is Lance Bean, they are future Earth’s premier alien-skull-busting specialists. They first appeared in the arcade version of the game and people loved them so very much they have been the default characters ever since. Bill Rizer’s character design is meant to evoke actor Arnold Schwarzenegger. And his name is a combination of two famous actors Bill Paxton (Bill) and Paul Reiser (Rizer). They both appeared in the action Alien movie when the game was first launched. Bill Rizer has then appeared in Contra Return, Super Bomberman R, Contra Evolution, Konami Classics Vol.1, Contra ReBirth, Contra 4, Konami Classics Series: Arcade Hits, Neo Contra, Konami Collector’s Series: Castlevania & Contra and many others.

Second player character Lance Bean is meant to evoke actor Sylvester Stallone’s John Rambo character. And his name is the combination of two actors: Lance Henrikssen (Lance) and Michael Biehn (Bean). Both appeared in the Alien action movie. And he was appeared in the games like Contra Return, Super Bomberman R, Contra Evolution, Konami Classics Vol.1, Contra ReBirth, Contra 4, Konami Classics Series: Arcade Hits, Neo Contra, Konami Collector’s Series: Castlevania & Contra and may other

There was a third character called Lance Gryzor which was present only in the European version of the game and only appeared in the first version of the game. And this version of the character was the combination of both Lance and Bill.

Now we will talk about the popular monsters of the contra series. One of the most dangerous and famous of all was the Java monster also known as Red Falcon of Red Falcon army and one of the highest ranked members of the army too. He was the highest-ranking leader in the command of the first alien invasion attempt on Earth in 2633 A.D. He is recognized as two different monsters, his upper body and head are known as “Emperor Demon Dragon God Java”, While his heart is known as “Gomeramos King”. His appearance in the game is just like his background stories, Java vomits an endless supply of the Bundles(flying and persistent grub-like creatures that attack the player). The player is continuously given new bundles by the Java and the player cannot move to the next corridor until the Java is destroyed.

Once the player has destroyed the upper body and the head of the Java monster his heart remains functioning and is characterized for covering up an entire wall, pulsating rapidly as it provides life to the rest of the biomass. And this monster is known as “Gomeramos King”. And it is protected by the alien eggs surrounding it on both ground and ceiling level against potential threats, which release an unrelenting hoard of scorpion- like creatures. Java first appeared in Contra III: The Alien Wars, l gas mini-bosses that are fought throughout the final stage.

Another famous weapon/boss of the game is Dogra which is nothing but a tank that shoots cannons and has a spike in the front bumper. Dogra has appeared twice in the stage number 5 known as “Snow Field”. And as the stages are timed it is important to do more damage in less time. So, weapons like Spread Gun, Machine Gun or Rapid-Fire Laser are really helpful in successfully destroying this vehicle. Dogra changes the colors as it takes the damage from the player, and it proceeds to aim the cannon at the player’s current position. In different versions of the game the destruction of the Dogra is meant to be different. For example, in Contra: Evolution Dogra’s can be destroyed with a single stab from Sally’s katana.

Defense Wall is another famous monster of the. It appears in the very first stage of the game. This stage is called “Jungle” and Defense Wall is the final enemy to proceed to the next stage which is called “Base”. It looks like a tall and heavily armed defense mechanism that usually guards the entrance to an underground enemy base. And after its debut in the original version this wall has been used in every single contra game as a mini- boss or a boss in one of the earlier stages. Until Contra 4 the core remained passive during encounters, but after Contra 4 it was given the ability to project a powerful energy beam. Another Defense wall which is famous for four different power sources is known as “Garmakilma” which appeared at the end of stage 2 called “Base” guarding the stage 3 called “Waterfall”. Here is the table of the different monsters in the different stages of the contra game line.

|  |  |  |  |
| --- | --- | --- | --- |
| **Stage** | **Name** | **Boss** | **Music** |
| **1** | Jungle | Defense Wall | Dense Forest Battle |
| **2** | Base | Garmakilma | Labyrinth Fortress |
| **3** | Waterfall | Gromaides | Waterfall of Bloodshed |
| **4** | [Base 2](https://contra.fandom.com/wiki/Contra_Stage_4) | Godomuga | Labyrinth Fortress |
| **5** | Snow Field | Dogra ×2 Guldaf | Fortress in the Ice |
| **6** | Energy Zone | Gordea | Fortress in the Fire |
| **7** | Hangar | Final Gate | Dense Forest Battle |
| **8** | Alien's Lair | Emperor Demon Dragon God Java  [Gomeramos King](https://contra.fandom.com/wiki/Gomeramos_King) | Horrible Heartbeat |

*Table listing Contra level numbers, names, bosses, and music.*

Contra is one of the most famous run and gun games of its time. And features like selecting your weapons and characters that evoke actors like Arnold Schwarzenegger and Sylvester Stallone were adding more and more popularity among adults and kids playing games those days. Even today there are a lot of contra fans all around the world playing the desktop version and our team is one of those. Now as we discussed earlier the game has many different weapons, characters, stages and monsters, but in our version, we will be not adding all of those monsters and characters due to the shortage of time and resources, in next few pages we will talk about exactly what stages and monsters we will offer in our version.

# Project Scope

As we mentioned earlier, original Contra was offered as an arcade, then there was the Nintendo version and recently there has been desktop and even smartphone versions of the game, but we will just be focusing on the desktop version as discussed with our client Dr. Galloway. And also we will be creating the version of Contra with fewer stages but they will be more interesting and take longer to finish as we intend to have more than one major monster and bosses on one stage. That way we have fewer stages but the same amount of entertainment for the players.

We will be implementing three stages which will be inspired from stages 1(Jungle), 5(Snow Field), and 8(Alien's Lair). But our goal is to add all the monsters and bosses we could to these three stages and also give as much ammunition and weapons we can get to our characters. We will add both main characters to the game. And our game will be able to connect to the gamepad to keep the nostalgia of the original game while also implementing remapping support.

Hardware

The hardware we expect our version of Contra to run on is a Desktop or Laptop. Unity’s requirements for the operating system are either Windows 7 or 10. The CPU requirements include x86/x64 architecture with SSE2 support. The graphics API DX 10, DX 11, DX12 capable. Some additional prerequisites are the drivers like visual studio 2015 with the C+ Tools component and Windows 10 SDK. The system resources required to play our remake shouldn’t be intense. The CPU required should be within the last decade. RAM usage should not be greater than 50 megabytes since our game has a more limited scope than the original. The space required to download our remake should be less than a gigabyte since we are not recreating the full game. We are also using the original sprites which are fairly lightweight and bloat free. Our development machines are running modern graphics and processing units. An extra piece of hardware the user may opt for is a NES gamepad. We will develop our game with the Mekhela 5.8 foot classic us wired controller in use.

Software

Logo, company name

Description automatically generated

Front-end frameworks: To be able to create our game and see our behaviors and assets we will use the Unity game engine. The front-end framework’s version we are using is Unity version 2020.3.17f1 LTS. We went with this unity version because it was recommended, and the 2021 version could be too new while the older version could be more stable. The scripting language we will use in Unity is C# version 8.0 that uses the Roslyn C# compiler.

Back-end frameworks: To retrieve the saved data from the user we will utilize a back-end framework. The back-end framework we will use for storing the high score, current level, current players items is php my admin and MySQL. The database will be locally hosted on the user's computer on an open non conflicting arbitrary port. We won’t use a public facing server to access the database but a locally hosted one for save game data.

# System Boundaries

Physical

* The Physical boundaries for this system will be as follows:
  + The Game will be run on Unity which will be run on windows 10
  + We will be using SQLite which will be implemented through the save feature of the game

Logical

* The Logical boundaries for this system will be as follows:
  + The game will be interacting with the Database for saving user info and saving the state of the the game while in a level
  + The player will interact with the system through a computer's keyboard or a NES controller with an USB plug in.

# Nonfunctional requirements

1. The project should be implemented using the Unity game engine and development environment.
2. The project team should create a complete replica of the game.
3. The project can use pre-developed images, artwork, audio, and other related media.
4. All project source code must be developed by the CS 360 project team.
5. The project must use a database.
6. Performance metrics should be gathered and optimized.
7. Security metrics should be gathered and optimized.
8. User interface metrics should be gathered and optimized.

# Functional Requirements

1. Controlled by the keyboard or gamepad.
2. Multiple players to create separate accounts.
3. Login system for users
4. Save the state/progress of all players and
5. Allow players to continue once they've logged into their account.
6. Splash screen when launched which shows register and login prompt.

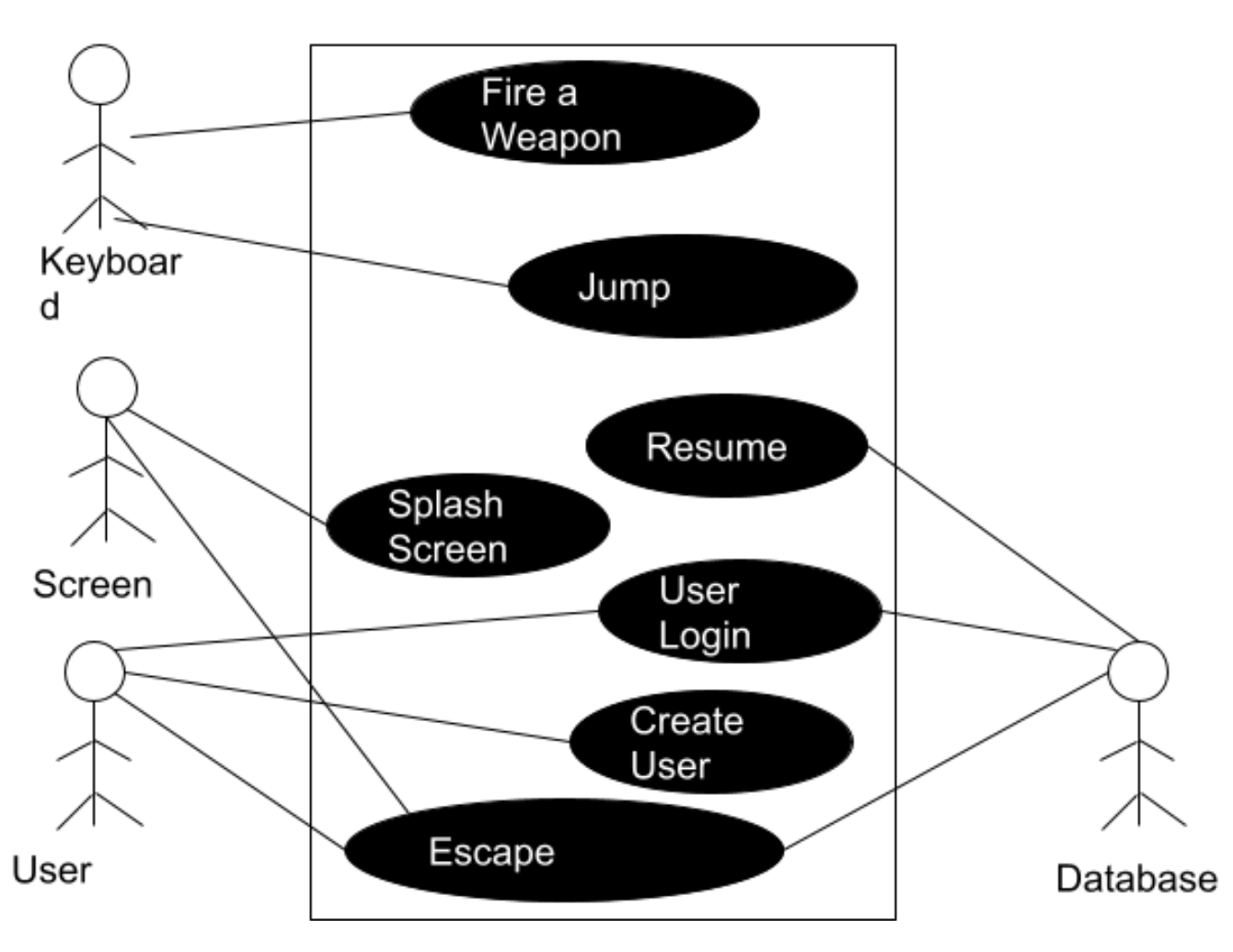
# Structural UML Diagrams

## Use cases diagram

Use case diagrams provide us with a better understanding of how system actors are intersecting with each other. On top of that they also give us a better understanding of system boundaries. Use case diagram consists of three elements. Actors, Use cases and System boundaries.

* Use cases: Actions that your system can perform by interacting with internal components and outside actors. In the diagram below we have chosen some of the many use cases possible from the game. Firing a weapon, admin/ user login, saving the progress, creating a new user etc.
* System boundaries: Defines the system in relation to external components. In our game there are going to be boundaries as discussed earlier.
* Actors: Individuals or external components that interact with your system. In our case these will be users, admins, database etc.

As shown below we have six use cases and five actors interacting with each other to work accordingly. Everything in the big rectangle is in the system boundary and the system is supposed to do when it's functional. According to the diagram, user uses keyboard to fire a weapon, Splash screen with the login prompt will be displayed on the screen , user will be able to log in, create new user save their progress, and admin will able to log into the administrator account.



Developed Use case Scenarios

Use case diagrams help us determine boundaries of the system and also get a better picture of how all of the stakeholders work together to make the system functional. But that is not enough to understand the system and why we have the functionality that we have. To tackle that problem, we will give examples of developed scenarios with the help of Use cases in the use case diagram and then trace requirements in the requirement traceability table to see if all the requirements are met.

Use case scenarios are just the broad explanation of use cases in the Use case diagrams. Every scenario has a primary scenario, actors, preconditions and a secondary scenario. Primary scenario is the main explanation of what's going on, preconditions are the requirements and secondary scenarios are the errors, exceptions or branches of the primary scenario. Actors can be the users, admin or database people or the external systems involved in the scenario.

We have two different scenarios here and the ID for both is UC1 and UC2, UC1 has three actors: Player1, Player2, and database. And the precondition is that player1 already exists, player1 starts the game, splash screen is displayed with the login prompt and high score is displayed on the top left corner once the player logs in the top score of the player is also displayed below the top score. Now the player starts playing the game using the controls on the keyboard. Now another player wants to join the game, but that user does not have the account, so he creates a new account and now both players start a game. Both after playing for a while stop the game and log out of the game and all the progress is saved under their respective accounts. And we have one secondary scenario here that is if a user logs in with the wrong password or username it gives a login error.

|  |
| --- |
| Use Case: Contra Game |
| ID: UC1 |
| Actors: Player1, Player2, Database |
| Preconditions: Player1 exists |
| Primary Scenario:   1. Splash Screen is displayed with the prompt to Login. 2. Player logs in to the game 3. Top score is displayed 4. The player's top score and level the player is on. 5. Player starts playing the game using the keyboard 6. Another player joins the game. 7. Creates a new account. 8. And both players are playing a game. 9. Both users stop playing the game 10. Their progress is saved in the database 11. Both log out. |
| Secondary Scenario:  Error Logging in |

For the second use case, preconditions are Player1 and Player2 both exist, and actors for this scenario are Admin and Database. Here the admin is logged to his/her account and then deletes a user which has been inactive for a while to free up space in the database. And same as earlier the secondary scenario is the invalid login information.

|  |
| --- |
| Use Case: Contra Game |
| ID: UC2 |
| Actors: Admin, Database |
| Preconditions: Player1 and Player2 Exists |
| Primary Scenario:   1. Administrator login to his account 2. Sees what all the players progress 3. Sees Player 2 is not active for a while 4. Deletes Player 2 5. Logs out of the account |
| Secondary Scenario:  Error Logging in  Invalid User |

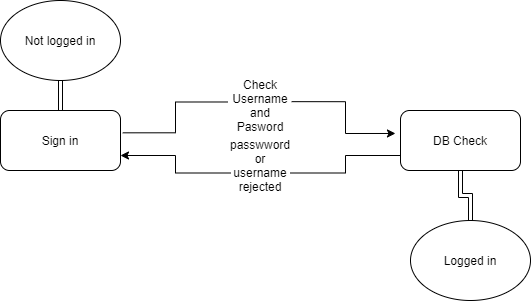
Requirement Tracing

Now we will map the requirement traceability table. Here the requirements are referred to Functional requirements discussed earlier. So according to the table Scenario 1 is using requirements like creating the user, controlling the character using the keyboard, saving the progress, user log in, splash screen, and multiple players. And Scenario 2 is using the admin login requirement.

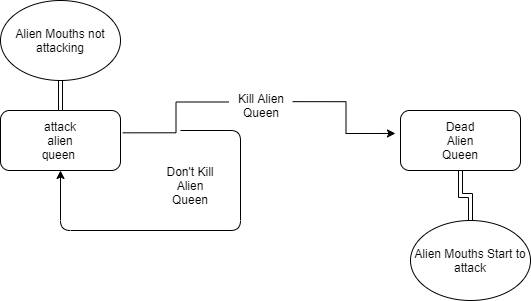
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Requirements | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| UC1 | ✔ | ✔ | ✔ |  | ✔ | ✔ | ✔ |
| UC2 |  |  |  | ✔ |  |  |  |

## State Diagram

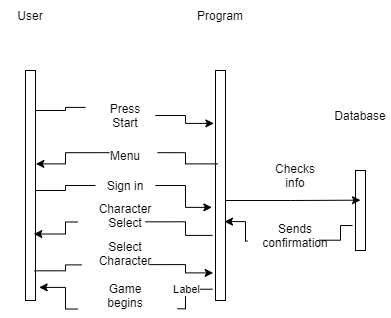
User is not logged in but the state changes to sign in when the database check is successful.



In the final boss the state of the alien queen is determined by the player killing it. If the queen dies, then the alien mouth attacks the player.



## Sequence Diagram



Our sequence diagram shows the interaction between the use and program. Some interactions include starting the game by pressing the start key, signing in, or seeing the menu generated by the program. The program interacts with the database by checking information and sending confirmation. The user does not interact directly with the database but uses the program as a proxy.

# Organizational approach

How/when did the group meet: Group has decided to meet twice a week that is Tuesday over the discord call and Thursday in person to discuss the progress and task each group member has to perform. Group also meets with the client every week on Friday at 4:00PM over the Zoom to share the progress of the project. Group has also decided to meet outside of the scheduled time if necessary. The organization and version control method we will use for our project is github. The group member hosting the repository is Ethan Moore or also known as the github user eldm-ethanmoore. The project is the repository Contra-Remake-CS360. This repository will contain our draft, weekly progress reports, and general project files like scripts and assets.

Sprint 3 overall team evaluation: Our team has three members and here is the background of all three members.

* Aman Patel: I am a CS major currently in my senior year, have a little experience in java coding and data structures. Am also familiar with the database functionality. Have worked in groups in the past so I will be an asset for this team moving forward.
* Ethan Moore: My major is also CS and I am in my sophomore year. I have experience with java, python, solidity, some javascript and C/C++. My main interest is in blockchain development and writing smart contracts. The most I have used the unity game engine for is to manipulate the player object via firing a ray at another gameobject to displace myself upward. I have worked on one group project in the past so this will be a new learning experience.
* Will Craddock: I am a CS major and I am a post baccalaureate student. I have experience with Java, C++, C# and HTML. The area that I am interested in is game design. I also have experience in making a game in unity as that was the focus of a group I had worked with in the past. My main focuses in that project were projectile mechanics and some inventory mechanics. I have worked with groups in the past over multiple subjects so there’s a lot I can bring to this team.

We have spent 5-6 hours a week each on implementing level 1. This time has been spent both on discord and in Raymond Cravens library planning and working on our draft and presentation. The days we have been working jointly are on Tuesdays and Thursdays.

# Technical feasibility

The technicality of our project includes unity version 2020.3.17f1 and C# version 8.0. It should be feasible to implement C# scripts since all group members are familiar with scripting and programming. We are also vaguely familiar with using Unity so learning and understanding both Unity’s development environment and the C# programming language shouldn’t be a risk in our project to remake the NES game Contra. An added guarantee of the project's feasibility using Unity is that we will all complete Unity tutorials and read documentation before writing source code. The database portion will utilize php my admin and my sql for the saved game data.

# Schedule feasibility

To get everything done in an orderly manner, and to see how much we are getting done our team will be using two tools. First is the gantt chart and another is the activity graph. Gantt chart will cover the whole project and will give the client and other stakeholders a better view of how the team is preparing and performing the tasks and what they are planning next. An activity graph will be how the team is adding those pieces together and how much time they are spending in doing that as a team. Let's first talk about the project and our estimated time to complete the project.

Duration

Our project is divided into four sprints. Each sprint will be about 3 to four weeks long. We will be following the Waterfall Software Process model to complete the project. It means we will complete task1 and then start on the next task.

* Our initiation of the project was on August 23rd when sprint one started and will end on September 12th. During this sprint we will be gathering all the requirements from the client and other sources like previous creation and similar works. Then our other task during this sprint will be team organization and feasibility study. We will familiarize ourselves with the Unity Game engine interface on which we will be creating the game. And also, during all this we will be keeping and updating the documentation so it will be easier for all stakeholders involved in the project to know what's going on and later after the project is finished for maintenance purposes.
* Second sprint will begin on September 13th and will end on October 10th. During the sprint 2 we will design the different aspects of the game and again keep the complete documentation of what we are doing.
* Third sprint will begin on October 11th and will end on November 7th. During the sprint 3 we will be implementing the actual game using the unity game engine developer and again keep the complete documentation of what we are coding.
* Final sprint will last for the remainder of the semester and will consist of testing of our product and changing what needs to be changed in the game and keeping all the record of how we achieved desired results, and what mistakes we made along the way.

Current Sprint

Time for each task:

* Creating a testing checklist took around 2 hours
* Code reviews took around 10 hours
* Creating flow graphs and calculating cyclomatic complexity took around 5 hours
* Integration testing took around 6 hours
* System testing took around 5 hours
* Validation testing took around 7 hours
* Acceptance testing took 3 hours
* Activity graph and Gantt chart took 1 hour altogether to complete.

During this sprint we will mostly work on different tests and test our project in different ways. Basically, test our game.

# Progress visibility

In this section we will discuss how we keep the visibility of our project to the stakeholders involved in the project. We are keeping the complete documentation of what we are doing along with the research materials and older versions of the documentations and diagrams. We share our work after every sprint by giving a complete presentation and demos of our game to our client, so they feel informed. Along with the presentation at the end of each sprint we also conduct client team meetings every week to give clients a snapshot of what we did in the current week and what our goal is for the week coming ahead. Along with those we use two other powerful tools to help our client feel more informed and those are Gantt charts and Activity Graphs.

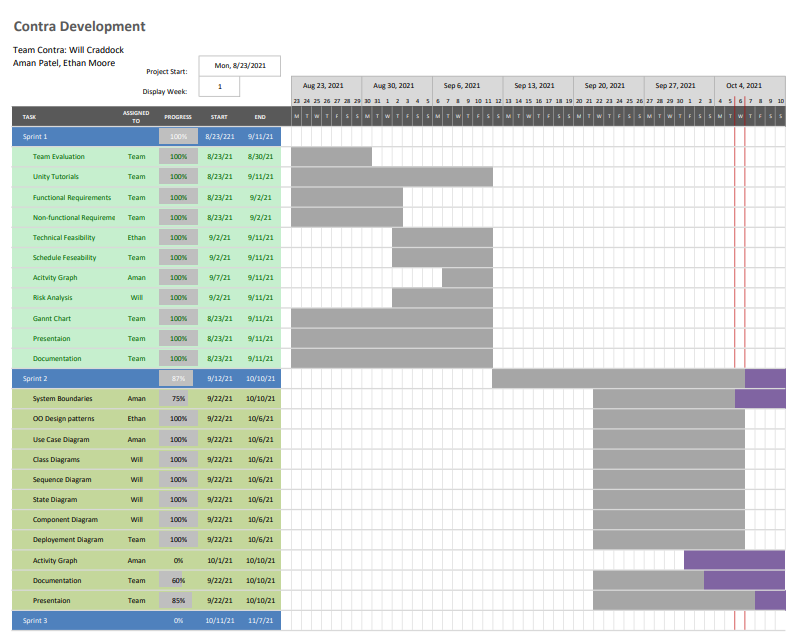
Gantt charts give a complete detail of what the team did throughout the sprint. We have included three versions of the last three sprint’s Gantt charts. In the Gantt chart you can find the different tasks and what day the team started working on those tasks and when did they finally complete it. Along with these photos of the Gantt chart we also give our client the excel version of the Gantt chart. So, they can better understand and understand what we are doing as a team.

Activities are a more detailed and specific view of the Gantt charts in our understanding. They are the more time oriented, and task manipulated graphs that show exactly how much the team spent on a specific task and how long it took them to complete that task and what other tasks were completed in order to complete the main task that was started. And just like Gantt charts we have provided all three versions of the activity graphs.

Chart

Description automatically generated

*Gantt chart version 1*

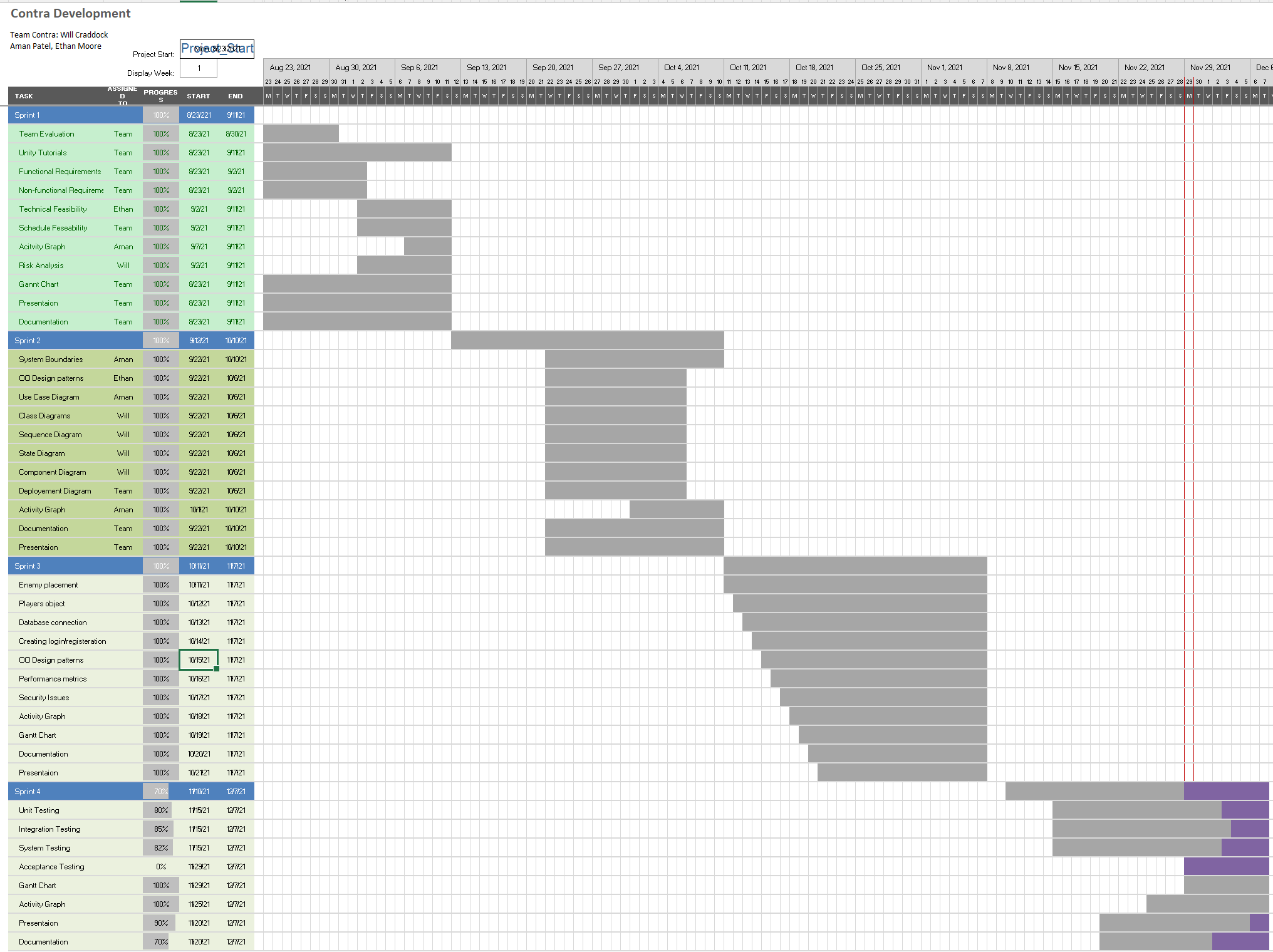


*Gantt chart version 2*

Chart

Description automatically generated

*Gantt chart version 3*

**

Activity Graph:

Diagram

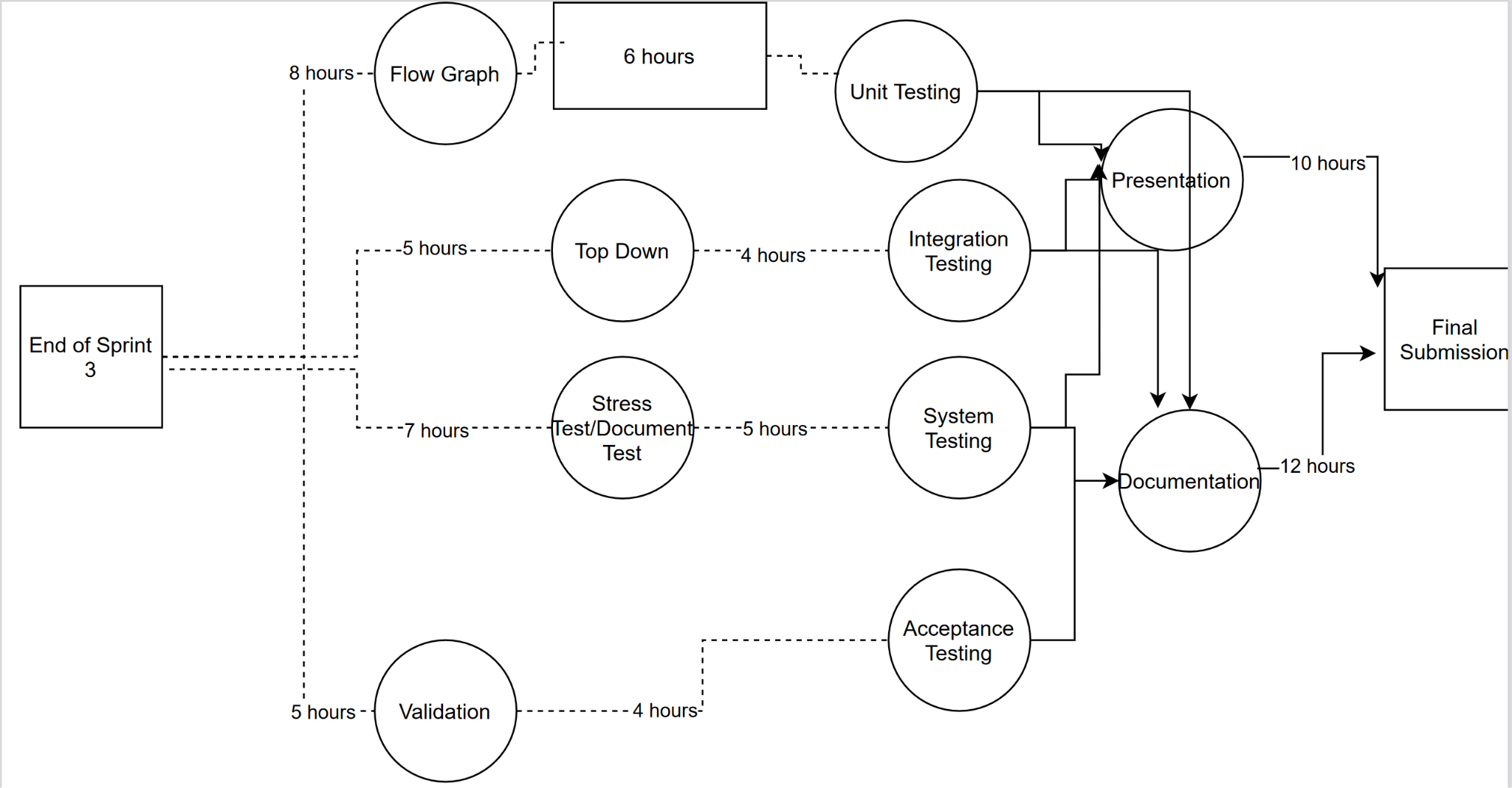
Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated



# Risk analysis.

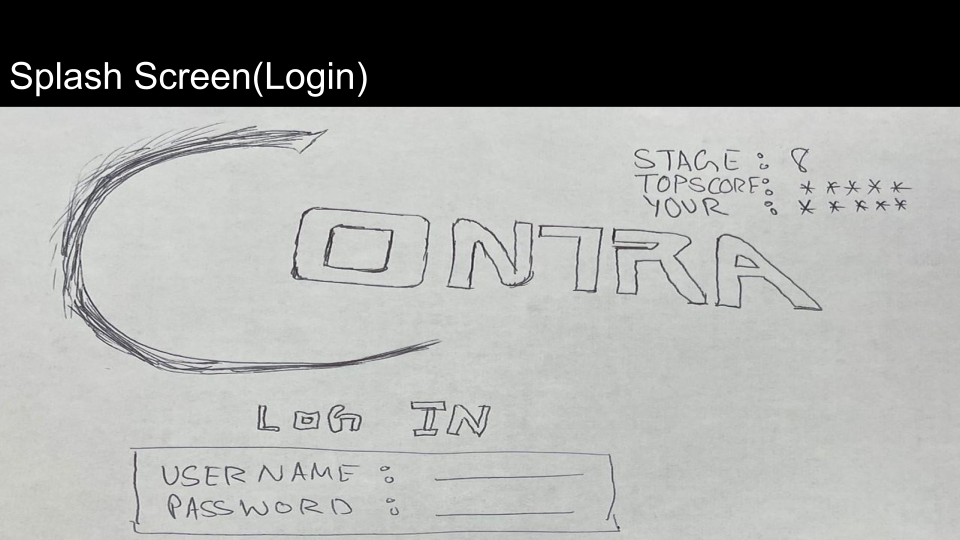
Risk analysis is something that most businesses take seriously. This has not changed in our group project. We have looked at this from different angles to ensure that we have a plan if something were to go wrong. We have looked into potential risks that could come up in the future and have provided some examples.

Potential risks

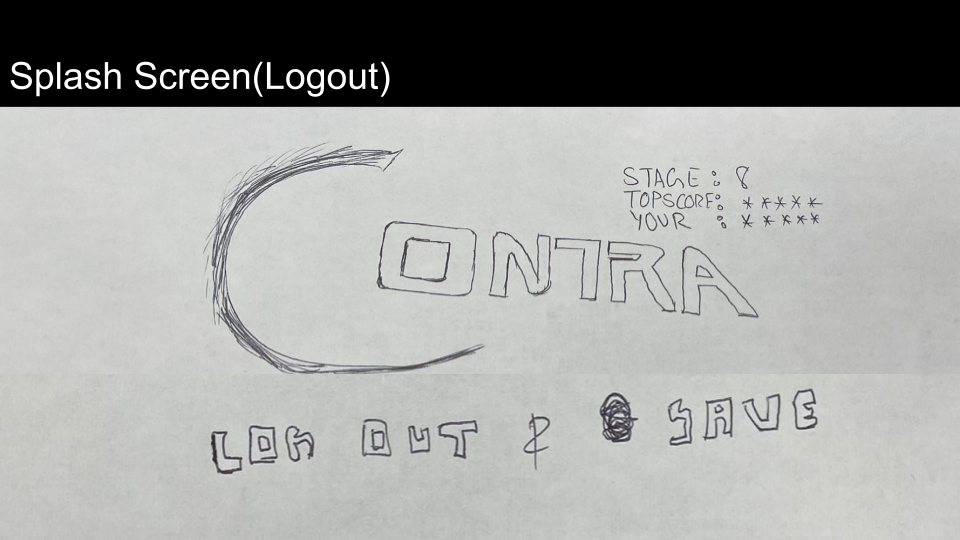
* Hardware malfunctions
* Incompatible code/code malfunctions
* Schedule availability
* Physical health
* Security
* Unknown factors

Looking at these risk factors the risk with the most potential to occur is most likely a piece of hardware or software being incompatible or malfunctioning. Rest assured we have a contingency plan if any of these risks were to come up.

# Wireframe diagrams/Storyboards



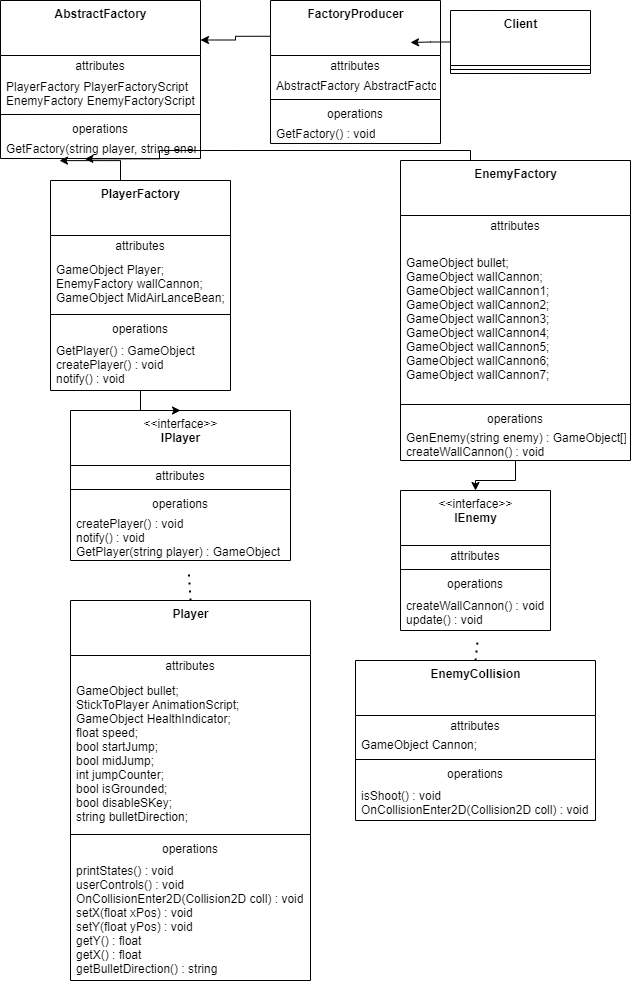




Our wireframe diagram above is showing how the user interacts with the start screen and logging into the database. The user then plays the game and in the picture above is attacking the final boss of Contra. After defeating the final boss or if the player gives up the user logs out and saves. The current user's information is shown next to the Contra logo stating the stage number, score, and username.

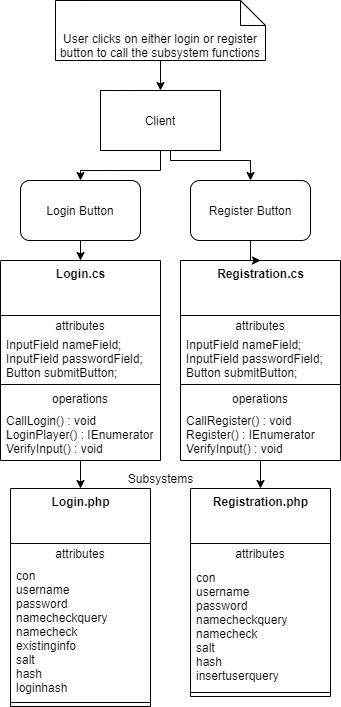
# OO Design Patterns

## Creational - Abstract Factory



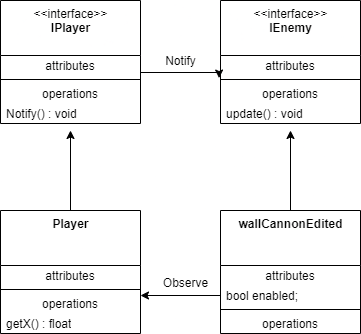
The diagram above is the creational object-oriented design pattern. The abstract factory design pattern was chosen because the different types of game objects in the game need to have their own groups but also be within a general factory. If the centralizing of our gameobjects creates too much code complexity, then we can separate the separate game objects into their own factories. The different object factories are PlayerFactory and EnemyFactory.

## Structural - Facade



The diagram above is the facade structural object-oriented design pattern. The facade design pattern was chosen because the database subsystems are sufficiently complicated and can benefit from having an accessible entry class. The save manager is using the Login and Registration scripts to input and verify data on the database. The login script is interacting with the php login script to input data like the username and password into the database. Whereas the registration script is verifying the data on the database by interacting with the registration php file.

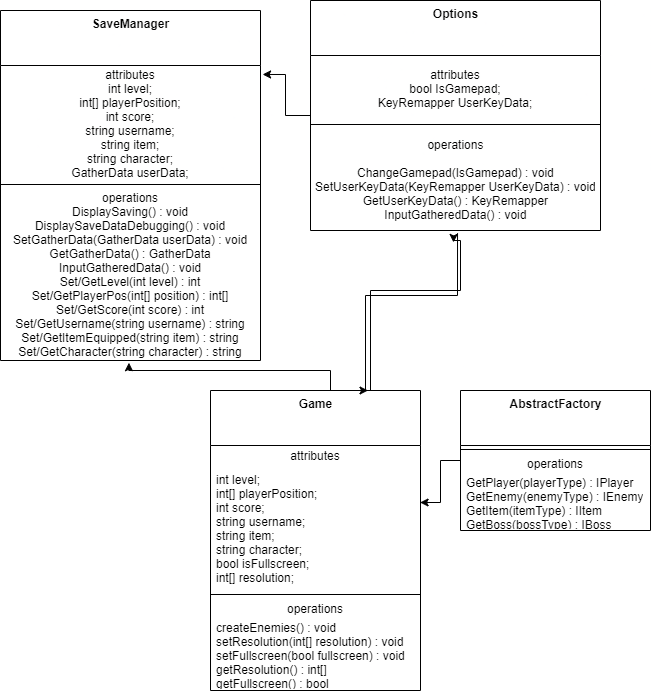
## Behavioral - Observer



This diagram is the observer behavior object-oriented design pattern. The observer design pattern was chosen because the Artillery game objects are activated only when the player is within a certain distance from them. The observer pattern watches for the player's x-axis position relative to them and a certain threshold activates the artillery game objects methods. The player knows that the observer pattern has activated the enemy when the sprite model changes, and the enabled attribute is set to true.

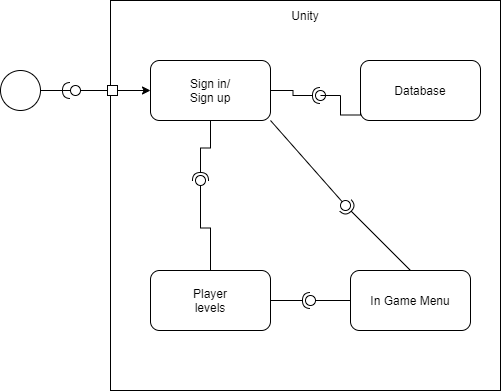
# Structural UML Diagram

## Class Diagram



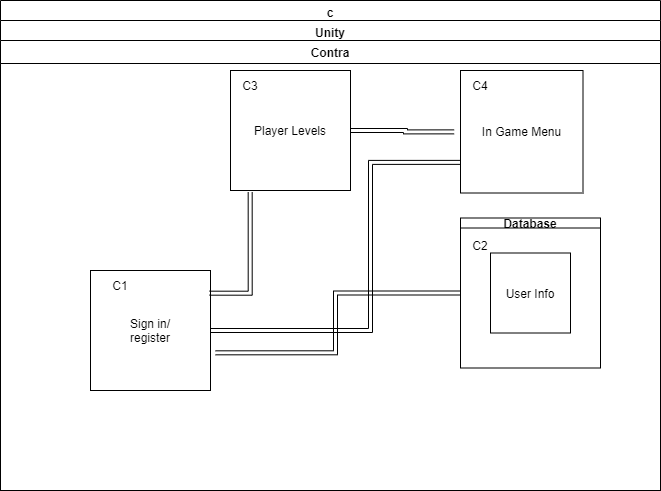
The diagram above is our class diagram for our remake of Contra. The game classes have been abstracted into the classes above. The Game class has attributes like the score and username of the player which gets gameobjects like the player and enemies through the abstract factory design pattern. The Game class calls on the Options and SaveManager facades to save game states to the database.

## Component diagram



The Contra component diagram outlines the interactions between the components like the sign in function, the database, and player levels. The user interacts with the system through the sign in/up component.

## Deployment diagram



The deployment diagram is outlining the interactions between the components of the game levels and saving to the database. These components are interacting within the unity engine and the user's computer allocating the necessary resources.

# Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field name | Datatype | Length | Constraint | Description |
| id | int | 10 | Primary key, Auto Inc. | Player ID |
| username | varchar | 255 | Not null, UNIQ | User name or player |
| salt | varchar | 255 | Not null | Salt value function |
| hash | varchar | 255 | Not null | Hash function |
| score | int | 10 | Default zero | Score of the player |

# Security

Our game is not that will be storing valuable data in its database or anything but for a game it must be somewhat secure to store information like players name and score and to not give access of one player to another and nobody plays the game without having the account. Following is our list of requirements of security our software follows.

* Authenticate user before login
* Secure connection to database
* Authenticate for user in every scene to track unauthorized user
* Not everybody can see anyone’s scores

Now let’s discuss why we choose the above listed security requirements and how we implemented them in our game. First one is about user authentication. The first thing that happens when a user starts typing their login credentials is counter starts and the button is turned interactable when 8 characters are entered. After that username goes under two steps of verification first is if the entered username is unique to the table, and other is if it matches to at least one row in the table. Next is password, once the username is found it is easy to find password using the primary key, but verifying the password requires two columns: hash and salt. We are using encryption of words into Shaw 256 for security purposes. Since our database is the most vulnerable point in our system, we decided to encrypt the data in it so even if the hackers hack into our database they won’t be able to see what the actual password is.

Second requirement we chose was the connection to the database. We have a name and password to our database which is only available to client, and team members. And they are the one who can make changes to the actual database. For making a connection of the database to our game we need to have a username and password, along with the URL of the database server.

Along with authenticating users for login, we also authenticate them when they are in the game, because if we don’t do that, they might just jump into the play scene and start playing the game which might cause the whole system.

If just anyone could see the information of anyone with no problem that would be a trouble. And all we did so far would not make any sense. To solve this problem, we just show users what belongs to them and hide all the rest.

## Security Threat Model

Diagram

Description automatically generated

Above is our security threat model and is divided into two compartments one is Player and game server and other is the database and the game server. Our weakest point according to us was the database server because we are just hosting it locally on our computers which is not the most secure and that is directly connected to our game to secure our database, we have username and password that is needed to connect to our database along with the link to the running server. And even in the database we have a salt and hash function to save valuable information which will be discussed later in the security section of the documentation.

# Software/System Performance

## Performance Requirements

Just like the software requirements for the functionality of all the methods and features, developers need to take performance into consideration. We haven’t talked about it in the last two sprints, because we really did not have any project to do our performance testing on. Following is our list of requirements for performance that we as a developer think should meet along with what our client’s needs.

* Minimize latency: As it sounds like we want to reduce the wait time for the players or admin as much as possible, this might be between the scene changes, database query calls and plenty of others. And to solve this problem we are using the latest version of Unity game engine and hosting our database locally and writing the scripts in a way that the complexity of the functions/methods is minimum possible.
* Reduce database lag: Our project is designed to use the database for a minimum period, in fact the database is used in the beginning of the game to authenticate users/ to register a new account, and at the end to save the progress of the game. But even to do so we want that experience of the user to be enjoyable and fast.
* Reduce bottlenecks: As it sounds like we don’t want our user to have the best equipment and still not be able to enjoy our game, so our goal is to keep our unity engine updated between our team members to avoid scripting for one version and another person not supporting that version. And we are taking the target system very seriously and designing our game accordingly to avoid bottlenecks between the user hardware and software.
* Minimize database redundancy: As mentioned earlier database is not the important part of the project but since it is then to improve performance our goal is to reduce redundancy by implementing tables in the database that follow third normalization.
* Multiple save state
* Careful game object design

## Performance Objectives

Now we have defined the requirements for our performance, it's time to see how we will test them once our game is ready for it.

* CPU performance: We will use Windows subsystem benchmarking tools for testing performance of our CPU while the game is running. We will just time how many processes are running and then time it.
* File I/O: This objective is not that important for our project since we will be using a database for reading and writing the information. But it is important to see how much a computer can do and this will be the required specification for the target hardware. This will also reduce bottlenecks.
* Response time: This is an important objective to test when you have a live database server, because there are some scenes and phases in the game that are completely dependent on the database. And to have a logy database will occur a problem in overall software performance.
* Processing time: This will test our code, meaning how long does it take to perform a task or a group of tasks. And we will test how long it takes for a player to jump, shoot, do both at same time and so on.
* Availability: Our goal is not for our game to be functional once but for it to have that functionality for a long period of time. This means just checking if all the functionalities are running as expected overtime they are called with no problem.
* Reliability: Along with availability of the function/method call at any given time it needs to successfully complete what it started. In other words, it should be reliable every time each function/method is called.

## Hardware/Software Bottlenecks

Reducing bottlenecks is also an important part of improving the performance of a software system. Bottlenecks means there is a gap between two processes or hardware pieces which result in slower performance. Here are some situations where our software might have bottlenecks.

* User hardware/operating systems are not taking advantage of the high-speed database we have to offer.
* Database server is not running as fast as php scripts are getting requests from the players/admin.

## System/Application Workload Data and Descriptions

## Time-Based Performance

Now we will start testing our game for performance. This is the time-based testing of some game objects; method calls and functions calls. We will discuss how we will perform the tests in this section and then in the next section we will show the results.

We will place start clock and end clock before and after the method call and then subtract it and print it so we can record it every time that method is called here is the syntax of how we will implement the test cases.

DatabaseCallMethod(String[] d){

          for 1 to String.length

            //format values in String[]

            //connect and update database with String data

    }//end method

Following are the methods we will be testing for the performance.

* Jumping
* Moving
* Shooting
* Cannons
* Login
* Register

## Test Case definitions and Results (Graphs)

On our test we decided to go with ticks interval for some test cases, that is pre-defined for c# scripts instead of seconds or milliseconds, because some of the function calls like jumping and moving forward and backwards are so simple and don't take a lot of time, because of that it was just showing zero milliseconds but ticks were showing some number. We are not exactly sure what interval ticks is using but it does go up when we call two functions at the same time or throw more workload on it.

First graph we have is for jumping and moving the player. Here is the script for our test case. What it does is we start the timer when the method is called and then we stop it when the method is done processing and display that time on the console.

public void setJumpingAnim()

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

//doing the method work

this.PlayerAnimation.SetTrigger("jumping");

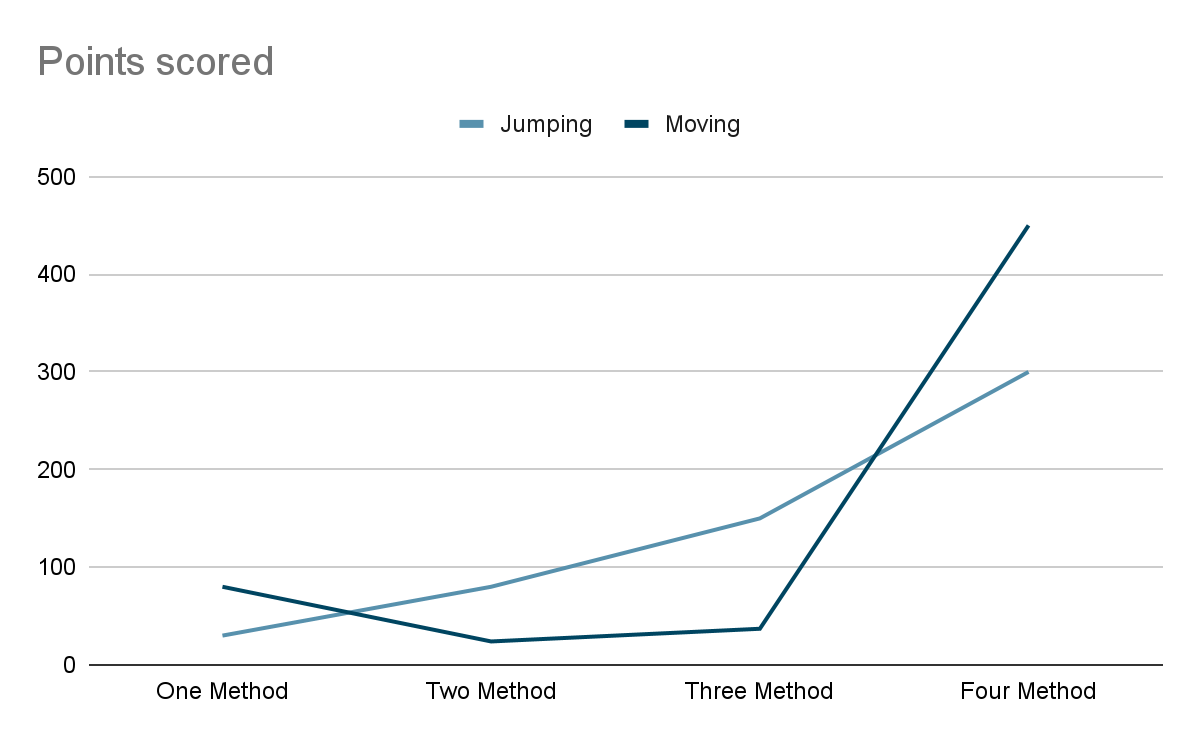
//method work is complete

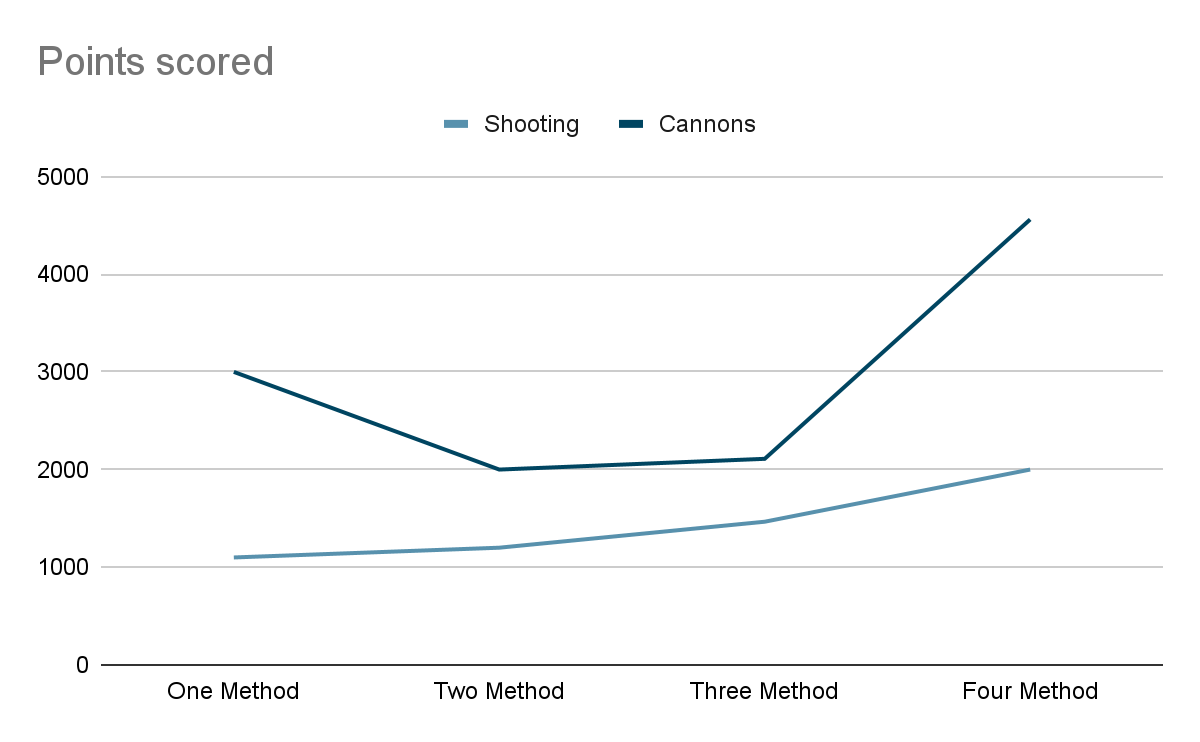
watch.Stop();

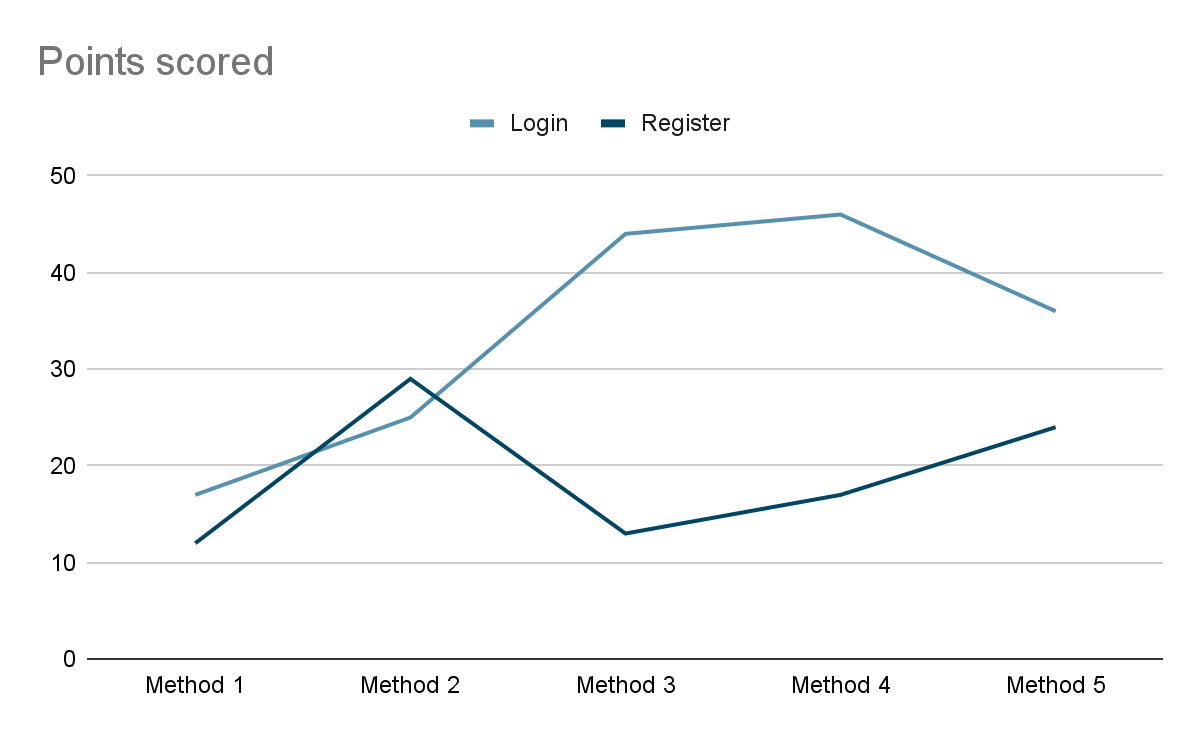
print($"Execution Time: {watch.ElapsedTicks} ticks");

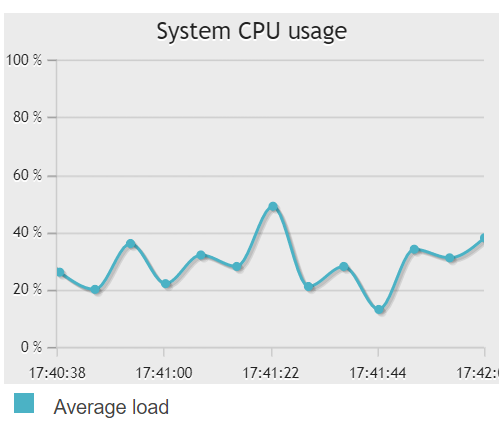
}

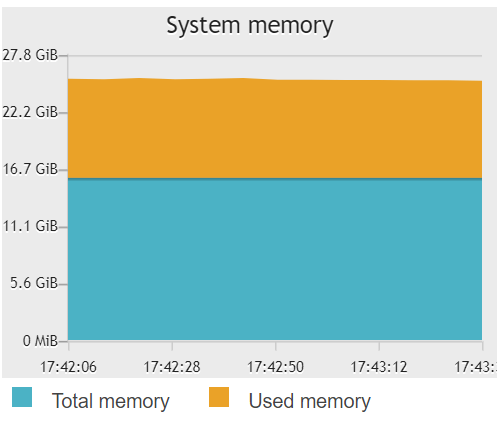
Fowling is the graph where x axis is the number of methods called at the same time and y-axis is the ticks on this graph to complete the process.











## Rate-Based Performance Data and Descriptions

We used a frame rate script attached to the main camera of every critical game scene to calculate the average fps. The 10 output statements were collected from running each scene, added, and then divided by 10 to get the average.

Main Menu - 46.87134, 60.59248, 61.03069, 59.45268, 35.65418, 18.67832, 29.37936, 58.83287, 30.6677, 59.46081

Avg fps = 46.062043

Login - 19.08087, 32.8322, 51.65262, 66.32927, 57.83957, 58.95531, 63.52473, 56.49239, 60.81393, 56.34311

Avg fps = 52.3864

Register - 41.3189, 60.92063, 37.65103, 17.53605, 59.1478, 61.04857, 62.0694, 60.54442, 57.86903, 63.68859

Avg fps = 52.17944

Resume - 19.32214, 60.53928, 60.13446, 75.58865, 19.44417, 54.68904, 60.7574, 60.36934, 56.7112, 20.48765

Avg fps = 48.804333

Level 1 - 20.3916, 50.61882, 58.4833, 33.07502, 52.465562, 59.75572, 58.76407, 60.91655, 56.54414, 20.43502

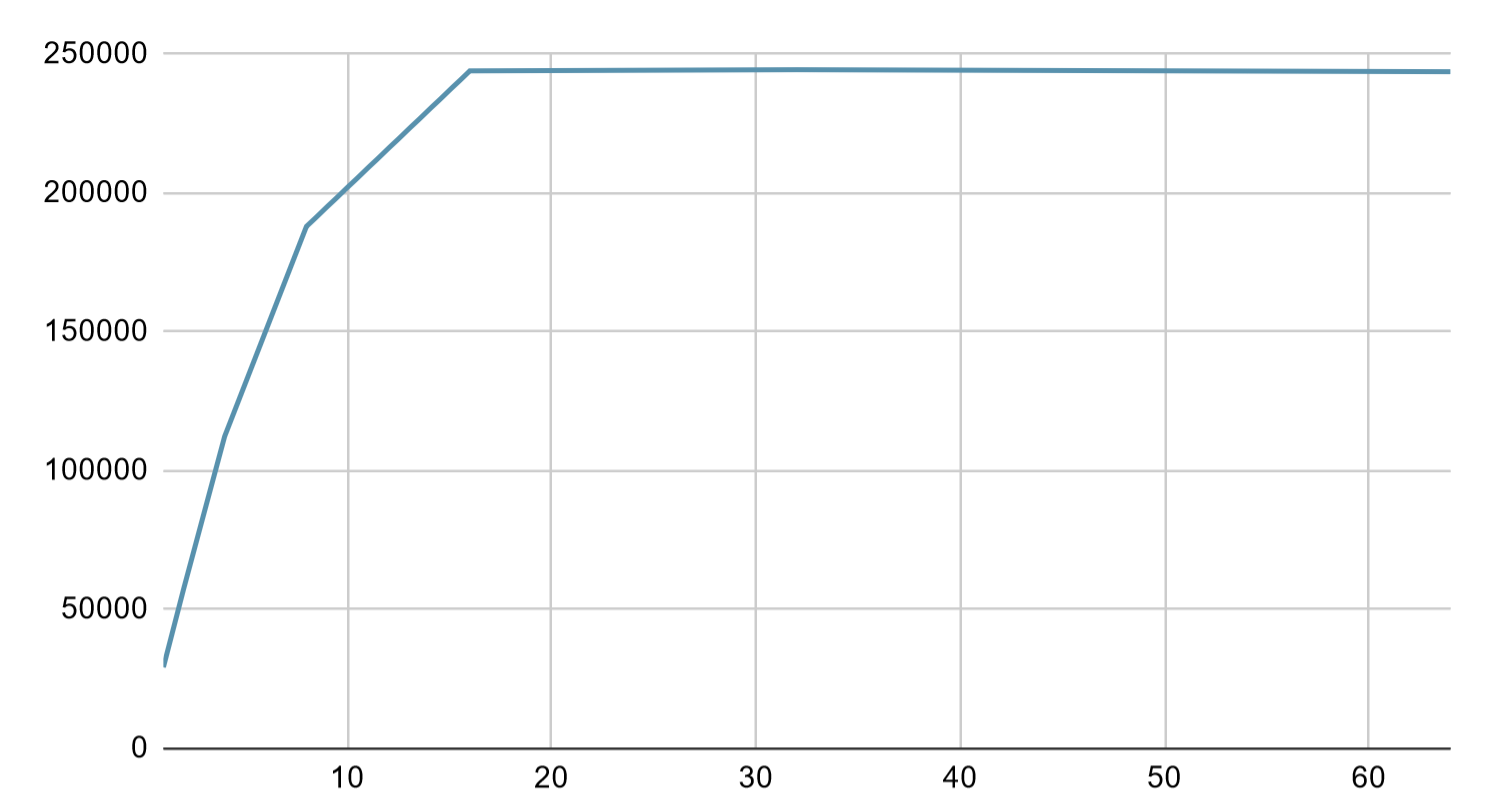
Avg fps = 47.1449802

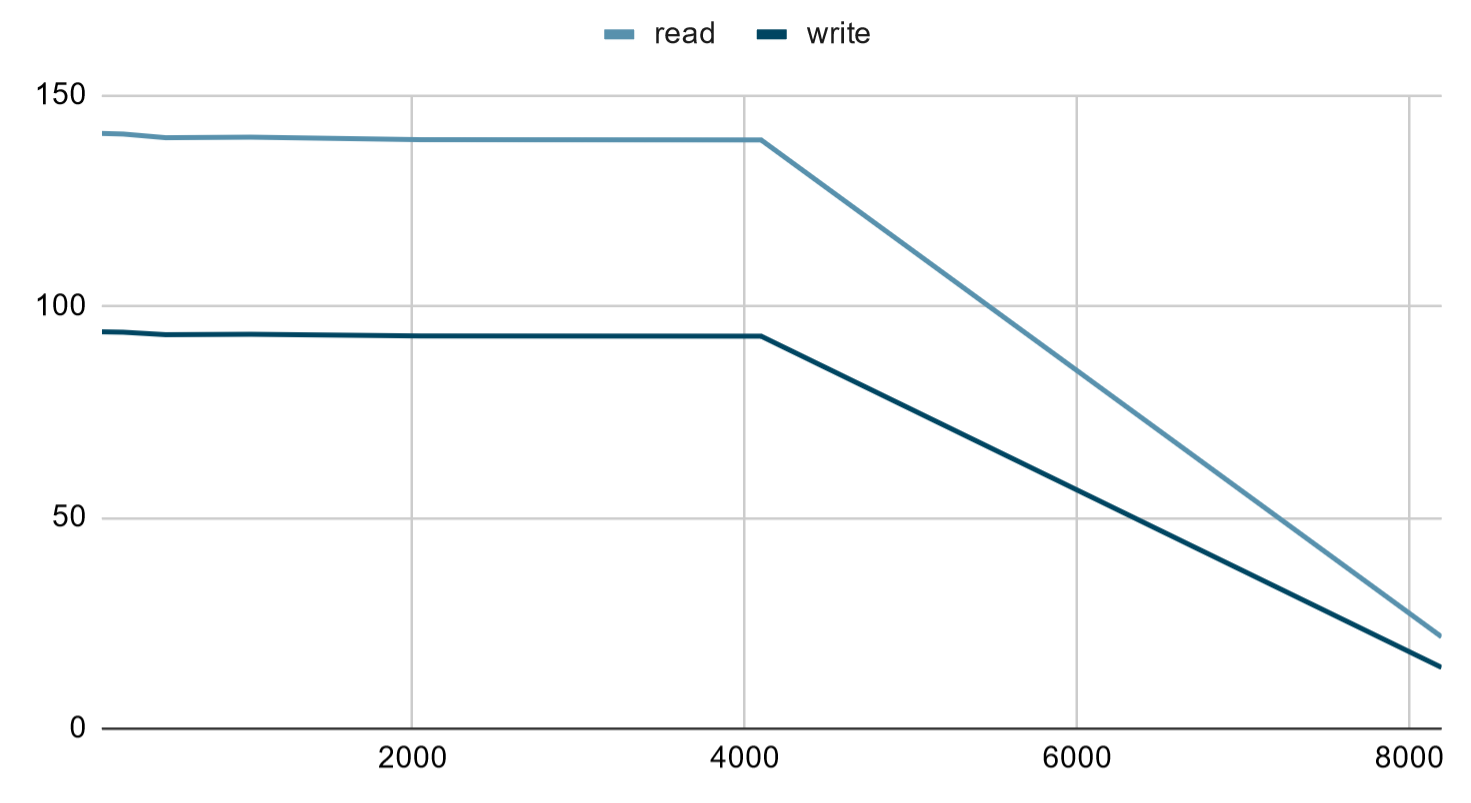
## Sysbench data

CPU performance: This is the CPU performance using the benchmarking utility sysbench. The graph shows the line's increasing slope that begins to stall around the 16-thread ma

rk on the X axis and ~220K total events are performed at that point.

File I/O performance: This is the File I/O performance using the benchmarking utility sysbench. The graph shows the line's decreasing slope as it approaches the 8GB mark on the X axis. The read starts at 150 and write starts at 100 and maintains a straight line until the 4GB mark is passed at which the slope drastically decreases.





# Software Testing

## Software Testing Checklist

In Software System Testing following steps needs to be executed:

1. The first step is preparation of System Test Plan:
   1. Introduction
   2. Scope
   3. Critical areas Area to focus
   4. Test Deliverable
   5. Pass/Fail Criteria
   6. Testing Strategy for System testing
   7. Testing Schedule
   8. Test Environment
   9. Risks and Mitigation
   10. Approvals
2. Second step is to create Test Cases: It is very much like functional test case writing. In test case writing you should write the test scenarios & use cases.

## Unit Testing

Test Plan Identifier: 1

Introduction: In this section we will perform unit testing on our project. For the unit testing we will be selecting the major methods and functions which can be compiled as independent programs.

Test item: For the scope of this project we are focusing on three important methods that will be used the most during the player playing the game, first method is login/logout, then userControls, and Oncollision2D method which determines whether player should be destroyed after collision or not.

Approach: Our approach on this test is to start with code reviews as a team and then do testing on a method at least ten times in all possible ways and see if the expected results are generated or not.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Cases | Input Values | Expected Output | Pass/Fail | Date Tested |
| # | Identify the test cases along with the expected results. |  |  |  |  |
| 1 | Example:  Test Procedure:  Login with a user account name and password.  Username:  123456789  Password:  123456789  Expected Results:  Successful user login. otherwise, an error will be displayed for the wrong credentials. | Username:  12345678  Password:  12345689  Username:  1234567  Password:  1234567  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  123456789  Password:  123456789  Username:  12345  Password:  12345 | Login failed  Login failed  Login pass  Login Pass  Login Pass  Login Pass  Login Pass  Login Pass  Login Pass  Login failed | Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass | 12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021 |
| 2 | Example:  Here we are testing a method for determining if we can kill a player based on where the player is. We have kill land and regular land. Our input will be Collision class and object type coll called “Kill land”, “Wallcannonedited”,  “Land” and “Solid Land” | “Kill Land”  “Land”  “wallCannoneditied”  “Kill Land”  “wallCannoneditied”  “solidLand”  “Kill Land”  “wallCannoneditied”  “Kill Land”  “Land”  “Kill Land”  “solidLand” | Destroy  Not Destroy  Destroy  Destroy  Destroy  Not Destroy  Destroy  Destroy  Destroy  Not Destroy  Destroy  Not Destroy | Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass  Pass | 12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021  12/6/2021 |

Item pass/fail criteria: The item pass/fail criteria for unit testing is to have no syntax errors and all the paths are bug free and are in working conditions.

Environmental needs: The environmental needs for the Unit tests require the project. The basic and performance tests require the unity editor.

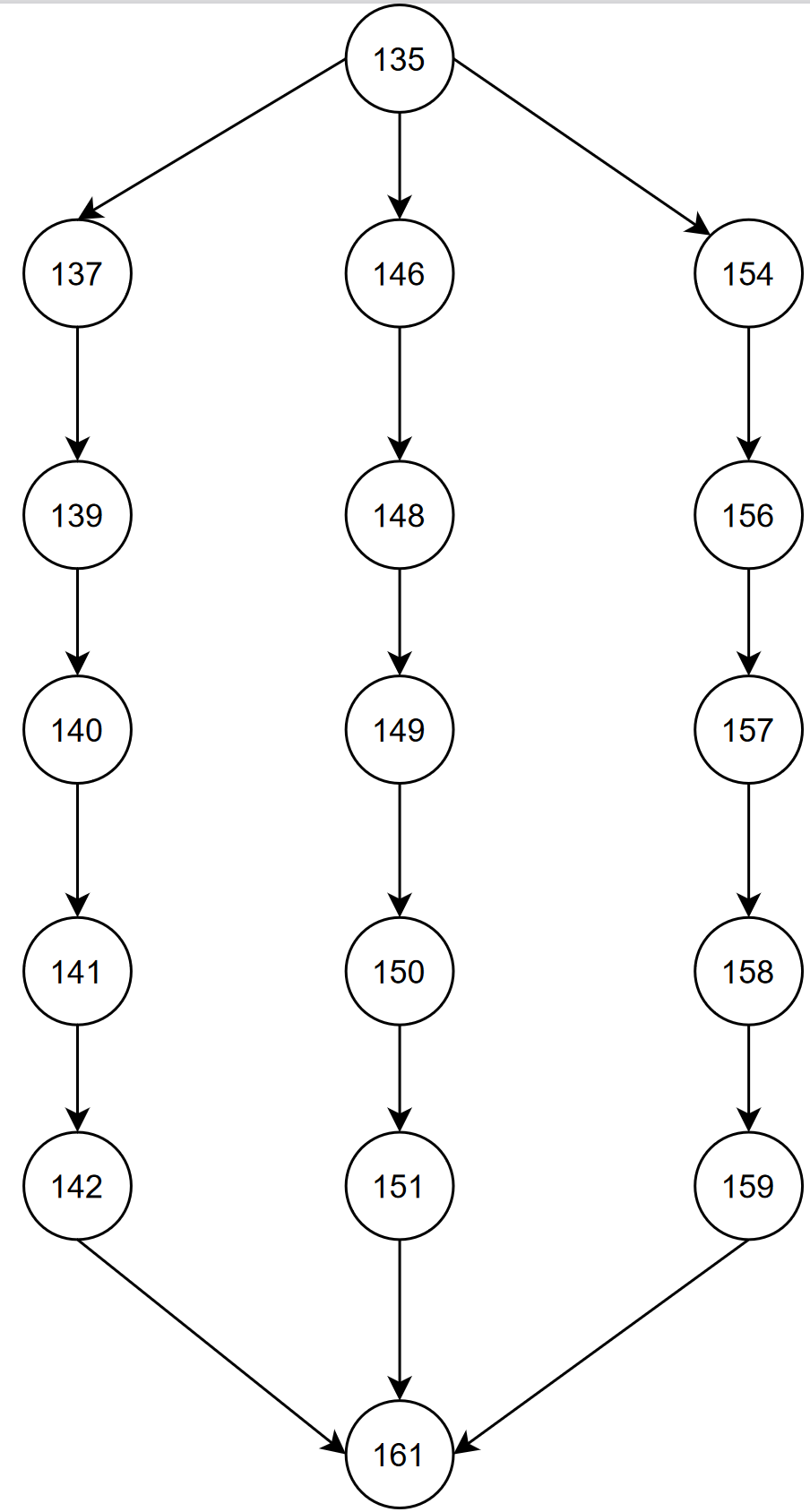
Responsibilities: N/A

Schedule: N/A

Risks and Mitigation: N/A

Approvals: Ethan Moore / Aman Patel / Will Craddock - 12/6/2021

## Flow graphs



Oncollision2D in player.cs

N=17

E=18

P=1

CC= E-N+2P

CC=18-17+2

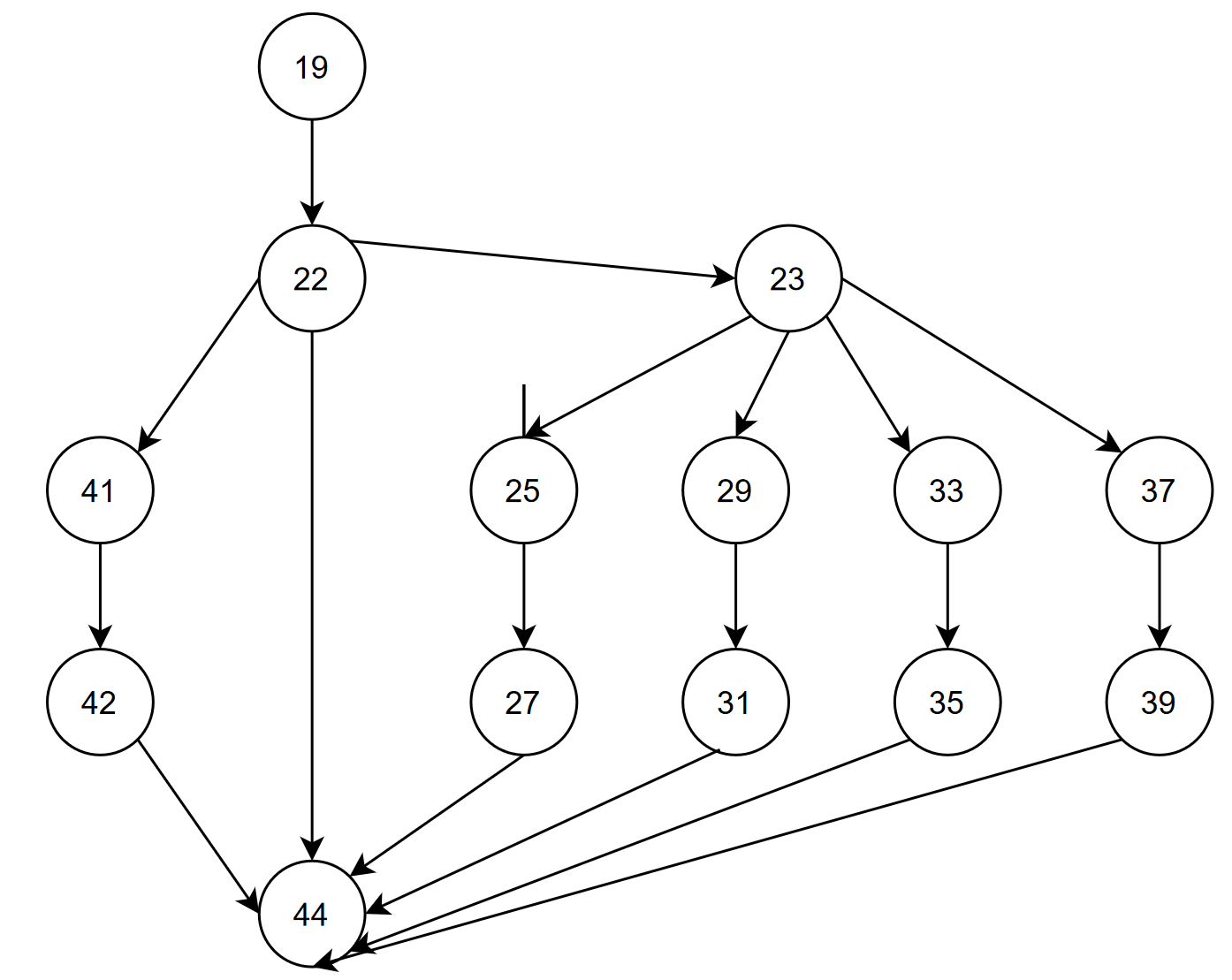
CC=3

Path 1: 135, 137,139,140, 141, 142, 161 //Kill Land or CannonCollision

Path 2: 135, 146, 148, 149, 150, 151, 161 //Land

Path 3: 135, 154, 156, 157, 158, 159, 161 //Solid Land

Above flow-chart is for the Oncollision2D method which can be found in the Player.cs script. This method has three paths. First path is when the player is not in the land it is supposed to or touching the cannon. Next two paths are not destructive paths; this is where the player can relax or will not get killed. For testing this individual method, we used string as an input. We had four types of input: “Kill Land”, “wallCannonedited”, “SolidLand” and “Land”. And then this is gone for comparison in the if statement and if the statement is correct accordingly the rest of the code is executed.



Update in bullet.cs

N=15

E=18

P=1

CC= E-N+2P

CC=18-15+2

CC=5

Path 1: 19, 22, 44

Path 2: 19, 22, 41, 42, 44

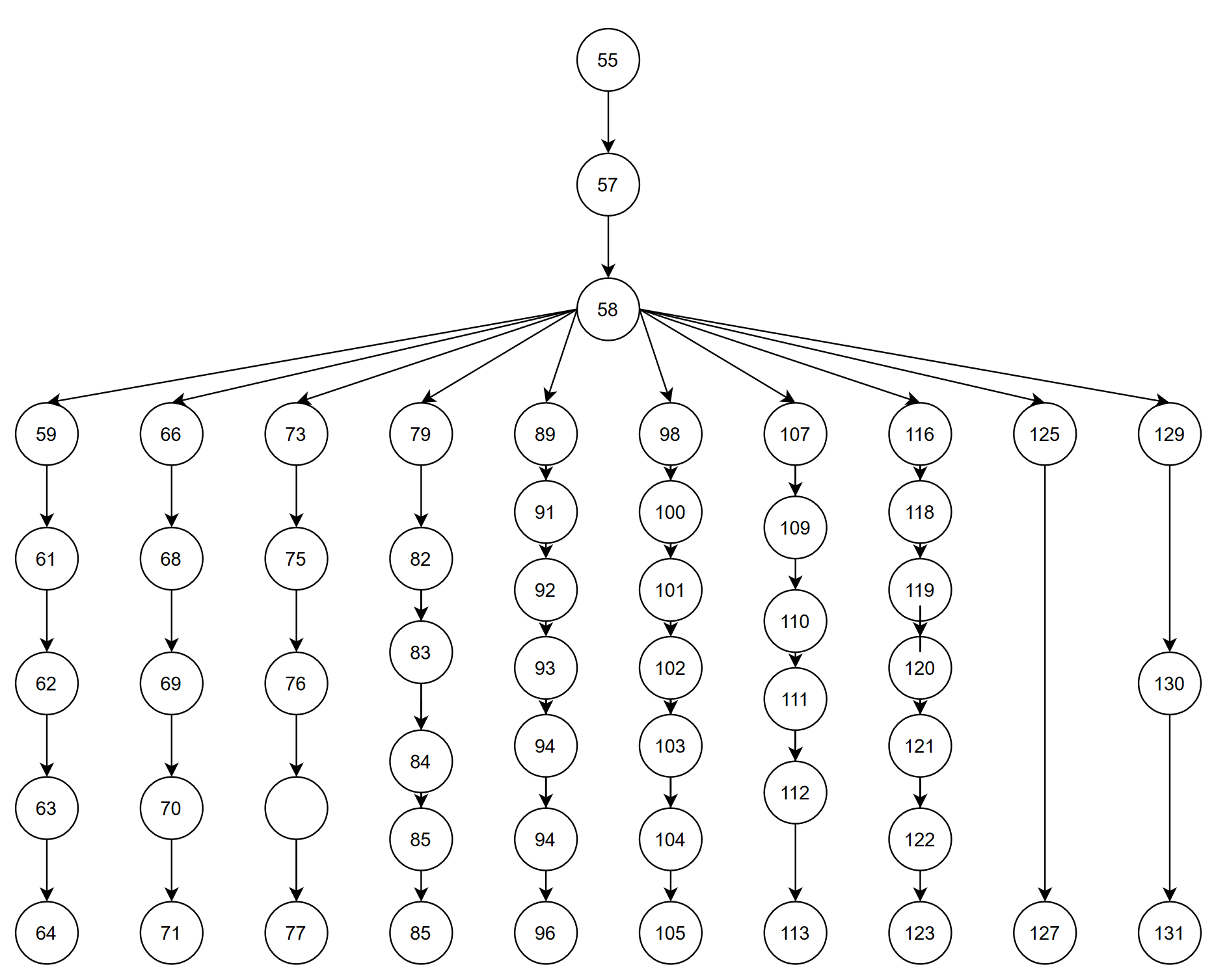
Path 3: 19, 22, 23, 25, 27, 44

Path 4: 19, 22, 23, 29, 31, 44

Path 5: 19, 22, 23, 33, 35, 44

Path 6: 19, 22, 23, 37, 39, 44

For the above flow graph, we are using the update method from bullet.cs script. We have six different paths here and these are all for different directions for the bullet and for the visualization of the bullet. All the if statements are for what directions the bullets are fired and the else statement is for if the bullet is fired or not.



userControls in Player.cs

N=55

E=55

P=1

CC= 55-55+2

CC=2

Path 1: 55, 57, 58, 59, 61, 62, 63, 64

Path 2: 55, 57, 58, 66, 68, 69, 70, 71

Path 3: 55, 57, 58, 73, 75, 76, 77

Path 4: 55, 57, 58, 79, 82, 83, 84, 85, 86

Path 5: 55, 57, 58, 89, 91, 92, 93, 94, 96

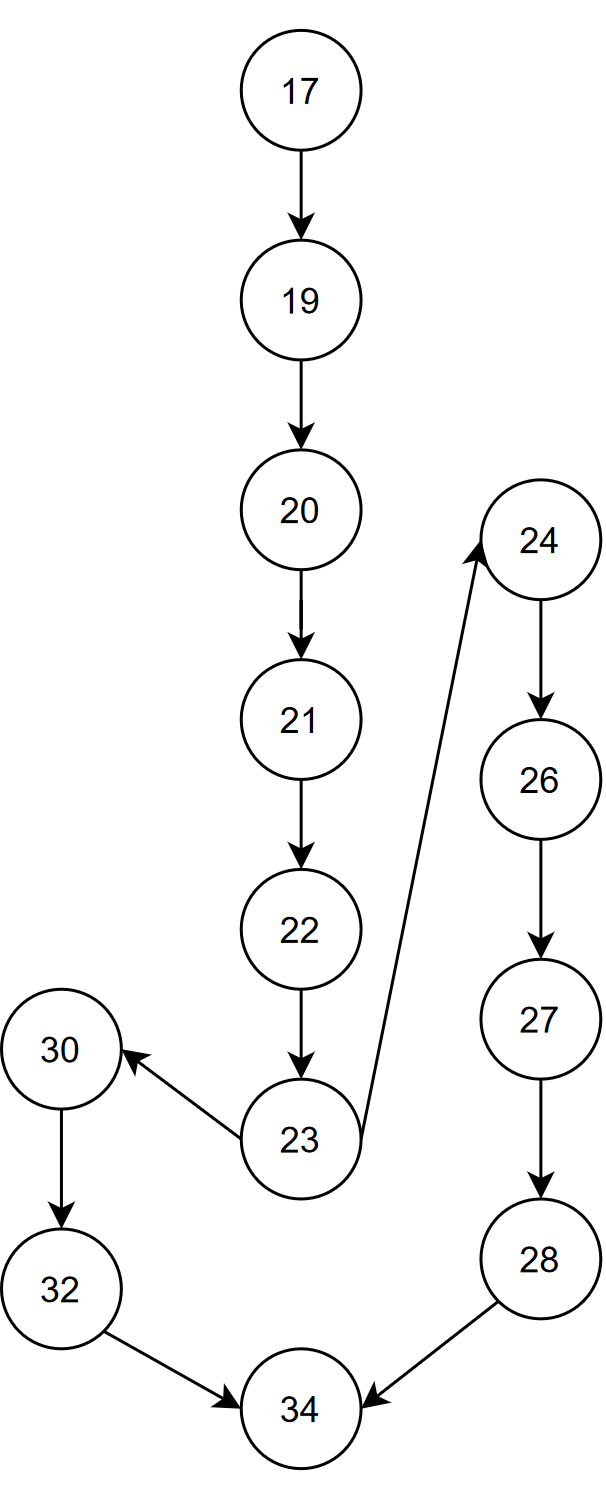
Path 6: 55, 57, 58, 98, 100, 101, 102, 103, 104, 105

Path 7: 55, 57, 58, 107, 109, 110, 111, 112, 113

Path 8: 55, 57, 58, 116, 118, 119, 120, 121, 122, 123

Path 9: 55, 57, 58, 125, 127

Path 10: 55, 57, 58, 129, 130, 131



Login and Logout

N=13

E=13

P=1

CC=13-13+2

CC=2

Path 1: 17, 19, 20, 21, 22, 23, 30, 32, 34

Path 2: 17, 19, 20, 21, 22, 23, 24, 26, 27, 28, 34

## Integration Testing

Test Plan Identifier: 4

Introduction: Our integration test plan consists of a top-down approach on the user buttons module connecting to the database and the module that brings the user back to the main menu.

Test item: The SUT will show if the database can successfully index the users information like the username and password. We are testing only the mainmenu’s register button and indexing that data. All other features of the main menu will not be tested since the registration functionality has a higher priority than the login function.

Approach: Our strategy to functionally test the software is to utilize a top-down approach with appropriate print statements in the mainmenu and registration scripts that are manually tested. Mainmenu is the first module and the registration script is the second module under test.

Test deliverables:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Input | Expected Output | Pass/Fail | Date Tested |
| 11 | Register button and user data is passed and indexed into the database. | User clicks register button  Username = TestUsername  Password = TestPassword |  | Pass  Pass | 12/6/21 |

Item pass/fail criteria: Upon clicking register with appropriate data in the username and password fields the database should accept the input and send the user to the level1 scene.

Environmental needs: The infrastructure required for the integration tests is Unity Editor.

Responsibilities: N/A

Schedule: N/A

Risks and Mitigation: Some risks include the database not being connected to a local instance or not updated dependencies breaking the buttons.

Approvals: Ethan Moore / Aman Patel / Will Craddock - 12/6/2021

Defined and Executed System Tests and Results(With Testing Plan)

## System Testing

Test Plan Identifier: 5

Introduction: Our system test plan consists of documentation, basic, and performance system tests.

Test item: The SUT is the whole game starting and documentation. The features that are being tested are the ability for the game to load, documentation being up to date, and the game running at an acceptable frame rate. The out of scope features are everything else not being tested because they are not a priority for this testing type.

Approach: Our approach for the documentation tests requires a verification that the spelling, grammar, and format is checked and it is a manual test. The approach for the basic test is to have the modules in the main menu print a statement that they are functional/loaded. This is an automatic test. Our approach for the last test on performance will be automated and be the framerate for each scene.

Test deliverables:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test Case | Input | Expected Output | Pass/Fail | Date Tested |
| 12 - Documentation | Spelling/Grammar errors free | N/A |  | Pass | 12/7/21 |
| 13 - Basic | On game start mainmenu.cs should output “mainmenu loaded” | N/A |  | Pass | 12/7/21 |
| 14 - Performance | After 10 iterations the average fps should be greater than 30. | N/A | Avg fps = ~43.86 | Pass | 12/7/21 |

Item pass/fail criteria: The pass/fail criteria for the documentation test is to have no errors in spelling, grammar, and the format that is required. The pass/fail criteria for the basic test are to have the mainmenu script print “mainmenu loaded”. The pass/fail criteria for the performance test are to on average be above 30fps.

Environmental needs: The environmental needs for the documentation tests requires the documentation. The basic and performance tests require the unity editor.

Responsibilities: N/A

Schedule: N/A

Risks and Mitigation: Some risks include losing the documentation to perform the documentation tests or not having the unity editor files for our Contra Remake.

Approvals: Ethan Moore / Aman Patel / Will Craddock - 12/6/2021

## Validation/Acceptance Testing

**Validation Testing**

Introduction:

The purpose for this type of test is to ensure that we made the game in accordance to the problem given in class which was to make the game contra.

Features to Test/Tot Test:

The features that are going to be tested in validation testing are the core components in our contra remake such as login, sign up, and being able to play level 1. Where not going to be testing the functional/ nonfunctional requirements

Approach:

The testing will involve a series of pass/fail tests which will be tested in the client meeting by the client.

Pass/ Fail Criteria:

The project is a remake of the game Contra

The user can sign up a new profile

The user can login using that new profile

The user can play the first level of contra

Deliverables:

|  |  |
| --- | --- |
| The project is a remake of the game Contra | Pass |
| The user is able to sign up a new profile | pass |
| The user is able to login using that new profile | pass |
| The user is able to play the first level of contra | pass |

Environment: The unity editor is required for these tests.

Schedule: NA

Staffing and Training Needs:

Needs to be familiar with unity

Responsibilities: NA

Risks and Mitigations: Some risks for this part is that if signing up or signing in is not on our local database the test will automatically fail

Approvals:

Aman Patel 12/7/2021

Ethan Moore 12/7/2021

Will Craddock 12/7/2021

**Verification**

Introduction:

The purpose for this type of test is to ensure that we made the game as we had designed it back in sprint 2

Features to Test/Tot Test:

The features that are going to be tested in validation testing are the components that cover our functional and nonfunctional requirements. We are not going to be testing what we tested in the validation testing.

Approach:

The testing will involve a series of pass/fail tests and criteria which will be tested in the client meeting by the client. And the nonfunctional requirements being tested by us.

Pass/ Fail Criteria:

Non-functional:

The project can run on Unity

The source code should be made in house

The project will work with myPHP admin

Security standards will work with sign up and login

Functional:

The game can be played with mouse and Keyboard

Multiple Users should be able to Sign up, Login in have their profile saved into the database

User should be able to play as soon as they login with their info

Deliverables:

Verification:

Non-functional:

|  |  |
| --- | --- |
| Th project can run on unity | pass |
| The source code should be made in house | pass |
| The project will work with myPHP admin | pass |
| Security standards will work with sign up and login | pass |

Functional:

|  |  |
| --- | --- |
| The game cab be played with mouse and keyboard | pass |
| Multiple Users should be able to Sign up, Login in have their profile saved into the database | pass |
| User should be able to play as soon as they login with their info | pass |

Environment: The unity editor is required for these tests and access to php my admin.

Schedule: NA

Staffing and Training Needs: N/A

Needs to be familiar with unity and phpMyAdmin

Responsibilities: NA

Risks and Mitigations: Some risks for this part is that if signing up or signing in is not on our local database the test will automatically fail

Approvals: Aman Patel, Ethan Moore, Will Craddock 12/7/2021

**Acceptance Testing:**

Introduction: The purpose of this test is to get someone outside the development team to go over pass fail criteria we have for the acceptance test.

Test Item:

Contra remake in Unity

Failure to test/ not test:

For the acceptance testing the user will be testing the login/signup, movement, shooting, and turret destruction. Things that will not be tested are anything that doesn’t have to do with how the user interacts with the game and in the game.

Test Deliverables:

Main Menu:

|  |  |
| --- | --- |
| Pass/Fail Criteria | Pass/Fail |
| User is able to click the buttons to move to a new scene | N/A, no ability to test |
| User is able to sign up on the sign up page | fail, no connection to database |
| User is able to login in the Login Page | no connection to database |

Level One:

|  |  |
| --- | --- |
| Pass Fail Criteria | Pass/Fail |
| User should be able to move using AWSD | pass |
| User is able to shoot using the arrow keys | Pass |
| User is able to jump using space bar | Pass |
| User is able to kill the stationary turrets | pass |

Pass/Fail Criteria

Main Menu:

User can click the buttons to move to a new scene

User is able to sign up on the sign-up page

User can login in the Login Page

Level One:

User should be able to move using AWSD

User can shoot using the arrow keys

User can jump using space bar

User can kill the stationary turrets

The reason so many tests failed when it came to the database is that on our side, we ran the database locally. When the client went to play the game, he did not have access to the database since it wasn’t downloaded onto his system.

Responsibility: N/A

Staffing and Training Needs: N/A

Risks and Mitigation: Some risks for this part is that if signing up or signing in is not on our local database the test will automatically fail

Approvals: Aman Patel, Ethan Moore, Will Craddock 12/7/2021

# Concluding Remarks/Future work

This project has been an amazing learning experience for all of us. And our client/mentor for the project Dr. Galloway has been a really amazing guide throughout the process and helped us where we were feeling lost or felt like we can’t move any forward. But there are some things we would like to do if we just had more time in our favor. Here is the list of things we would do if we had more time.

* Implement more enemies: Enemies were the one of the most difficult assets to create, well not just creating it but to make it work where it kills the player when it is supposed to and not when it is not supposed to be the biggest problem. But until we figured out how to efficiently and correctly do it, we were out of time. The original Contra game had over twenty different small enemies and ten main bosses after each stage.
* Create more interactive player character: Our player script is the longest and most complicated, well after the code reviews we have trimmed it a little and made it less messy, but if we had more time, we would like to fix the bugs the player character has and allow users to choose from different character options.
* Implement health feature: We were so caught up in our kill feature that we were not able to implement the health feature to its entirety. We have just one health and if a player is hurt once it must start the whole level again.

# Appendix

Build Instructions for the project

<https://github.com/eldm-ethanmoore/Contra-Remake-CS360/blob/master/README.md>

## Final Source Code with Comments

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AbstractFactory : FactoryProducer

{

// Declares Scripts for the enemy and player factory

PlayerFactory PlayerFactoryScript;

EnemyFactory EnemyFactoryScript;

// Start is called before the first frame update

void Start()

{

// Calls Factory Method

GetFactory("Lance Bean", "Wall Cannon");

}

// GetFactory method initializes the script objects of type PlayerFactory and EnemyFactory

public void GetFactory(string player, string enemy)

{

GameObject PlayerFactoryScript = GameObject.Find("PlayerFactory");

this.PlayerFactoryScript = PlayerFactoryScript.GetComponent<PlayerFactory>();

GameObject EnemyFactoryScript = GameObject.Find("EnemyFactory");

this.EnemyFactoryScript = EnemyFactoryScript.GetComponent<EnemyFactory>();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Bullet : MonoBehaviour

{

Player PlayerScript;

Renderer renderer;

public string bulletDirection;

// Start is called before the first frame update

void Start()

{

// Initializes the PlayerScript, renderer, and bulletDirection

GameObject PlayerScript = GameObject.Find("Player");

this.PlayerScript = PlayerScript.GetComponent<Player>();

this.bulletDirection = this.PlayerScript.getBulletDirection();

}

// Update is called once per frame

void Update()

{

/\*

While the bullet is visible from the cameras point of view then

the bullet gameobject is alive. Otherwise the bullet gameobject

is destroyed. Depending on the direction that the bulletDirection

is set to the corresponding X or Y position is changed.

\*/

this.renderer = GetComponent<Renderer>();

if(this.renderer.isVisible)

{

if(this.bulletDirection == "right")

{

setX(getX()+.1f);

}

else if(this.bulletDirection == "left")

{

setX(getX()-.1f);

}

else if(this.bulletDirection == "up")

{

setY(getY()+.1f);

}

else if(this.bulletDirection == "down")

{

setY(getY()-.1f);

}

} else {

Destroy(gameObject);

}

}

// Getters and setters for the individual bullets

public float getY()

{

return transform.position.y;

}

public float getX()

{

return transform.position.x;

}

public void setX(float xPos)

{

transform.position = new Vector2(xPos, getY());

}

public void setY(float yPos)

{

transform.position = new Vector2(getX(), yPos);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraToPlayer : MonoBehaviour

{

Player PlayerScript;

bool isSnapped = false;

// Start is called before the first frame update

void Start()

{

// Initializes PlayerScript and player gameobject

GameObject player = GameObject.Find("Player");

this.PlayerScript = player.GetComponent<Player>();

}

// Update is called once per frame

void Update()

{

/\*

While the camera is not snapped nothing changes.

Otherwise the camera moves depending on the characters X position.

\*/

checkSnap();

if(isSnapped)

{

snapCamera();

}

}

// Checks gameobject player's position.

public void checkSnap()

{

if(this.PlayerScript.getX() >= 0.5)

{

this.isSnapped = true;

}

}

// Sets cameras X position to the corresponding player gameobjects X position.

public void snapCamera()

{

float x = this.PlayerScript.getX();

transform.position = new Vector3(x, gameObject.transform.position.y, -1.26f);

}

// Getters for X and Y

public float getY()

{

return transform.position.y;

}

public float getX()

{

return transform.position.x;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class DBManager

{

// Declares username and score vars

public static string username;

public static int score;

// Getter for if user is loggedIn

public static bool LoggedIn { get {return username!=null; } }

// LogOut sets username to null

public static void LogOut() {

username = null;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyCollision : MonoBehaviour

{

// Declares Cannon and bullet gameobjects

GameObject Cannon;

public GameObject bullet;

void Start()

{

// Checks if player gameobject is in range.

// Initializes Cannon decleration

this.Cannon = GameObject.Find("wallCannonEdited");

isShoot();

}

void update()

{

// Checks if player gameobject is in range.

isShoot();

}

// NOT OPERATIONAL

// isShoot trys to make the cannon fire a bullet when the player enters range

public void isShoot()

{

GameObject Player = GameObject.Find("Player");

Player PlayerScript = Player.GetComponent<Player>();

if(PlayerScript.getX() >= transform.position.x-0.5f)

{

print("in range");

Instantiate(bullet, new Vector3(transform.position.x-0.1f, transform.position.y, 0), Quaternion.identity);

}

}

// When a bullet collides with the cannon then the cannon is destroyed

void OnCollisionEnter2D(Collision2D coll)

{

if (coll.gameObject.name == "Bullet(Clone)")

{

Destroy(this.gameObject);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyFactory : AbstractFactory

{

// Declares all of the cannons in the level 1 scene.

public GameObject bullet;

public GameObject wallCannon;

public GameObject wallCannon1;

public GameObject wallCannon2;

public GameObject wallCannon3;

public GameObject wallCannon4;

public GameObject wallCannon5;

public GameObject wallCannon6;

public GameObject wallCannon7;

// Initializes one wallCannon

void Start()

{

GameObject wallCannon = GameObject.Find("wallCannonEdited");

this.wallCannon = wallCannon;

createWallCannon();

}

// NOT OPERATIONAL

// Print out all the created cannon gameobjects

public GameObject[] GetEnemy(string enemy)

{

GameObject[] enemyArray = new GameObject[7];

enemyArray[0] = this.wallCannon1;

enemyArray[1] = this.wallCannon2;

enemyArray[2] = this.wallCannon3;

enemyArray[3] = this.wallCannon4;

enemyArray[4] = this.wallCannon5;

enemyArray[5] = this.wallCannon6;

enemyArray[6] = this.wallCannon7;

return enemyArray;

}

// Instantiates all of the cannon's in level 1 at specific spots.

public void createWallCannon()

{

this.wallCannon1 = Instantiate(wallCannon, new Vector3(4f, 0.25f, 0), Quaternion.identity);

this.wallCannon2 = Instantiate(wallCannon, new Vector3(10f, 0.25f, 0), Quaternion.identity);

this.wallCannon3 = Instantiate(wallCannon, new Vector3(15f, 0.69f, 0), Quaternion.identity);

this.wallCannon4 = Instantiate(wallCannon, new Vector3(22.5f, -0.45f, 0), Quaternion.identity);

this.wallCannon5 = Instantiate(wallCannon, new Vector3(24.1f, 0.65f, 0), Quaternion.identity);

this.wallCannon6 = Instantiate(wallCannon, new Vector3(31.01f, -0.1f, 0), Quaternion.identity);

this.wallCannon7 = Instantiate(wallCannon, new Vector3(20f, 0.69f, 0), Quaternion.identity);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FactoryProducer : MonoBehaviour

{

// Declares AbstractFactory.

AbstractFactory AbstractFactoryScript;

void Start()

{

// Initializes AbstractFactory and calls the GetFactory method.

GameObject AbstractFactoryScript = GameObject.Find("AbstractFactory");

this.AbstractFactoryScript = AbstractFactoryScript.GetComponent<AbstractFactory>();

this.AbstractFactoryScript.GetFactory("Lance Bean", "Wall Cannon");

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FrameRate : MonoBehaviour

{

// Prints framerate.

void Start()

{

print(1.0f / Time.deltaTime);

}

void Update()

{

print(1.0f / Time.deltaTime);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public interface IEnemy

{

// Interface for the Enemy script to specify available methods.

void createWallCannon();

void update();

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public interface IPlayer

{

// Interface for the Player script to specify available methods.

void createPlayer();

void notify();

GameObject getPlayer(string player);

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Login : MonoBehaviour

{

// Declares fields and button to submit field data.

public InputField nameField;

public InputField passwordField;

public Button submitButton;

public void CallLogin() {

// CoRoutine for the LoginPlayer function.

StartCoroutine(LoginPlayer());

}

IEnumerator LoginPlayer()

{

// Checks user input via php script that interfaces with MAMP database.

WWWForm form =new WWWForm();

form.AddField("name", nameField.text);

form.AddField("password", passwordField.text);

WWW www = new WWW("http://localhost/sqlconnect/login.php", form);

yield return www;

if(www.text[0] == '0')

{

DBManager.username =nameField.text;

DBManager.score =int.Parse(www.text.Split('\t')[1]);

UnityEngine.SceneManagement.SceneManager.LoadScene(3);

}

else

{

Debug.Log("User login failed. Error #" + www.text);

}

}

public void VerifyInput()

{

// Requires that the length of Password and Username is above 8 characters.

submitButton.interactable = (nameField.text.Length >=8 && passwordField.text.Length >=8);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

//new Main Menu scrip

public class MainMenu : MonoBehaviour

{

// Declares Text field for player username.

public Text playerDisplay;

private void Start(){

print("Mainmenu has loaded");

// If user is loggedIn then the Players username will be displayed.

if(DBManager.LoggedIn)

{

playerDisplay.text = "Player: "+ DBManager.username;

}

}

public void GoToRegister()

{

// Loads Register Scene

SceneManager.LoadScene(1);

print("Register Scene Loaded");

}

public void GoToLogin()

{

// Loads Login Scene

SceneManager.LoadScene(2);

}

public void GoToGame()

{

// Loads level1 Scene

SceneManager.LoadScene(3);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.Timers;

using System.Diagnostics;

public class Player : MonoBehaviour

{

// Declares Bullet and HealthIndicator GameObject also declares AnimationScript.

// Initializes speed, startJump, midJump, jumpCounter, isGrounded, and disableSKey.

public GameObject bullet;

StickToPlayer AnimationScript;

GameObject HealthIndicator;

public float speed = 0.1f;

public bool startJump = false;

public bool midJump = false;

public int jumpCounter = 0;

public bool isGrounded = false;

public bool disableSKey = false;

public string bulletDirection;

// Start is called before the first frame update

void Start()

{

// Sets player gameobject's default position.

setX(-1.772f);

setY(1.053f);

// Checks if player gameobject has pressed a keyboard button.

userControls();

// Initializes AnimationScript and HealthIndicator gameobjects

// Initializes HealthIndicator and AnimationScript of type StickToPlayer

GameObject AnimationScript = GameObject.Find("MidAirLanceBean");

GameObject HealthIndicator = GameObject.Find("Health");

this.HealthIndicator = HealthIndicator;

this.AnimationScript = AnimationScript.GetComponent<StickToPlayer>();

// Sets AnimationScript player position as default position.

this.AnimationScript.setXPos(-1.772f);

this.AnimationScript.setYPos(1.053f);

}

// Update is called once per frame

void Update()

{

// Triggers the idle animation.

GameObject AnimationScript = GameObject.Find("MidAirLanceBean");

this.AnimationScript = AnimationScript.GetComponent<StickToPlayer>();

this.AnimationScript.setIdle(true);

// Checks if user has initiated a keypress.

userControls();

/\*

If a jump has been initiated then the player will move vertically until

the jumpCounter equals 10.

\*/

if(this.startJump)

{

setY(getY()+.1f);

this.jumpCounter = this.jumpCounter + 1;

if(this.jumpCounter == 10)

{

this.startJump = !this.startJump;

this.jumpCounter = 0;

}

}

// Updates AnimationScript player gameobjects X and Y position.

this.AnimationScript.setXPos(getX());

this.AnimationScript.setYPos(getY());

}

public void userControls()

{

// Initializes AnimationScript GameObject and AnimationScript StickToPlayer type.

GameObject AnimationScript = GameObject.Find("MidAirLanceBean");

this.AnimationScript = AnimationScript.GetComponent<StickToPlayer>();

/\*

space - Start jump animation, triggers idle to false, starts jump bool, and makes the player not grounded.

a - Flips player sprite to the left, idle to false, running animation starts, changes player x position.

s - Idle to false, starts jumping animation, changes players Y position.

d - Idle to false, flips player sprite to the right, sets moving right attribute to true, starts running animation, and changes players X position.

up arrow key - Starts timer for shooting bullets upward, sets bulletDirection attribute to up, creates bullet above player, prints time to execute.

down arrow key - Starts timer for shooting bullets downward, sets bulletDirection attribute to down, creates bullet below player, prints time to execute.

left arrow key - Starts timer for shooting bullets left, sets bulletDirection attribute to left, creates bullet to the left of the player, prints time to execute.

right arrow key - Starts timer for shooting bullets right, sets bulletDirection attribute to right, creates bullet to the right of the player, prints time to execute.

esc - Loads new scene to go back to the mainmenu or continue.

Otherwise set moving right attribute to false and set animation to idle.

\*/

if (Input.GetKeyDown("space") & this.startJump == false && this.isGrounded)

{

this.AnimationScript.setIdle(false);

this.AnimationScript.setJumpingAnim();

this.startJump = true;

this.isGrounded = false;

}

else if (Input.GetKey("a"))

{

this.AnimationScript.setIdle(false);

this.AnimationScript.setFlipX(true);

this.AnimationScript.setRunningAnim();

setX(getX()-(Time.deltaTime \* speed));

}

else if (Input.GetKey("s") && !this.disableSKey)

{

this.AnimationScript.setIdle(false);

this.AnimationScript.setJumpingAnim();

setY(getY()-(Time.deltaTime \* speed));

}

else if (Input.GetKey("d"))

{

this.AnimationScript.setIdle(false);

this.AnimationScript.setFlipX(false);

this.AnimationScript.setMovingRight(true);

this.AnimationScript.setRunningAnim();

setX(getX()+(Time.deltaTime \* speed));

}

else if (Input.GetKey("up"))

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

this.bulletDirection = "up";

Instantiate(bullet, new Vector3(getX(), getY()+.1f, 0), Quaternion.identity);

watch.Stop();

print($"Execution Time: {watch.ElapsedTicks} ticks");

}

else if (Input.GetKey("down"))

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

this.bulletDirection = "down";

Instantiate(bullet, new Vector3(getX(), getY()-.1f, 0), Quaternion.identity);

watch.Stop();

print($"Execution Time: {watch.ElapsedTicks} ticks");

}

else if (Input.GetKey("left"))

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

Instantiate(bullet, new Vector3(getX()-.1f, getY(), 0), Quaternion.identity);

watch.Stop();

print($"Execution Time: {watch.ElapsedTicks} ticks");

}

else if (Input.GetKey("right"))

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

this.bulletDirection = "right";

Instantiate(bullet, new Vector3(getX()+.1f, getY(), 0), Quaternion.identity);

watch.Stop();

print($"Execution Time: {watch.ElapsedTicks} ticks");

}

else if (Input.GetKey("escape"))

{

UnityEngine.SceneManagement.SceneManager.LoadScene(4);

}

else {

this.AnimationScript.setMovingRight(false);

this.AnimationScript.setIdleAnim();

}

}

public void OnCollisionEnter2D(Collision2D coll)

{

// If player collides with the KillLand then he loses health, triggers hit animation, and is moved to the games default position.

// NOT OPERATIONAL - Player doesnt get hurt from colliding with wallCannon - Potentially because wallCannon name doesn't match.

if (coll.gameObject.name == "KillLand" || coll.gameObject.name == "wallCannonEdited")

{

this.AnimationScript.setHitAnim();

Destroy(this.HealthIndicator);

setX(-1.772f);

setY(1.053f);

}

// If player is on land then he can clip through the land

if (coll.gameObject.name == "Land")

{

this.GetComponent<Player>().isGrounded = true;

this.GetComponent<Player>().disableSKey = false;

this.AnimationScript.setGrounded();

this.AnimationScript.setIdleAnim();

}

// If player collides with solid land then he cant hold down the S Key to clip through the floor.

if (coll.gameObject.name == "SolidLand")

{

this.GetComponent<Player>().isGrounded = true;

this.GetComponent<Player>().disableSKey = true;

this.AnimationScript.setGrounded();

this.AnimationScript.setIdleAnim();

}

}

// Setters and getters

// Getter for the bulletDirection

public void setX(float xPos)

{

gameObject.transform.position = new Vector2(xPos, getY());

}

public void setY(float yPos)

{

gameObject.transform.position = new Vector2(getX(), yPos);

}

public float getY()

{

return transform.position.y;

}

public float getX()

{

return transform.position.x;

}

public string getBulletDirection()

{

return this.bulletDirection;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerFactory : AbstractFactory

{

// Declares Player, Sprite gameobjects, and EnemyFactory wallCannon.

public GameObject Player;

public EnemyFactory wallCannon;

public GameObject MidAirLanceBean;

void Start()

{

// Initializes Gameobjects

GameObject Player = GameObject.Find("Player");

this.Player = Player;

GameObject MidAirLanceBean = GameObject.Find("MidAirLanceBean");

this.MidAirLanceBean= MidAirLanceBean;

createPlayer();

}

void Update()

{

notify();

}

// Getter for player.

// Instantiates Player Gameobject.

public GameObject GetPlayer(string player)

{

return Player;

}

public void createPlayer()

{

this.Player = Instantiate(this.Player, new Vector3(-1.772f, 1.053f, 0), Quaternion.identity);

}

public GameObject getPlayer(string player)

{

return this.Player;

}

public void notify()

{

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Registration : MonoBehaviour

{

// Declares user input fields, submition button, and testing variable of type string.

public InputField nameField;

public InputField passwordField;

public string TestUserNameIntegrationTest;

public Button submitButton;

public void CallRegister() {

// Creates CoRoutine for Register function.

StartCoroutine(Register());

}

IEnumerator Register()

{

// Indexes user input via php script that interfaces with MAMP database.

TestUserNameIntegrationTest = "Username Indexed: "+nameField.text+" Password Indexed: "+passwordField.text;

WWWForm form =new WWWForm();

form.AddField("name", nameField.text);

form.AddField("password", passwordField.text);

WWW www = new WWW("http://localhost/sqlconnect/register.php", form);

yield return www;

if(www.text == "0")

{

Debug.Log("User created successfully "+TestUserNameIntegrationTest);

UnityEngine.SceneManagement.SceneManager.LoadScene(0);

}

else

{

Debug.Log("User creation failed.Error #" + www.text);

}

}

public void VerifyInput()

{

// Requires user input for username and password to be at least 8 characters.

submitButton.interactable = (nameField.text.Length >=8 && passwordField.text.Length >=8);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

//new Main Menu scrip

public class resume : MonoBehaviour

{

// Loads scenes for level1, mainmenu, and resume based on user input.

public void GoToResume()

{

SceneManager.LoadScene(4);

}

public void GoToMainmenu()

{

SceneManager.LoadScene(0);

}

public void GoToGame()

{

SceneManager.LoadScene(3);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class StickToPlayer : MonoBehaviour

{

// Declares PlayerAnimation var of type Animator.

Animator PlayerAnimation;

// Initializes state of gameobject player variables.

public bool isGrounded = false;

public bool isIdle = false;

public bool currentIdleState = false;

public bool isMovingRight = false;

void Start()

{

// Initializes playerAnimation.

Animator playerAnimation = GetComponent<Animator>();

this.PlayerAnimation = playerAnimation;

}

void Update()

{

Animator playerAnimation = GetComponent<Animator>();

this.PlayerAnimation = playerAnimation;

}

// Setters for setting position X and Y, is player grounded, is player idle, and if player is moving right.

public void setXPos(float xPos)

{

transform.position = new Vector2(xPos, transform.position.y);

}

public void setYPos(float yPos)

{

transform.position = new Vector2(transform.position.x, yPos);

}

public void setGrounded()

{

this.isMovingRight = false;

this.isGrounded = !this.isGrounded;

}

public void setIdle(bool isIdle)

{

this.isIdle = isIdle;

this.currentIdleState = isIdle;

}

public void setMovingRight(bool isMovingRight)

{

this.isMovingRight = isMovingRight;

}

public bool getMovingRight()

{

return this.isMovingRight;

}

// Flips players sprint horizontally.

public void setFlipX(bool isFlip)

{

SpriteRenderer mySpriteRenderer = GetComponent<SpriteRenderer>();

mySpriteRenderer.flipX = isFlip;

}

// Triggers for running, jumping, idle, and player gameobject being hit.

public void setRunningAnim()

{

this.PlayerAnimation.SetTrigger("running");

}

public void setJumpingAnim()

{

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

this.PlayerAnimation.SetTrigger("jumping");

watch.Stop();

print($"Execution Time: {watch.ElapsedTicks} ticks");

}

public void setIdleAnim()

{

this.PlayerAnimation.SetTrigger("idle");

}

public void setHitAnim()

{

this.PlayerAnimation.SetTrigger("hit");

}

}