Software Design & Development 2021

MAJOR WORK

Mr El-Miski

Arnold Quiocho

Table of Contents

Defining and understanding the problem

Defining the problem

Description of the problem

The client desires fairly vague specifications to allow flexibility and self-expression through areas which the coder excels to achieve a product that displays their full potential:

- Design and Develop (hence Software Design and Development) a piece of software through all the stages from conceptualization to implementation.
- Follow a software development approach like Agile, structured (waterfall), etc.
- Document all stages of the Software Development cycle undertaken throughout the project.
- Use Object Oriented Programming in Java or other OOP languages in the implementation (during coding).
- Test, evaluate and maintain the implemented software.

Reflecting on previous major projects reveals that many software major works are exclusively video games in greenfoot, and since many students (and even the client) are familiar with video games, it could make the software design process simpler due to their exposure to this medium.

The project requires a level of complexity that goes over my current skill level to show an improvement in skill and learning. A problem is that the assignment must be finished in Term 3, and in-between the terms assignments are given which can hinder work on the software solution.

Action Side-scroller games are a dried-out genre and I wish to bring new life to it by following what succeeded in previous video game titles. But even if I were unable to achieve newer or more interesting concepts and mechanics, I could still draw inspiration from previous titles, taking what worked from them and applying them to my solution to make mine as effective as their solutions.

Preliminary Timeline (Gantt chart)

PROPOSED

STAGE	ACTIVITY	TERM 4										HOLIDAY						TERM 1										HOLIDAY		TERM 2										HOLIDAY		TERM 3
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	1	2	1	2	3	4	5	6	7	8	9	10	1	2	1
Defining and Understanding the Problem	Defining the Problem	🟩																																								
	Issues Relevant		🟩																																							
	Design Specifications			🟩																																						
	System Documentation				🟩	🟩																																				
	Communication Issues						🟩																																			
	Quality Assurance							🟩																																		
Planning and Designing Software Solutions	Standard Algorithms								🟩																																	
	Custom-Designed Logic									🟩	🟩																															
	Standard Modules											🟩																														
	Documentation of overall Solution												🟩	🟩																												
	Interface Design														🟩																											
	Factors of Language Used														🟩																						🟩					
Implementation of Software Solutions	Implementation of Design															🟩		🟩		🟩		🟩		🟩		🟩		🟩	🟩	🟩		🟩		🟩		🟩		🟩		🟩		
	Techniques in Well Written Code																🟩																						🟩			
	Documentation of an Overall Solution																	🟩																						🟩		
	Hardware Environment																		🟩																					🟩		
Testing and Evaluation	Testing the Software Solution																										🟩												🟩		🟩	
	Post Implementation Review																																								🟩	
Maintaining Software Solutions	Modifying Code to meet Requirements																																								🟩	

ACTUAL

STAGE	ACTIVITY	TERM 4										HOLIDAY						TERM 1										HOLIDAY		TERM 2										HOLIDAY		TERM 3
		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	1	2	1	2	3	4	5	6	7	8	9	10	1	2	1
Defining and Understanding the Problem	Defining the Problem		🟦																																							
	Issues Relevant			🟦																																						
	Design Specifications				🟦																																					
	System Documentation																		🟦																				🟦			
	Communication Issues																		🟦																				🟦			
	Quality Assurance																		🟦																				🟦			
Planning and Designing Software Solutions	Standard Algorithms																			🟦	🟦	🟦																	🟦			
	Custom-Designed Logic																						🟦	🟦		🟦	🟦												🟦			
	Standard Modules																											🟦														
	Documentation of overall Solution																												🟦													
	Interface Design																												🟦										🟦			
	Factors of Language Used																													🟦							🟦	🟦				
Implementation of Software Solutions	Implementation of Design															🟦	🟦											🟦	🟦	🟦	🟦	🟦	🟦				🟦	🟦		🟦	🟦	
	Techniques in Well Written Code																																				🟦	🟦				
	Documentation of an Overall Solution																																			🟦			🟦			
	Hardware Environment																																									
Testing and Evaluation	Testing the Software Solution																																								🟦	
	Post Implementation Review																																								🟦	
Maintaining Software Solutions	Modifying Code to meet Requirements																																								🟦	

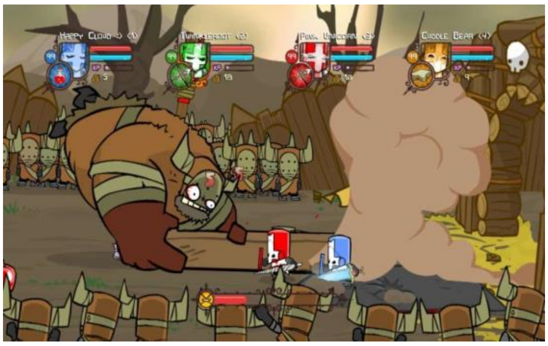
Proposed	🟩
Actual	🟦

Idea Generation

Based on games I have previously played, I have chosen a few game which are possible for me to replicate the basic mechanics, graphics, and gameplay. The games I have chosen are simple enough for me to easily create myself, while still engaging enough to be fun even if it's quite simple.

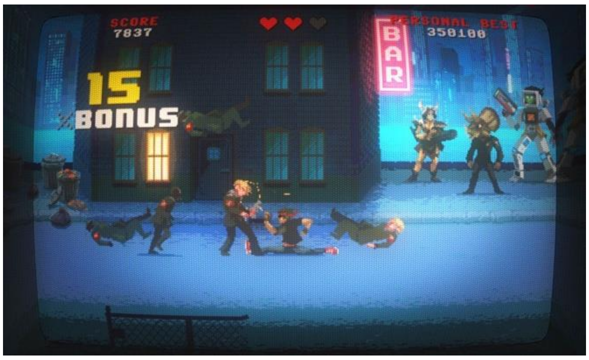
- **Castle Crashers**

A 2D beat em' up set in a fantasy world by The Behemoth. It's a game that continuously progresses in a similar fashion to the original Mario games, where you can't go backwards. It has multiple ways to deal damage including melee, magic, air attacks, etc. Attacks can be chained into combos to give the player more control of the action. The game is mission based and sets you on the path of a light-hearted depiction of war and princess rescuing.



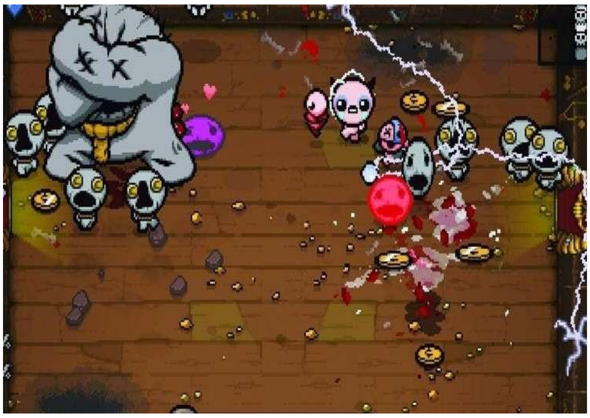
- **Kung Fury: Street Rage**

An arcade 2D beat em' up following Kung Fury in a story-based mode or in an endless survival mode. Due to the game being designed for mobile, the gameplay is quite simple in controls as there is only one way to attack, by tapping the screen on the left or right side to do a simple hit in either direction. However, the enemies have different amounts of hits before death, and some enemies switch it up by teleporting after being attacked, having their own combos which the players must learn to progress and get a higher score with.



- **The Binding of Isaac**

A top-down roguelike featuring randomized pathways and dungeons. This game has fairly simple controls, having only direction inputs for movement and shooting in each direction, however, the hook of the game is that the randomized rooms have chances to drop different items to change, improve or even disadvantage the player in the way it moves, shoots or more. The game has normal enemies in each room but frequently encounter bosses to reach the next level.



Idea Selection

For my project I have decided to create a game more similar to Castle Crashers. Due to the games fairly traditional gameplay and mechanics, I believe that it is possible to combine my current knowledge of coding and Object-Oriented Programming with the knowledge of internet tutorials from people who have worked in this medium and genre of games. However, I will be taking inspiration from Kung Fury's endless mode, which will make it a bit easier to implement as it requires no story or extended mechanics outside of the gameplay, menu, and end screen.

Define Boundaries of Software

In the creation of my game, there are many boundaries in the gameplay and core mechanics.

- The ability to string combos and attacks requires more than 1 way to attack
- At least one character (the main character) must be able to be controlled by the player
- An endless mode may require implementation of enemy spawning, and increased frequency for higher difficulty levels
- A beat em' up video game requires the enemies and player to have their own health, damage and attacks
- Requires graphics to telegraph player and enemy attacks
- Requires hitboxes for the enemy and character for dealing damage
- Endless mode requires score and scoreboard for replay ability
- A main menu for accessing the game and other difficulties
- A graphics select option to enable use of lower graphics for low end machines
- Keyboard inputs must follow WASD movement
- Mouse input only requires left click for attacking/menu select
- No control of what kind of keyboard, mouse or speaker is used
- Support for different Operating Systems (Mac, Linux, Windows, etc)
- Support for x32 or x64 bit Operating Systems
- Colour-blind support for different colours in UI and other graphics
- Gore disable for censorship

Issues relevant to a proposed solution

Can existing solutions be used?

It is possible to use existing solutions within the project, as many ideas that exist commonly take inspiration and concepts from solutions that have already come before it. This is relevant to my project as my game idea already exists as a solution in the game "Castle Crashers". However, there are many issues to be considered when thinking of the social and ethical issues.

Copyright-law is an important consideration. Existing solutions must be either open source or public domain, which eliminates the risk of copyright. And existing solutions cannot simply be copied fully, and the code must be altered to avoid plagiarism. Code may sometimes be open to all, but it may not be obviously clear whether the code is covered by copyright laws, so contacting the developer of the code to clarify is always the safest option to take.

However, looking at the assignment's description for the game's development :

- Design and develop a Software project from an idea to a fully implemented software program.
- Document your development process following the Software Development Cycle proscribed by the syllabus.
- Follow OOP principles in your implementation

It reveals that the code created by me must be completely original. Simply copying code from other solutions will not be enough to meet the requirements for this assignment, so existing solutions must simply be used as inspiration or as a steppingstone or scaffold in showing how to build your code, with the rest of the work being done by yourself.

Comparable existing solutions

There are many existing solutions which can be found within the genre of hack and slash games. Games like **Rayman Legends** and **Towerfall Ascension** are other games within the genre shared by **Castle Crashers** which both share ideas and concepts, and their own unique ones.

Rayman Legends

Rayman Legends is a 2D side scroller made by Ubisoft. It has fluid movement and platforming which ties well with the games fairly simple combat mechanics. It has 3 playable characters, co-op, and 6 stages, being a direct sequel to the game Rayman Origins, it follows the adventure of Rayman and friends. The game has 2D graphics that are layered on top of each other which gives a 3D effect to the games graphics even though they are all drawn graphics.

It shares gameplay mechanics with Castle Crashers in that it keeps moving forward, with movement and combat, however Castle Crashers combat is more fleshed out, rather than focussing on platforming like Rayman Legends.

Towerfall Ascension

Towerfall Ascension is another 2D platform fighter, however the game is focussed on arenas or stages rather than long moving levels. The fast combat and movement are used to fight different kinds of enemies with different methods of attacking and evading the player. Rather than having 2-dimensional attack patterns for the player, the player is allowed to attack in any direction with a bow and arrow.

It features 4 player co-op multiplayer with different characters that change the visual style of the character and gameplay. The gameplay is focussed on challenging levels rather than story, offering a hard mode for more seasoned players. The gameplay is more unique compared to castle crashers, but in the same vein as the game Kung Fury : Streets of Rage, it has combat in still arenas where enemies continue to spawn to fight the player.

While these games share the genre of 2D combat and arena combat like Castle Crashers and Kung Fury, they all give their own unique spin on the genre with their own gameplay styles and schemes. Although they are all great games, my vision for my game shares more similarities to the game Castle Crashers, however both Rayman Legends and Towerfall Ascension offer their own unique concepts which I can take inspiration from and combine with my own idea.

Determine existing solutions

The following are existing solutions that are relevant to the concepts in my project.

Title	**Brackeys**
Link	**https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA**
Description	Channel focussing on concepts of game design, coding design and Unity C# game development tutorials. His tutorials range from basic game mechanics coding to niche tutorials like specific game engine graphics effects and game mechanics. His videos also talk about game design discussion which can help in locking down the important game mechanics that should be focussed on.
Social and Ethical Considerations	Ethical Considerations – Following a tutorial on his channel may most likely lead you to copying parts, or even most of his code. Although it works, or may fit into your own code, you must still consider whether taking this code is not stealing as it could be his intellectual property. However, in the description of all of his videos, he states **"All content by Brackeys is 100% free. We believe that education should be available for everyone."**. This allows students like me to use code that is the same as the tutorial with no worry. Social Considerations - Due to the nature of the solution being tutorials, code that may impede on social boundaries can choose to not be added to your own code. However, due to the tutorials being made for a broad audience, it may not include benefits or help for minorities like people with disabilities.
Customisation Options for existing solutions	Code made and used from the tutorial can always be repurposed or changed to benefit the users own code and purposes. With knowledge, code can always be swapped around, changed, or modified. The range of tutorials on the channel is effective in allowing the coder to use whichever pieces of code they need, and any other code or concepts can be used as inspiration.
Cost-Effectiveness	YouTube videos are completely free, and the content within the tutorials of Brackeys are free as well which makes these tutorials extremely cost-effective. The only requirements are to have a working computer and internet connection which is easily accessible due to Australia being a first world country, either through already having both or using the facilities provided by the school.
Licensing Considerations	There are few to no licensing considerations through Brackeys videos. In the description it says that all content is 100% free, meaning that all content like code and concepts can be used with no copyright issues.

Select and Justify Software Development Approach

The software development approach is an important aspect of the Software Development Cycle, as it outlines the way in which the developers will be planning the construction of their solution, with the rest of the cycle being built around the chosen approach. The approach is often chosen depending on the needs of the user and the abilities of the developer, but most importantly caters to the user. There are a wide range of approaches:

- RAD Approach (Rapid Action Development)
- End-User Development Approach
- Prototyping Approach
- Structured Approach

The RAD approach is a very fast development approach, which requires the developer to already have a strong understanding of the user requirements for the software and the projects scope and limitations. A part of why it's fast is because it allows you to the documentation part of the software development cycle. This would be an ineffective development approach as documentation is a requirement of the client of my software solution.

The End-User Development approach is an approach which requires the end-user itself to create the solution. This solution is also very quick and effective, as the end-user knows what the requirements for the software are. This often comes with very little to no documentation as the end-user knows how to use their own code, and therefore doesn't need to understand how to use it. This is an ineffective development approach for the solution as the client requires not only documentation of the solution, but also maintenance, because no documentation makes it hard for other developers other than the user to maintain the software after implementation.

The Prototyping Approach requires the software solution to constantly be tested throughout all stages of its implementation. This means that beta testers are often brought in to test each build of the software as its released and give feedback on what to improve or whether the software meets the user requirements. Prototyping continues until the software meets the user requirements. This is a long development process however and requiring the software to be tested means that the user must constantly test it. Because the client has to manage the software of other developers, there would be very few opportunities for the client to test the software, making this approach unsuitable for this project.

The Structured Approach is the most reliable and formal development approach. It requires being thorough and detailed with each step of the development process, spending a fairly large amount of time on each step and documenting it, creating a complete record that can serve as a reference for the subsequent stages of the development process. This approach follows the requirements of the client, having documentation in all of the stages of the approach, and includes all stages of the Software Development Cycle. The Structured Approach is the most appropriate approach for this project.

Design specifications

List of specifications of the system

The specifications for the system must follow the client's requirements as well, which are:

- Design and develop a Software project from an idea to a fully implemented software program.
- Document your development process following the Software Development Cycle proscribed by the syllabus.
- Follow OOP principles in your implementation.

Following the requirements for the client, and the concepts and rules of my game, the specifications of the system would be:

- At least a single stage or level.
- An object-oriented programming language.
- Support for both Mac and Windows.
- Combat with combos that are fun and easy to do.
- Basic left to right movement and jumping.
- A health and damage system.
- A pause menu that freezes the game.
- A start menu to start either the story mode or endless mode.
- An ending with high score.
- A settings menu with ability to change graphics settings, sound volume, blood and music volume.
- Interactive UI.
- Proper hitboxes.
- Time based attacks for the player and enemy.
- At least two characters.
- Original art, graphics and animation.
- Different enemy types to force the player into different strategies.
- A boss battle.
- An endless mode that shows online high scores.
- Audio through sound effects and music.

Basing the game on the specifications of the Unity software and the Castle Crashers specifications, to run the game smoothy the computer should have:

- Windows 7 or higher.
- Intel Core 2 Duo or higher.
- 2 – 6 GB of RAM.
- 256 MB video card (GeForce 7900 GT equivalent or higher).
- Sound Card.
- 500 MB of disk space.

Developer's Perspective

The developer of the software has a lot of backdoor issues to consider when creating the software. There must be proper planning before actually coding to ensure that there are no setbacks or stops during this phase of development. During development, the developer must consider:

- Data structures undertaken.
- Documentation Methods (Flowcharts, IPO diagrams, Data Flow diagrams, etc.).
- Programming Practices for appropriate Quality Assurance.

Data structures are important to consider, because of the high score system that will be in the game. In Unity, scripts allow the creation of lists which store data, allowing the storing of player scores which can be outputted and displayed as text after the game is finished.

Attacking enemies also requires a data structure, as enemies that are hit in a hitbox are placed in an array to make sure that only the enemies in the hitbox are attacked rather than all enemies on the screen. This is because Unity's collision modules place the hit boxed enemies in an array to apply the functions for things like player damage to enemy to every collided enemy at once.

The developer must also follow good programming practices like internal documentation techniques (white space, indentation, etc) to improve the readability of your code which in turn improves the maintenance of your code in the future. Developers will often encounter errors which they have to remedy and solve, so good documentation makes it easier to find these issues because readability is good. Error detection techniques like debugging, peer checking, and desk checking are important for the developer to undertake to solve issues of logic errors or runtime errors that they may encounter in the implementation of the software solution.

Algorithm specifications

Algorithms are an important part of game development, so they must always follow the rules of the algorithm specifications to ensure that their readability and quality is always consistent. Creating an algorithm that is easy to visualise and understand is imperative for the developer to understand it themselves and put it into practical use once they start to code.

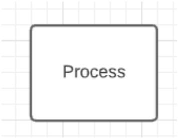
Algorithms help to show the inputs and steps taken to create an output, usually having a start and an end. Creating an algorithm in Pseudocode and in flowcharts should both be understood hand in hand as they both represent the same thing.

To first understand algorithms, it's important to understand that algorithms can be shown in Pseudocode and in flowchart form. The symbols of the flowchart are all distinct to ensure that they are not confusing and are easy to understand. The symbols are:

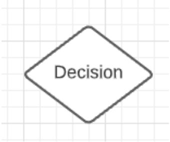
The algorithm specifications look as follows:

These lines with arrows indicate the path the process follows.

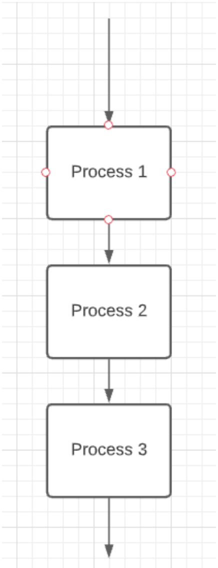
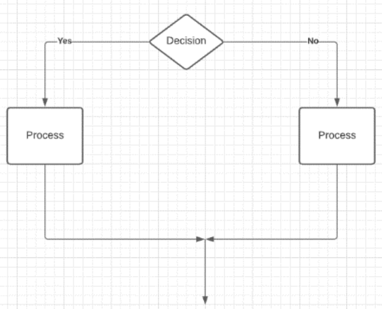
This symbol shows the process.

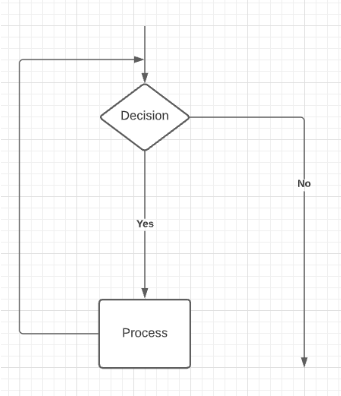
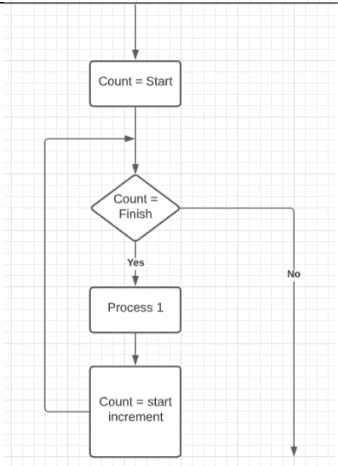
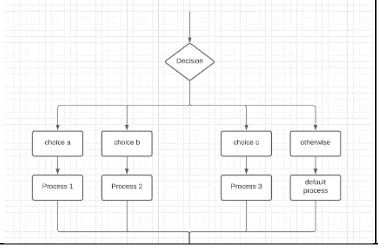


This symbol shows a decision in the process. Each corner can be a separate path.



<u>The algorithm specifications of flowcharts and pseudocode are as follows:</u>

Algorithm	**Flow Chart**	**Pseudocode**
Sequencing		. . . Process 1 Process 2 Process 3 . . .
Selection		Pseudocode . . IF DecisionTrue Process 1 ELSE Process 2 ENDIF . .

Iteration		. . . WHILE DecisionTrue Process ENDWHILE . . .
Counting Loop	 Count = Start Count = Finish Yes No Process 1 Count = start increment	FOR count = Start TO finish STEP increment Process 1 NEXT Count
Multiway Selection	 Decision choice a choice b choice c otherwise Process 1 Process 2 Process 3 default process	CASEWHERE Something evaluatesto choice a : Process 1 choice b : Process 2 choice c : Process 3 OTHERWISE: default_PROCESS ENDCASE

The pseudocode specifications are as follows:

For the start of each procedure or routine:

```
    BEGIN name

    END name
```

For binary selection:

```
    IF condition THEN

        Statements

    ELSE

        Statements

    ENDIF
```

for multi-way selection

```
    CASEWHERE expression evaluates to
        A: process A
        B: process B
    …………………
    OTHERWISE: process ….
    ENDCASE
```

For pre-test repetition

```
    WHILE condition
        Statements
    ENDWHILE
```

For post-test repetition

```
    REPEAT
        Statements
    UNTIL condition
```

For FOR / NEXT loop

```
    FOR variable = start TO finish STEP increment
        Statements
    NEXT variable
```

User's Perspective

Considerations

The user's perspective is much different to the developer's perspective. Since they are not coding or developing it, their perspective consists of the final product, what it will do, how everything is delivered, how it looks, etc. It is important to consider the users perspective on top of the developers. To deliver a satisfactory final product for both the user and the developer, communication is key in all stages of development. Communication should involve in person meetings, questionnaires, emails, etc.

The user's perspective goes beyond the software and coding and lies on how the user will interact with the system, if the system is ergonomically compliant and if the user's environment is suited to the software, as in what he is using to operate it.

The menu or interface is a crucial part of development as it is how the user will interact with the software. Things that must be considered include where buttons or UI elements will be placed. It should not be too hard to find any buttons that are important, and it should not be visually jarring to look at as well. The design of the UI, like visual and audio elements are important in improving the user's experience.

Ergonomics is another important consideration. Ergonomics is the consideration of the human's physical capabilities when interacting with the software. Things like what buttons would be pressed and how hardware will be used to interact with the system. Gameplay wise, movement will be used using the international standard WASD controls, as they are close together and easy for the user to press each key. UI interaction will be done with the mouse, as navigating the UI will be easy with mouse movement and clicking. All of these are common in most software and will make it easier for the user as they would have used other software which have similar controls.

The user's environment involves what is being used to run the software, and the hardware involved as well. Having as little hardware required as possible, or at least the standard should be good to making the software aid to the user's environment. Hardware specifications are included in this. A game that is too hard to run would go against the user's environment as they may not have access to such specifications. Access to external hardware is a consideration too, but it is safe to assume that the user has access to a computer, a keyboard, and a monitor. Having the software require external hardware like specialised controllers or headsets goes against the user experience as it requires them to get this hardware just to use it.

System documentation

IPO

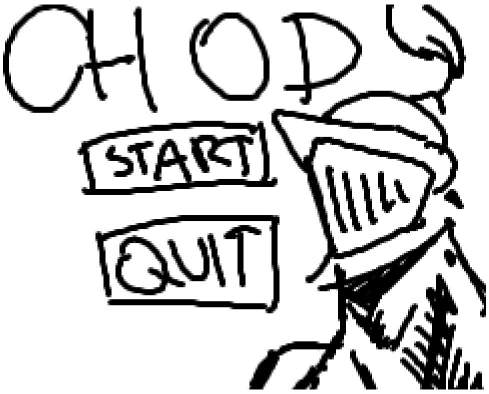
CharacterMovement Class

Input	Process	Output
W key is pressed.	• Make the character jump. • Play jump sound effect. • Play jump graphic.	• Character will jump. • Hitbox will move with character.
A key is pressed.	• Play left movement graphic. • Play movement sound effect.	• Character will move left. • Hitbox will move left with character.
D key is pressed.	• Play right movement graphic. • Play movement sound effect.	• Character will move right. • Hitbox will move right with character.

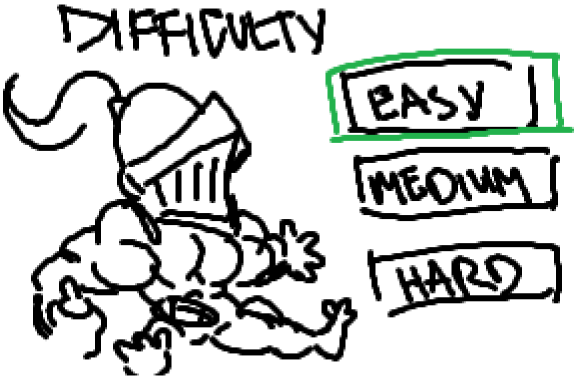
CharacterAttack Class

Input	Process	Output
Left Click is pressed.	• Play Sword Swing graphic. • Play slash sound effect. • Create attack hitbox on sword graphic. • Create blood graphic.	• Character will attack.
Left Click is pressed (while in air).	• Play Jump Swing graphic. • Play slash sound effect. • Create large attack hitbox on sword graphic. • Create blood graphic.	• Character will air attack.

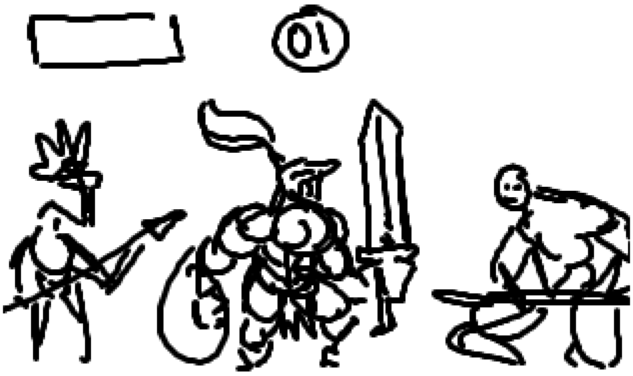
Storyboard



This is the main menu of the game. It will have the main character with an idle animation, with 2 buttons, the start, and the quit button.



Upon clicking start, it will ask you for the difficulty you wish to play on. There will be an easy, medium, and hard difficulty. Each difficulty will have a different graphic to show how hard it will be.



This is the gameplay. There will be a timer on the top showing how much time you have left. Different enemy types will come from either side of the screen. The main character will be playable at the center. Your healthbar is on the top left of the screen.

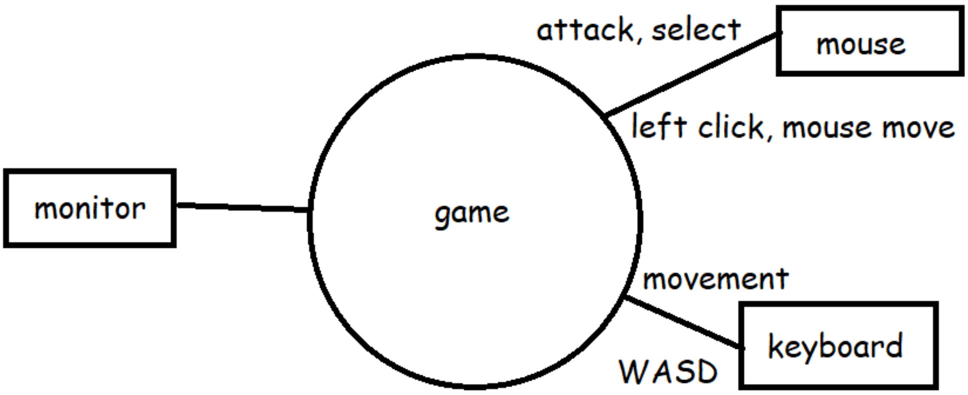
After the timer is over, you will reach the victory screen, and a graphic animation will play.

SCOREBOARD

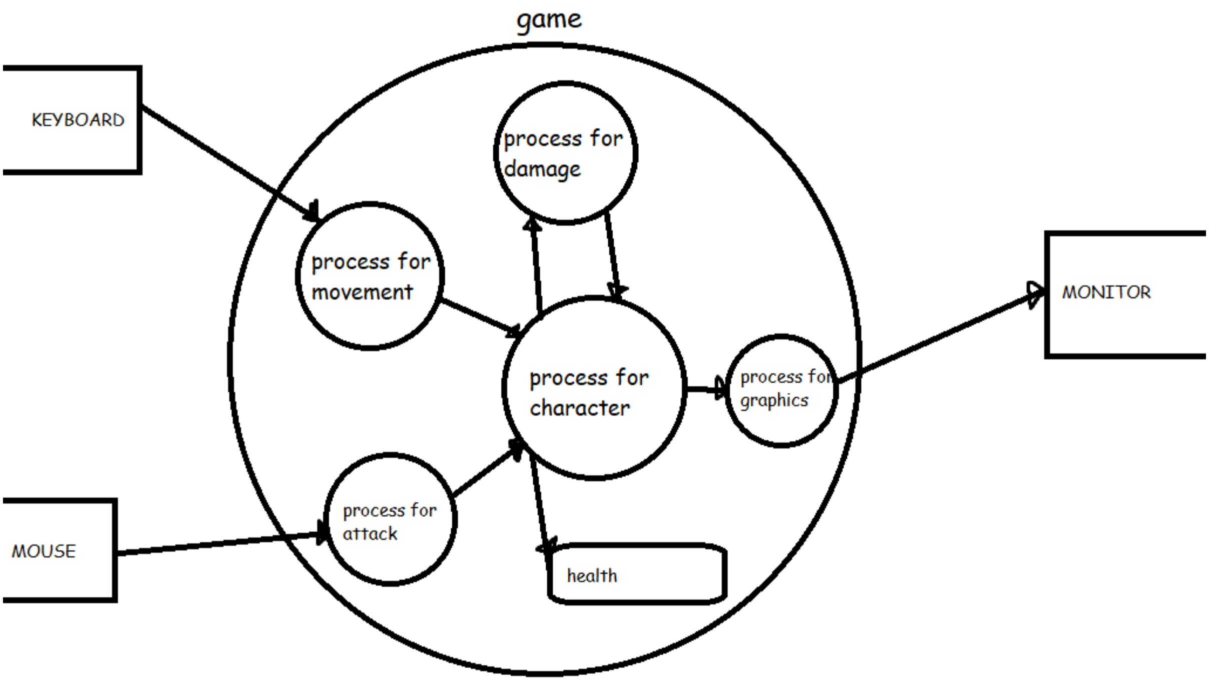
After the victory animation, the scoreboard screen will fade in, showing your score and the top scores on the score board. The main character will have an idle animation on the left of the screen.

DFD

Game DFD Level 1



Game DFD Level 1.1



Data Dictionary

Identifier	Data Type	Scope	Description	Example
HealthMax	Integer	Private, object level	This integer holds the max health of the character. It so that if the game resets, it returns to this value which should be the original health.	500
HealthCurrent	Integer	Public, object level	The current health of the character during gameplay. Can be modified as damage is taken.	278
Damage	Integer	Private, object level	The damage the sword does to enemies.	100
Difficulty	String	Public, object level	The difficulty chosen to play.	“Easy”
Score	Integer	Public, object level	The score of the player.	1000
Name	String	Private, Object level	The name inputted to identify the player on the scoreboard.	“Freddy Freaker”
Countdown	Integer	Private, object level	The countdown until the level is over.	60
End	Boolean	Public, object level	Chosen if alive is true or false.	False

Structure Diagram

Communication Issues between client and developer

Issues can arise between communication for the client and the developer. However, both have made a commitment to the project therefore must make time to ensure that the software is satisfactory for both parties. As a developer, to make sure they understand the client, they must:

- Schedule daily meetings with the client. They must always be ready beforehand with questions about the software and their requirements for it.
- Give daily updates on how the software is going through emails or calls.
- Create prototypes daily to send to the client to give them hands on experience with how the software is currently working.
- Have a collaboration chatroom/document in which at any time they could add their own ideas or concerns about the project.

Having common and daily interaction between the client and the developer allows both to get to know each other, and eventually begin to see each other's perspective more clearly. Multiple kinds of interaction mean that if any scheduled meeting or call had to be cancelled, it would not be much of a worry because either interaction is already so common, or there are other ways of communicating that make up for it. It is good to share the plans of the software with the client, so they understand the developers time plan and make appropriate requests and requirements.

Quality Assurance

Throughout the development it is important to make sure that it is always working as intended, at least in the current time frame. As parts of the software are developed and completed, the relating requirements (the requirements of the software previously stated) will be compared to see if they follow them. If requirements aren't satisfactory, then the software will continue to be worked on until they meet them.

Quality assurance is also important for the user, so prototypes that are developed and sent will be evaluated by the user to see if they fit their needs as well. Through this we can see if the software can run on the client's hardware which is an important consideration.

Planning & Designing Software Solutions

Standard Algorithms

Standard algorithms are the most common algorithms used in code. It is important to identify which standard algorithms will be used in the project, and these algorithms can easily be chosen from the set of standard algorithms specified by the SDD course. The standard algorithms relevant to the project are:

- Random Number Generation
- Random Access Files
- String Extracting
- String Insertion
- Insertion Sort

Random number generation is an important standard algorithm. Many languages have a class which can generate random integers instantly, with a range able to be set to generate a random number within a range. While the integers may appear as random, they are known as pseudo random, because machines are unable to actually do things at random and use a mathematical formula to output an integer that appears at least to us as random. This algorithm is commonly used in games to create a sense of chance like in critical attacks or make NPC spawning seem natural as they spawn in an order-less fashion.

Random Access Files are a list of files or text that are all the same length. For example, a list of names would have a limit of 20 characters, the names being places like RoronoaZoro………, MonkeyDLuffy…….., meaning that if you were to access the file and look for the second name in the list, you would only have to go to the 20th letter in the list, which would be the second file since all the files would be the same length. In files or text that are in order, this is effective in finding them quickly without having to start from the beginning of the list.

String processing reads a string of text as an array of characters. In string extraction, a copy is made of the original string without affecting the original string in any way. The start and end for the extraction of the string can be chosen so that only part of the string can be taken. Extraction is important so that a copy of the original can still be found in case issues with the altered version are encountered. In games, this is used when things like settings are changed, because the original settings can be returned to by creating another copy of the original settings to replace the current one.

String Insertion reads a string of text as an array like in extraction. However, it changes the current string and inserts a new string in a specified location in the string. String insertion separates the string into two parts in the section where the new string will be placed. It concatenates it (joins the two strings), placing the new string between the previous split string, creating a new and modified string. String insertion is used commonly in string files used as storage, where an input of things like names or stats are inserted into extracted original files for the storage.

Insertion sorting is a way to arrange the date into an order and is effective when new items have to be added to the list that is already sorted. It sorts it by looking through the list until it finds an item or integer that is a larger value than it, and places it in that spot. In lists of integers, this is most effective. In games this is most often used in the leaderboard in which a new score is made after a game, and the insertion sort algorithm keeps searching up from the bottom of the list until it finds a number larger than it, or until it reaches the end of the list and places the score there.

Custom Designed Logic Used in Software Solutions

This is pseudocode designed to help in the construction and the implementation of processes in the actual code. This helps make the implantation process to become much faster because it allows us to block out the processes and variables that would be required for the game.

Algorithms

PLAYER ALGORITHM

```
START PlayerAlgorithm(CollisionAlgorithm,MovementAlgorithm,JumpingAlgorithm,PlayerHealthAlgorithm)
    WHILE PlayerHealth > 0
        CollisionAlgorithm
        MovementAlgorithm
        JumpingAlgorithm
        PlayerHealthAlgorithm
        IF Player LeftClick THEN
            AttackAlgorithm
        END IF
    END WHILE
END PlayerAlgorithm
```

COLLISION ALGORITHM

```
START CollisionAlgorithm(PlayerHealth,EnemyCollision)
    PlayerTouch = Player intercepting object
    WHILE PlayerHealth > 0
        IF PlayerTouch Ground THEN
            Player(X) =/= < Ground(X)
        END IF
        IF PlayerTouch Enemy THEN
            EnemyCollision = TRUE
        END IF
        IF PlayerTouch HealthItem THEN
            HealthCollision = TRUE
        END IF
    END WHILE
END CollisionAlgorithm
```

MOVEMENT ALGORITHM

```
START MovementAlgorithm(PlayerHealth,KeyInput,PlayerPositionX)
```

```
    WHILE PlayerHealth > 0
        IF KeyInput = LeftArrow THEN
            WHILE KeyInput =LeftArrow
                PlayerPositionX = PlayerPositionX – 1
            ENDWHILE
        END IF
        IF KeyInput = RightArrow THEN
            WHILE KeyInput=RightArrow
                PlayerPositionX = PlayerPosition + 1
            ENDWHILE
    ENDWHILE
END MovementAlgorithm
```

JUMPING ALGORITHM

```
START JumpingAlgorithm(GroundedAlgorithm, JumpHeight)
    JumpHeight = 5

    If CollisionAlgorithm = TRUE AND KeyInput = Spacebar THEN
        PlayerPositionY = PlayerPositionY + JumpHeight
END JumpingAlgorithm
```

PLAYER HEALTH ALGORITHM

```
START PlayerHealthAlgorithm(EnemyCollision,HealthCollision)
    MaxHealth = 800
    PlayerHealth = MaxHealth
    WHILE PlayerHealth > 0
        IF EnemyCollision = TRUE THEN
            PlayerHealth = PlayerHealth – 50
            EnemyCollision = FALSE
        END IF
        IF HealthCollision = TRUE THEN
            PlayerHealth = PlayerHealth = 100
            HealthCollision = FALSE
        END IF
    ENDWHILE
    IF PlayerHealth >= MaxHealth THEN
```

```
        PlayerHealth = MaxHealth
    END IF
    If PlayerHealth <= 0 THEN
        PlayerHealth = 0
    END IF
END PlayerHealthAlgorithm
```

CHARACTER SELECT ALGORITHM

```
START CharacterSelect
    Public int Character
    IF Button1 Pressed THEN
        Character = 1
        Load Arena1
    ELSE IF Button2 Pressed THEN
        Character = 2
        Load Arena2
    ELSE IF QuitButton Pressed THEN
        QuitGame
END CharacterSelect
```

ATTACK ALGORITHM

```
START Attack(EnemyHealth)
    Damage = 100
    AttackRange = 50
    Enemy[Array] = EnemiesHit in AttackRange
        IF EnemiesHit THEN
            EnemyHealth = EnemyHealth – Damage
        END IF
END Attack
```

TAKE DAMAGE ALGORITHM

```
START TakeDamage(PlayerHealth,EnemyDamage,EnemyRange)
    IF Player in EnemyRange THEN
        PlayerHealth = PlayerHealth – EnemyDamage
    END IF
END TakeDamage
```

PLAYER HEALTH LIMIT ALGORITHM

```
START HealthLimit(CurrentHealth)
    Dead = False
    IF CurrenHealth <= 0 THEN
        CurrentHealth = 0
        Dead = True
    END IF
END HealthLimit
```

Test Data

Test data is an important part to ensure the functionality of the software solution. Throughout code implementation, testing the algorithms and variables used can help not only ensure that there are no logic errors, but also set boundaries for the effective range of what the variables can be. For example, in my software solution for the spawning algorithm, I effectively chose an appropriate spawn rate and change rate by changing the code while having the game running (which Unity allows). This made me find the most appropriate and fun spawn rate for the enemies that come from each side of the screen.

Another variable tested frequently during gameplay is the stats for each character. Balancing things like player health, damage, speed, jump height, and vice versa for the enemy stats allows me to balance whether the games mechanics are fair and fun to play. It allowed me to balance the game in a way where the player feels significantly stronger than the enemies, while not making them too weak or hard to beat.

Desk Checking

This desk check finds out whether the health functions detects whether the player is damaged, and sets the bool **Dead** to true, while also keeping the health at 0, stopping it from going under.

Health	**isDead**	**Health <= 0**	**Damaged**
100	“False”	“False”	20
80	“False”	“False”	20
60	“False”	“False”	20
40	“False”	“False”	20
20	“False	“False”	20
0	“True”	“True”	20
0	“True”	“True”	20
0			

This is the spawner desk check to see whether the spawn rate of change works. For sake of time, the time rate of change is every 2 seconds, with the rest being their default values. The desk check will only go up to 30 seconds.

STime (s)	minusTime (s)	changeInTime (s)	changeRate (s)	Current Time (s)
10	0	0.1	2	0
				1

	0.1			2
9.9				3
				4
	0.2			5
9.7				6
				7
	0.3			8
9.4				9
				10
	0.4			11
9				12
				13
	0.5			14
8.5				15
				16
	0.6			17
7.9				18
				19
	0.7			20
7.2				21
				22
	0.8			23
6.4				24
				25
	0.9			26
5.5				27
				28
	1			29
4.5				30

Standard Modules used in software solutions

There were many standard modules provided by the Unity program. It provides modules for basic game mechanics like collision, graphic transform, sprite animators, etc. This is to save time coding rather than creating these basic parts of the code each time you create a game. This allows more time to be spent on more important parts of the game.

The gameObject module is the most commonly used module in the Unity game engine. It is the base class for all entities in your Unity scene. All entities in a game like the floor, the character, projectiles, etc. are all gameObject modules. This module allows you to modify the entity in the code, allowing you to track its current position, change its scale, and even change and modify the modules attached to it.

The collision2D module is a very commonly used module in the software solution that adds physics and collisions to game objects for you. It saves the time of coding collisions between each and every object, and easily allows collision between multiple objects with the collision2D module applied to them. This module is added to all gameObjects that require collision, leaving only specific collision checkers like the Grounded check to be coded by the developer.

The Rigidbody2D module is another widely used module that applies adjustable physics properties to graphics and game objects. It applies gravity, mass, drag, and other adjustable properties to game objects allowing different properties between different objects which makes it versatile in creating a variety of character or object types that are interactive with the physics in the game engine. This module is mostly only used on interactive entities however, so static objects like the ground or other terrain should not use this module.

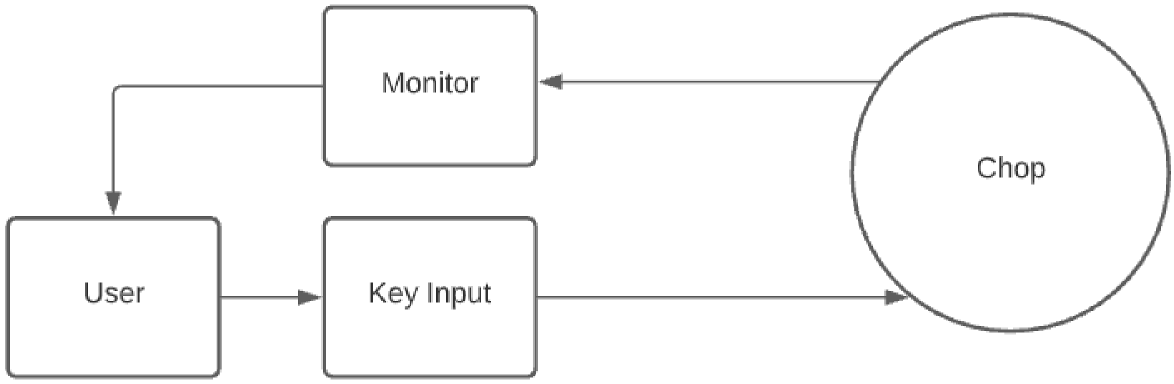
The animator module is very important time saving module for creating animations using a string of graphics. In other software like Greenfoot, animations are very difficult to create, requiring you in the code to type in the file location of the graphic, and call each graphic in the code one at a time. Calling GIF animations are also very difficult as well. The animator module on the other hand only requires you to attach the module to a game object, and simply place the graphics in the animator timeline. This makes it much easier as it doesn’t require you to call each frame by hand, it allows you to focus on timing and other animation principles instead.

All of these modules can be called and referenced in the code. They have to be called however as private or public classes, with their modules being specifically called in the start or awake function. These modules and scripts are incredibly important and useful as they increase the speed of code generation. They save time because they help create the processes for the basic game mechanics.

Documentation of Overall Software Solution

Represent your software solution in diagrammatic form

Context Diagram



IPO Diagrams

Player Class

Input	**Process**	**Output**
"A" Key or "LeftArrow" is pressed	• Move character left • Check facing right as false • Play run animation • Flip run animation	• Character is moved left • Run animation is played • Animation and hitboxes are flipped
"D" Key or "RightArrow" is pressed	• Move Character right • Play run animation • Check facing left as false • Play run animation • Flip run animation	• Character is moved right • Animation is played
"W" Key or "UpArrow" is pressed	• Check if Grounded • Move character up • Play jump animation	• Character is moved up • Animation is played
"LeftMouse" is pressed	• Attack animation is played • Attack Hitbox registers • Enemy class health is reduced	• Deals damage to enemy • Attack animation is played

Menu

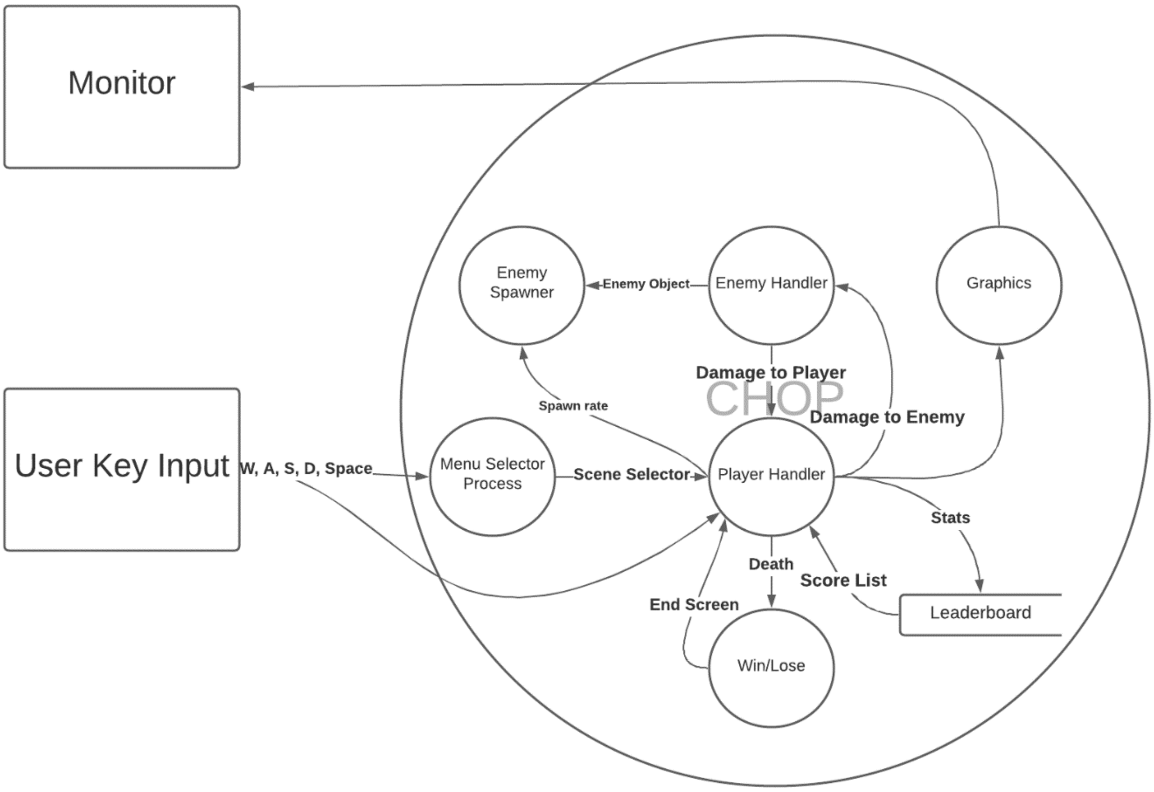
Input	**Process**	**Output**
"Start" button is pressed	• Access character select screen • Play select animation • Change game scene to character select screen	• Character select screen is opened
"Quit" button is pressed	• Play select animation • Quit process is started	• Game quits

Character Select Screen

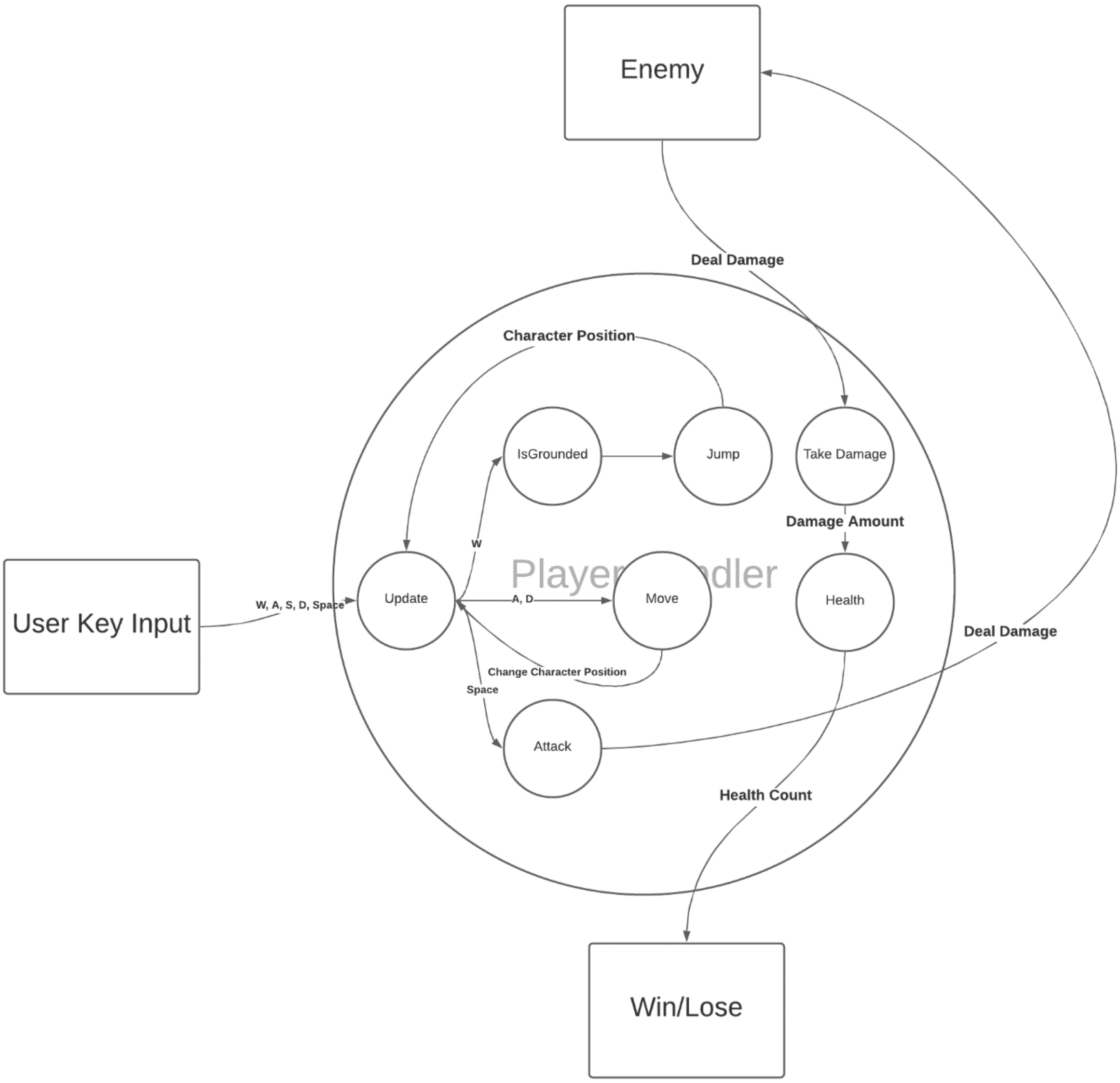
Input	Process	Output
"Character 1" button is pressed	• Find Arena scene with Character 1 • Play character select animation • Change game scene to Arena Character 1	• Arena with Character 1 is opened
"Character 2" button is pressed	• Find Arena scene with Character 2 • Play character select animation • Change game scene to Arena Character 1	• Arena with Character 1 is opened

Data Flow Diagrams

CHOP DFD Level 1

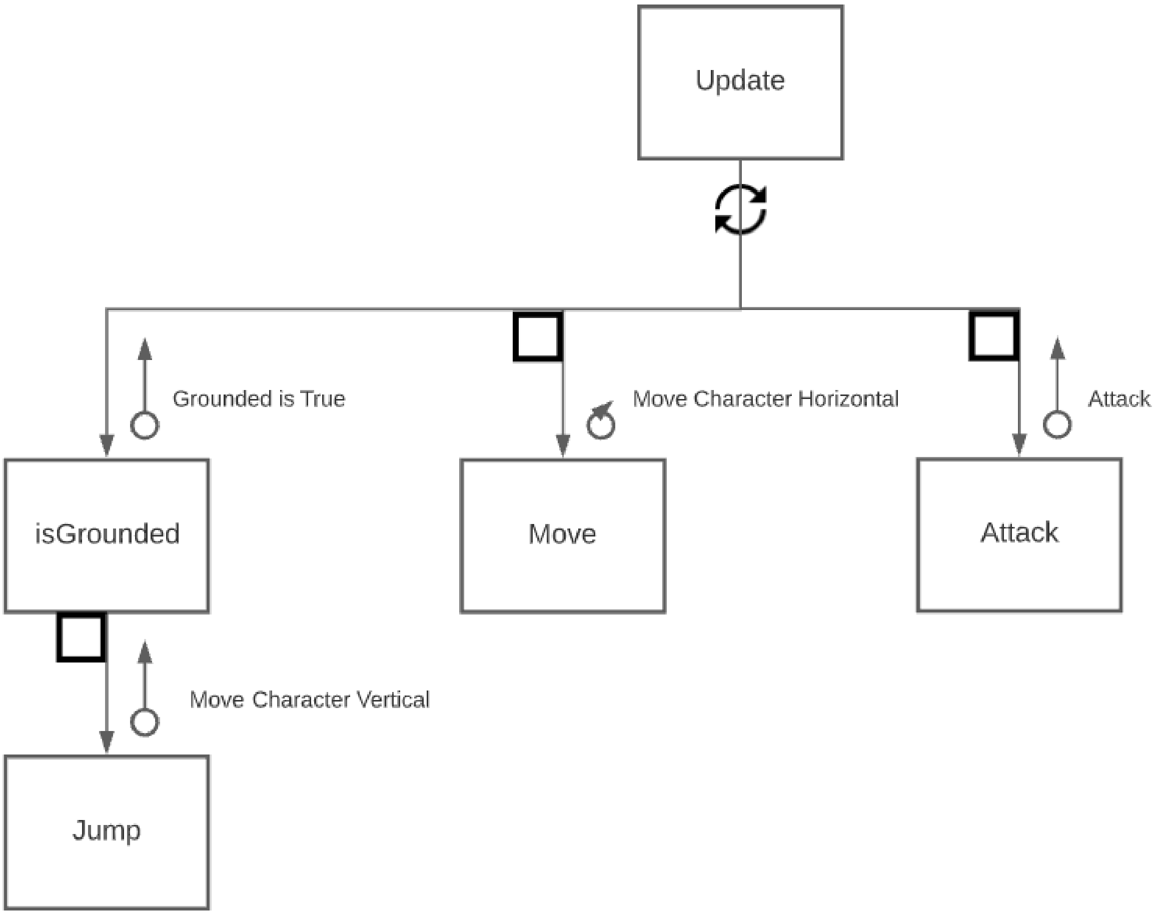


Game Handler DFD 1.1

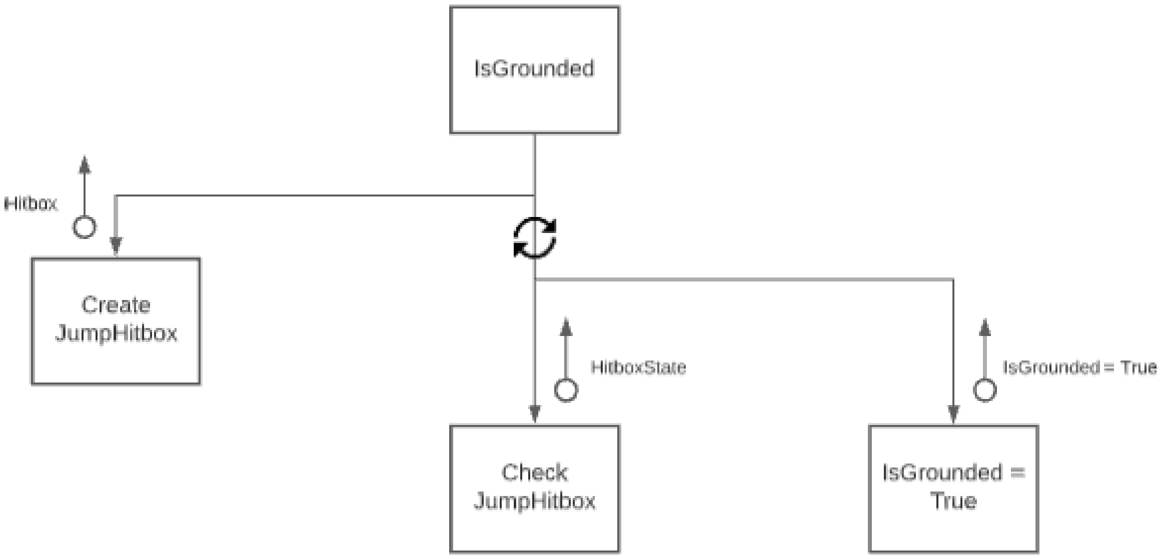

Enemy
Deal Damage
Character Position
IsGrounded
Jump
Take Damage
Damage Amount
W
Player Handler
User Key Input
W, A, S, D, Space
Update
A, D
Move
Health
Deal Damage
Change Character Position
Space
Attack
Health Count
Win/Lose

Structure Diagram for Each Class

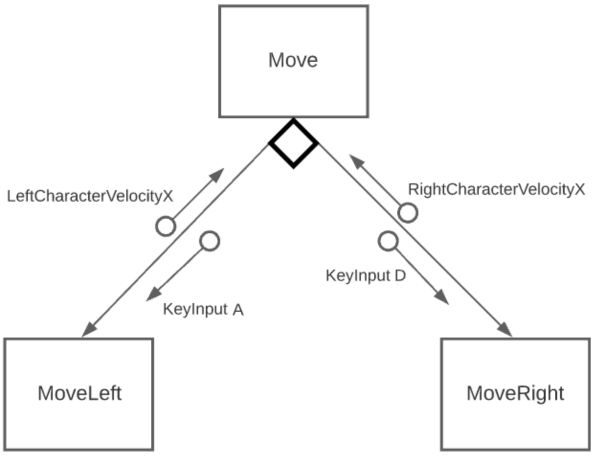
Update Class



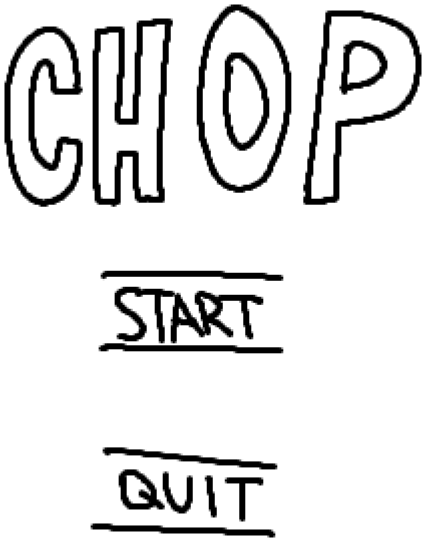
IsGrounded Class

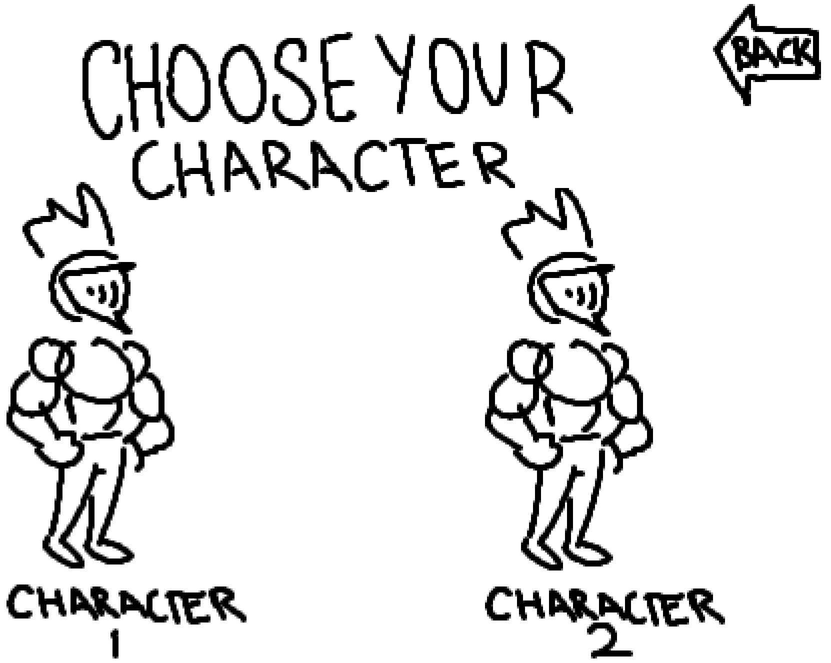
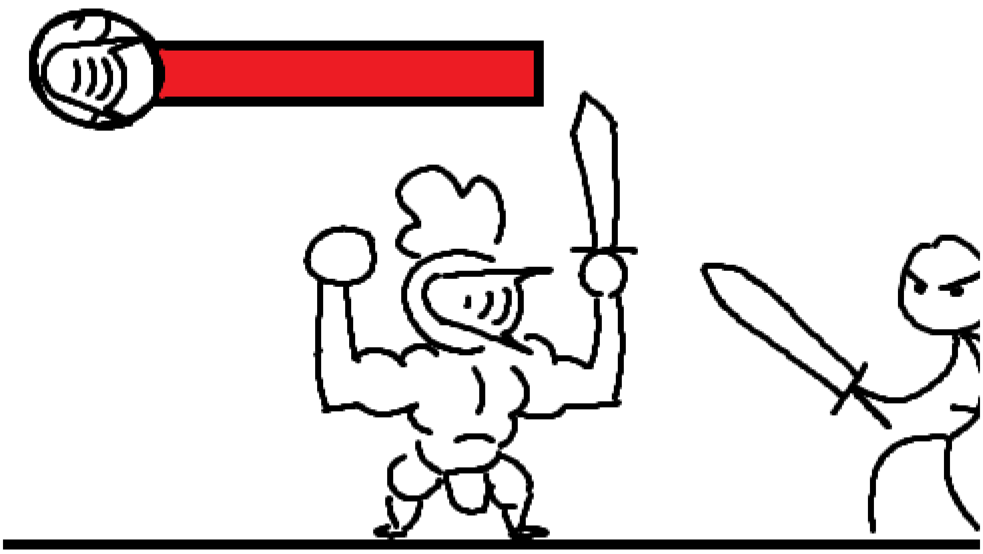


Move Class

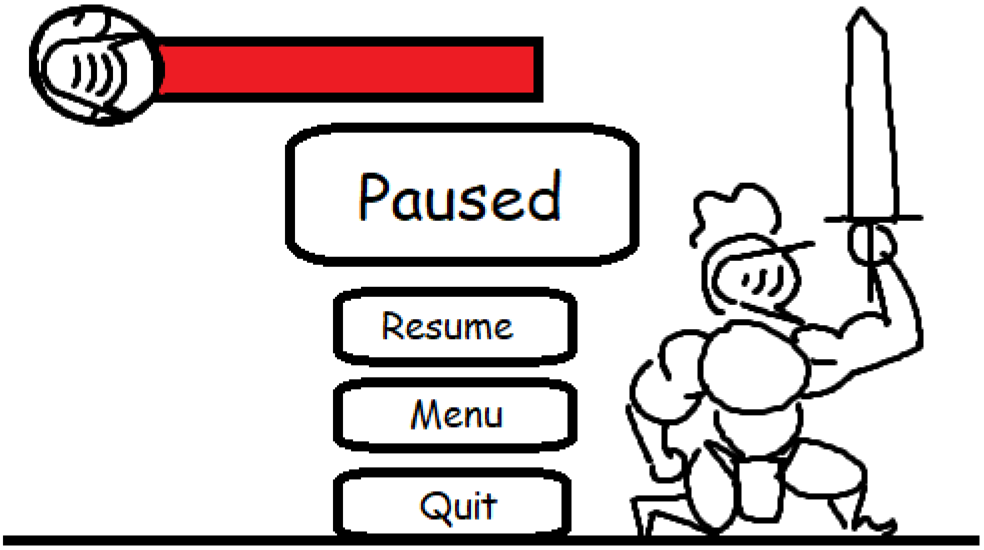


Storyboards of Screen Design

Main Menu

The main menu will simply have the title of the game in a large graphic that may be animated for extra flair. It contains the start button and the quit button, The start button sending the player to the character select screen, and the quit button quitting the game. The menu is visually simple and uncluttered to make sure the user knows exactly what’s happening. The background will also be animated to add visual interest to the main menu.

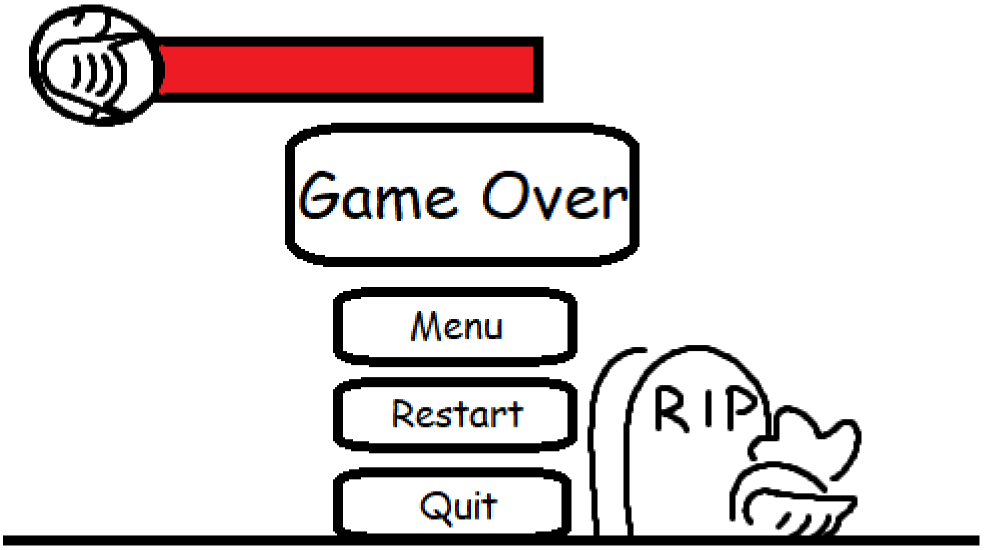
Character Select

he character select screen will be in the same vein as the main menu screen, but instead featuring a button to select either character 1 or character 2. Each character will have an animated button showing the general play style of each character. There is a back button below to return the player back to the main menu screen. There will also be a large logo showing the user that this is the character select screen.
Game Screen

The game screen is visually uncluttered to make gameplay simpler. There will be a character spawned in the center who can move across the screen, and the floor which overlaps slightly behind the character to give the floor a 3D effect. There will also be a healthbar, with a portrait of the character attached next to it. Enemies will spawn from the left and right of the screen and walk towards the character.

Pause Menu



The pause menu comes up anytime during gameplay, and completely pauses the game behind it. There will be a big logo that shows the game is paused, and the screen behind will be darkened to make the buttons contrast from the graphics of the game. There will be a resume button that unpauses the game, a menu button that sends the player back to the menu, and a quit button that quits the game.

Game Over



The game over screen works in the same vein as the paused screen, instead showing a large Game Over graphic, and instead replacing the resume button with a restart button which restarts the game scene, but still keeping the menu and the quit buttons. The gameplay behind it will also be darkened so the buttons contrast from the gameplay.

Create a Data Dictionary to document Class and method variables, broken down by class.

Identifier	Data Type	Number of Storage Bytes (MAX)	Scope	Description	Example
isGrounded	Bool		Private, object level	Based on whether the grounded hitbox is collided with the ground or another object, it returns either True or False.	True
Speed	Float		Public, object level	Controls the speed at which the character moves.	50
jumpHeight	Float		Public, object level	Controls the height that the character jumps at.	30
MaxHealth	Int		Public, object level	Controls the maximum health of the character.	100
currentHealth	Int		Private, object level	Holds the current health of the character and can be affected by damage or healing.	35
attackDamage	Int		Public, object level	Controls the damage output of the character.	200
attackRate	Float		Public, object level	Controls the rate at which the character can attack, making attack spam impossible.	0.5
nextAttackTime	Float		Public, object level	Holds the next time which the character can attack. Is connected to the attack rate.	0f;
AttackRange	Float		Public, object level	Controls the range at which the character can attack.	50
MoveSpeed	Float		Public, object level	Controls the move speed of the enemy.	20
attackChase	Int		Public, object level	Controls the range at which the enemy will chase the player object.	100

Interface design in Software Solutions

Issues related to interface design

While developing the software, because of its genre as a game, the user interface is a very important aspect of the user experience. A bad user interface design can be detrimental to the success of a software solution, making the effort and time spent on the software be wasted because they are drawn away by a bad or ineffective user interface design.

When designing the user interface, there are many important factors and issues to consider, like :

- The target audience of the software solution
- The screen size of the software
- The consistency of the screen design
- Social and ethical issues
- Online help
- Support for disabilities
- Device compatibility (PC, touchscreen, etc.)
- Over designing or over simplification
- Offensive UI design (to a religion, race, culture, etc.)
- UI design of previous software solutions

While there are many issues and factor to consider when creating a software solution, it is not possible to cater for each and every issue or demographic, so some considerations may be outside the bounds of the software solution.

For the target audience, due to the software solution being a video game, it may target audiences from teenagers to people in their mid 30s. This allows the graphics and user interface to be more complex and less straightforward, as older or much younger audiences are not the target audience for this game. However, this doesn't mean that I'm allowed to overdesign the user interface because even to an audience within the teenager to young-adult range, a complex user interface may be too confusing or cluttered and has no place in a game that's meant to be fun, so a simpler user interface that even older or younger audiences could understand is appropriate for the software solution.

The screen size is another important consideration for the software. The size of the screen could affect the interface elements. For example, in the game, a health bar is always in the top right of the screen while still maintaining a consistent size. At larger resolutions, the health bar would be too small to see. So, limiting the maximum resolution to 1280 x 720p allows the UI elements to stay readable, while still maintaining a high-quality game resolution for players with hardware that can handle it.

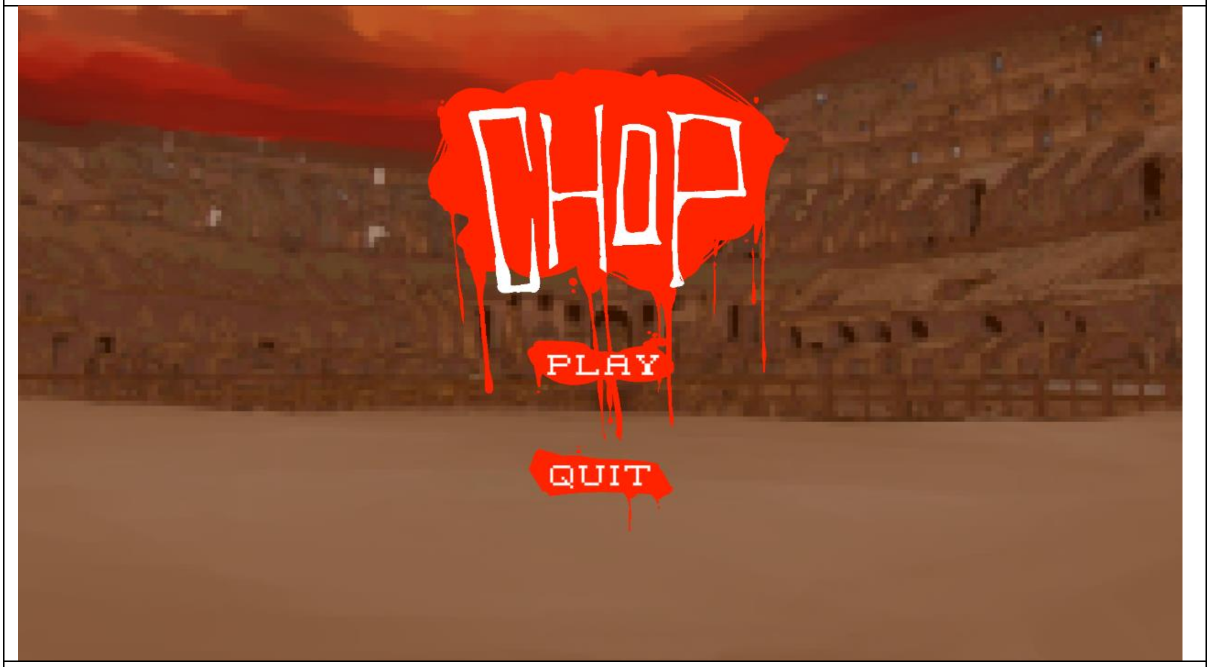
The offensiveness and the social and ethical issues of the software is hard to cater towards. The offensiveness of the title "CHOP" and the characters featured could bring up some issues to the public. The issue of inclusivity is easily remedied by allowing a range of different ethnicities and genders within the characters of the game, allowing the player to choose 2 characters of opposing genders. However, the issue of race and culture may still be problematic. Having only a single race apparent as the enemies is an obvious racial issue, however, allowing both the main characters AND the enemies to have a range of ethnicities and genders will remove this issue. But this means coding in a way to randomize the ethnicities and genders of enemies which can prove to be difficult, so for the game, the enemies and characters will have a generalised non-specific skin tone which doesn't suggest a specific race or culture.

The user interface also has to keep a consistent design throughout gameplay and use of the software. All UI elements from the main menu, the character select, pause screen and the gameplay should all have a consistent style and design. Having different designs and styles throughout the game would be jarring and unappealing to the user. Consistent placement and alignment of UI elements also helps with the flow of the user interface. Having menu elements align with the centre of the screen in places like the main menu and the pause menu make it easier for the user to locate each button, as they are right in the middle. The consistent of placement is important as well, as making each menu have similarly placed buttons and other user interface elements help to make the user know what to expect with each menu and makes things easier to find. Having interface elements in different spots each time can also make the interface jarring to navigate.

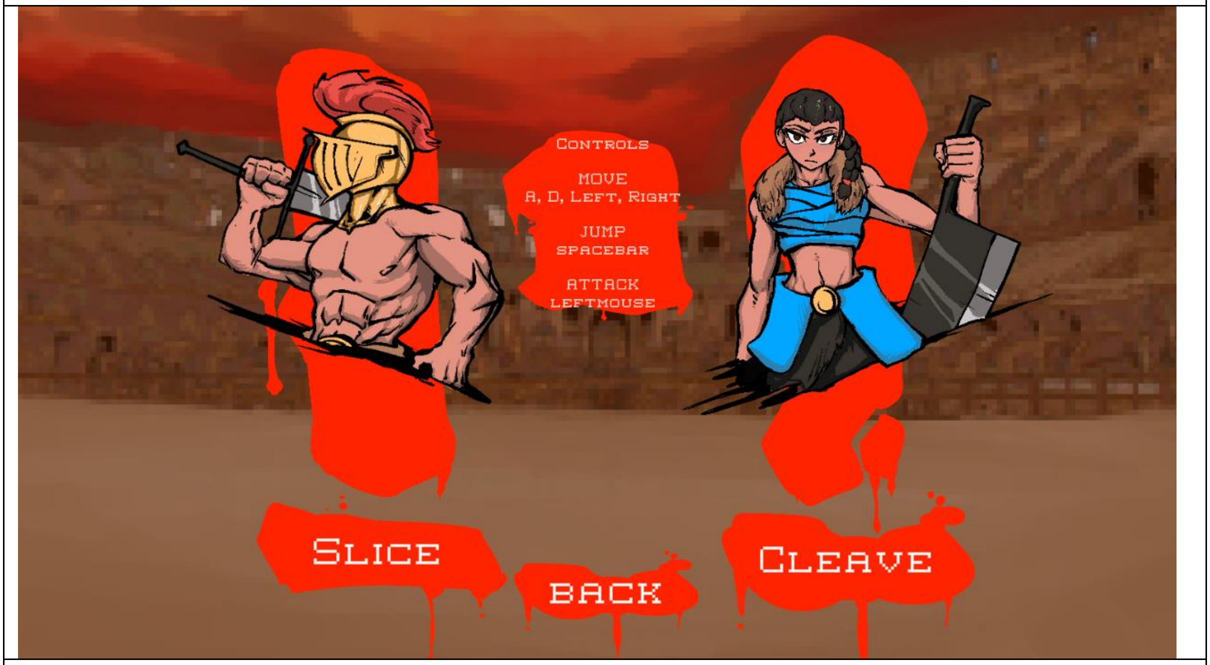
While it is great to think of your own ideas, referencing or drawing inspiration from past software solutions should still be considered. This is an effective way to create an effective UI design because previous software solutions have gone through the same development cycle as you, meaning that they have an idea of what works. It is possible to take the ideas of the previous solutions that apply to your one. However, previous software solutions can be a way to see which user interface designs DON'T work, so they can serve to find out what to avoid when designing a UI.

The Castle Crashers game UI is an effective reference for UI design as they have very simple UI elements that don't clash with the overall feel or even the graphics and the gameplay. Following their design for the UI with simple and easy to read buttons and elements like the health bar and the inventory menu are even a way to add even more personality or flair to the overall project.

Storyboard with Complete Screen Designs

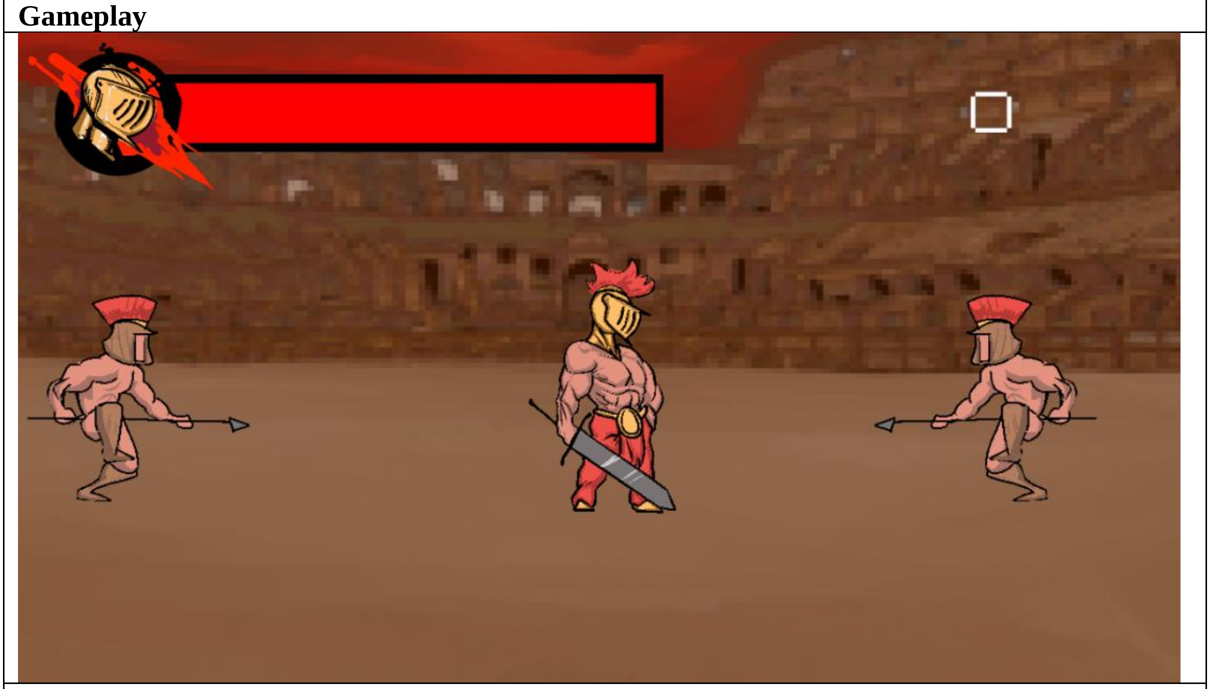
Main Menu

The main menu is quite simple like stated in previous designs. For purposes of contrast, the text is a bright white, and to make it stand out even more, it has a contrasting red element behind it to make the thin text easier to read against the visually complex background. The play and quit buttons are evenly spaced out from each other to make each easier to distinguish.

Character Select



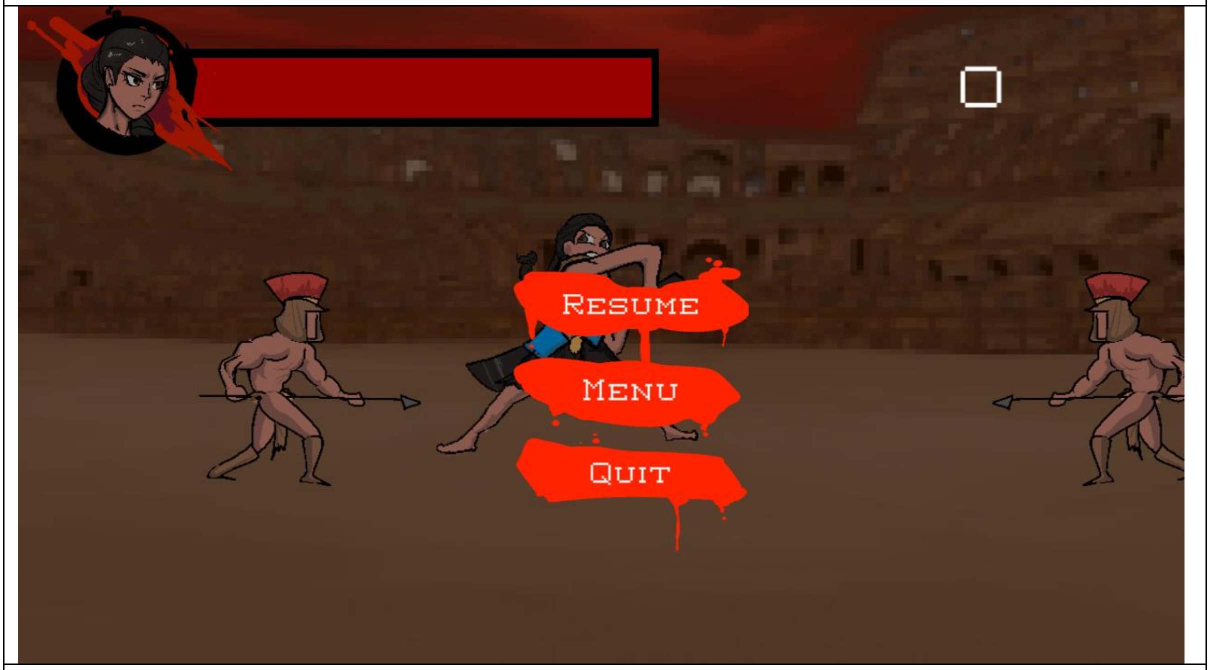
The character select screen shows a back button to return to the previous menu, and a button labelled with the name for each character. To make it easier for new players to distinguish the playstyles of both characters, a character portrait is added to each button. Basing the characters play style on their design, with the faster character having the lighter weapon, and the slower and stronger character having a heavier one. This makes it easier for players to understand each characters strengths and weaknesses through visual communication.

Gameplay



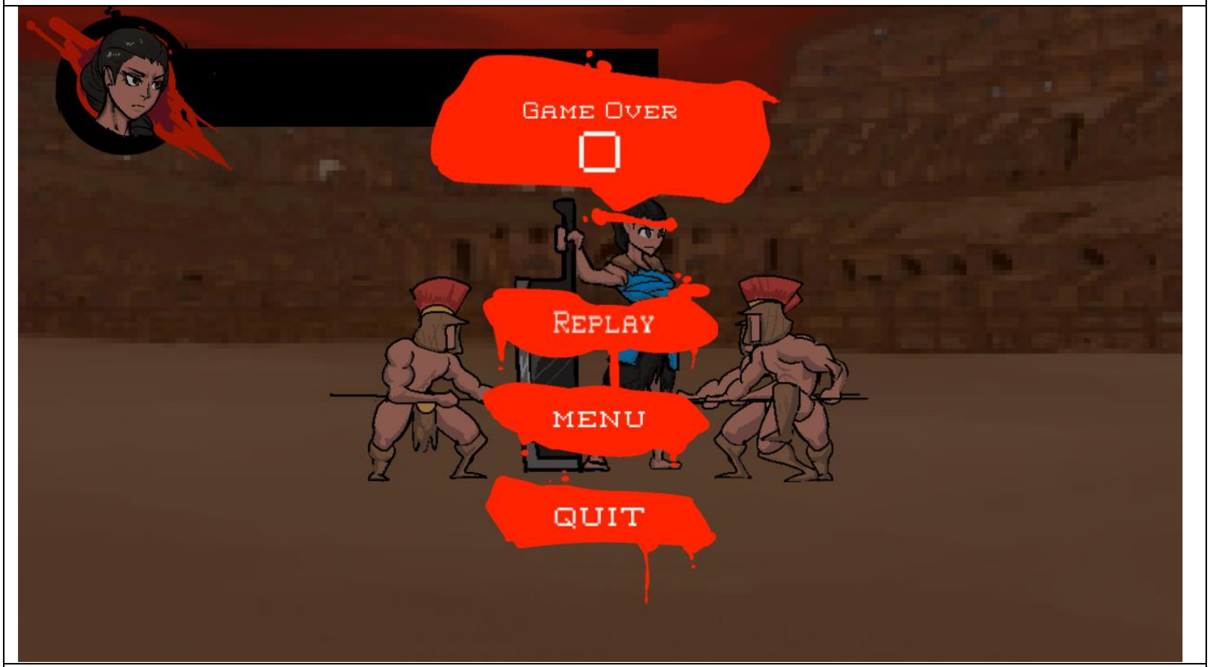
The gameplay, like in the storyboard sketches has enemies coming in from the left and right. There is a red health bar with a black border, and the current characters icon next to it. The score is on the top right but has no red element behind it, but to make it easier to read, it is much larger to make the font easier to read. It is also placed in a spot where the background is much darker and contrasts with the text.

Pause Menu



The pause menu is simple and has a darkened background to make it easier to read from the gameplay. The screen being darkened also helps to make the player understand that the game is currently paused and is in a different state. The UI elements are evenly spaced in the centre to make it easier for the player to find and click.

Game Over Screen



The game over screen follows the same principles as the pause menu, and shares the same buttons, but this time the resume button is replaced with the replay button which restarts the level with the same character. There is also a large UI element at the top which displays “Game Over” to make the player know why the game stopped. It also displays the final score of the player.

Factors to consider when selecting the programming language

The programming language is the backbone of the software solution and must be chosen wisely according to the needs of the user and the developers' skills. There are many factors to consider when choosing the appropriate programming language:

- The purpose of the programming language
- Its advantages and disadvantages
- The requirements of the solution
- The performance of the language
- The programs and support for the language
- The fluency of the developer when using the language
- The speed of code generation using the language

While I have coded in Java through Greenfoot for my previous project and in many in class projects, The Greenfoot program itself offers outdated tutorials and support, and the program is less modern and may lose support in the future. Even though I have experience in Java, my it is still very limited. My Python coding proficiency and fluency is also the same story, yet I am a bit more fluent in it, however the ability to create advanced graphics in real time is much more difficult in the python programming language which does not follow the needs of the solution.

Another language I have coded in is Prolog, which allows the use of artificial intelligence and logic. However, the needs of the software have no use for artificial intelligence, and the software is not very effective in creating graphics and is not very developed.

C# is a general-purpose programming language and is used for Unity. The language is general purpose and has a tremendous amount of support. Like the rest of the listed programming languages, it is an Object-Oriented Programming language which fills the needs of the user requirements. The large amount of support covers a wide variety of software solutions which helps to broaden my ideas for the game. C# and its support with Unity also helps with compatibility because of its constant updates as a software, allowing support for a wide range of hardware.

The language must support the use of the two most common platform MAC and PC, and luckily due to the versatility of coding languages, the most appropriate choices both Java and C# are supported on both. This allows the client to run the software no matter what hardware they have to run it.

Java and Greenfoot are a low performance combination, and its ability to create games is very limited and basic. In the maintenance of the software after implementation, it would be hard to add newer and more complex features to the software due to the limitations of the Greenfoot program. Planning for the future, using more modern software like Unity and C# allows not only for more complex features to be added after implementation, but also helps in finding developers to maintain the code due to the abundance of C# programmers in the industry.

The programming language chosen for this solution is C# as the large number of resources for learning allow me to surpass the knowledge I have for both Java and Python. The software supported by C# is also much more versatile and is even an industry standard, allowing me to train myself for the future. The language also aligns with the user requirement of it being an Object-Oriented Programming language.

Factors to consider when selecting the technology to be used

Performance Requirements

The performance requirements are an important aspect of the software development because it outlines what users should expect to use when running the software. Without performance requirements, users would not know whether they could run the software, which could even shy them away from purchasing the product.

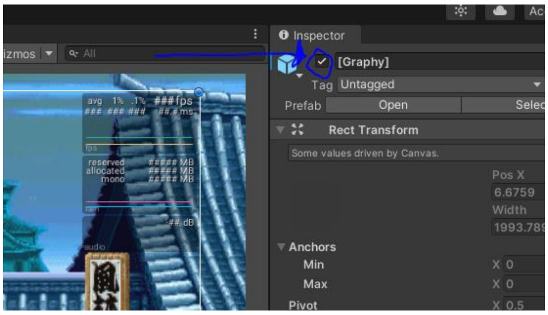
On the developer end of performance requirements, they must ensure that the software consistently runs at a maintainable speed throughout the use of the software. It is important for them to decide factors which they must reach and maintain in order for the software to be optimized. Factors like:

- Game must run perfectly on frame caps like 60FPS and 30FPS
- Enemies should disappear after being killed to save performance
- The load times should be short (less than 5 seconds) as the game is not very hard to run

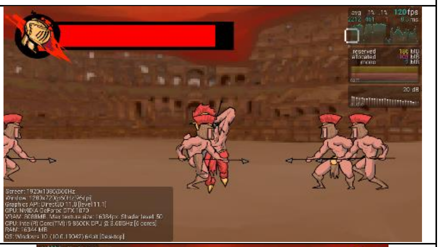
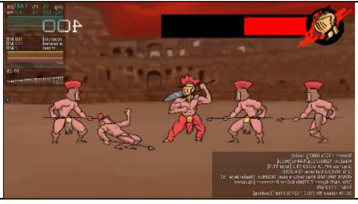
These factors must be considered and benchmarked to maintain a consistent performance throughout gameplay and use of the software.

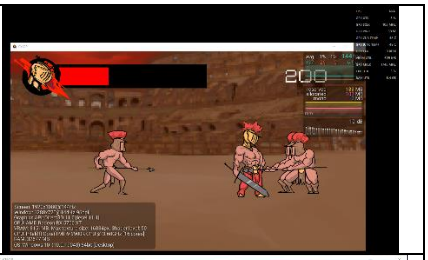
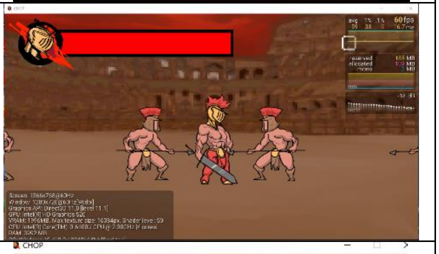
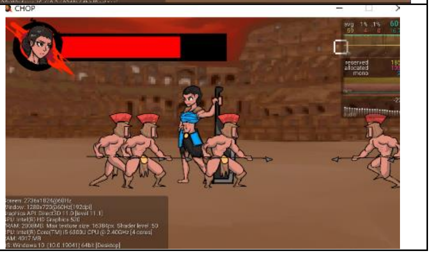
Benchmarking

Benchmarking can help not only address the performance requirements of the game, but also to help developers see whether they must optimize their software. Thanks to the popularity of Unity, free benchmarking tools have been made by other users which can be downloaded and used for your own software.

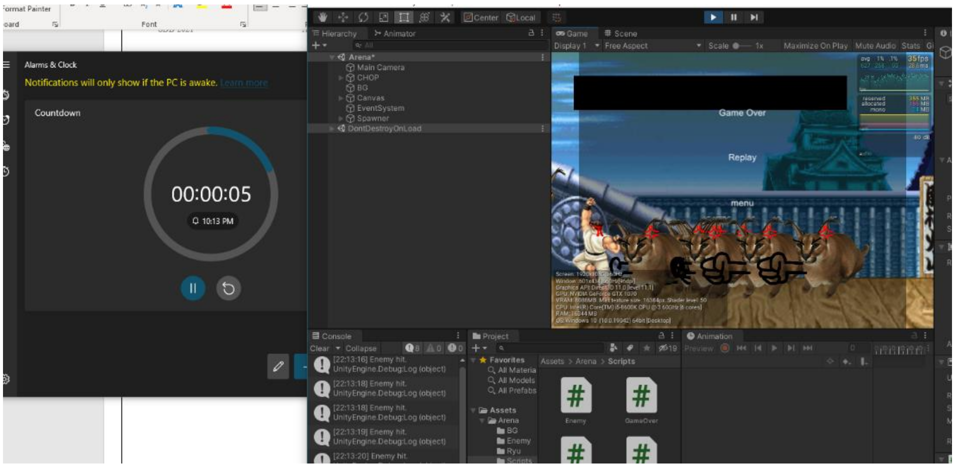


For my software I added a free frame counter that can be opened in the inspector section of Unity. Using the frame counter, I can set the limitations of the system based on how many frames each system can get within 30 seconds of gameplay. In the table it shows:

Platform	Frame Count Avg. (after 30 seconds of gameplay)	Screencap
PC (GTX 1070, i5 8600k)	120FPS	
PC (RTX 2060, Ryzen 7 3700X)	144FPS	

PC (RX 5700 XT, i9 9900K)	144 FPS	
PC (HD Graphics 520, i3 6100U)	60FPS	
PC (Intel HD Graphics 520, i5 6300)	60 FPS	

An issue I had with benchmarking was that the FPS was actually capped in the build version of the game. Inside unity, the frame count exceeded the cap upwards to 2000 FPS. However, while it is capped, all PCs reached an FPS over 60, meaning that each machine reached a suitable performance. The most important benchmark however was the PC with HD Graphics 520 and an i3 6100U. From experience, I remember this person PC having issues simply running a word document if the word count was over 5000, or 30 pages. Having my game run at full 60 FPS means that a PC with low specifications like this is not even the minimum, but the ideal system requirements for my software solution.



Alarms & Clock

Notifications will only show if the PC is awake. Learn more

Countdown

00:00:05

10:13 PM

```
[22:13:16] Enemy hit.
UnityEngine.Debug:Log (object)
[22:13:18] Enemy hit.
UnityEngine.Debug:Log (object)
[22:13:18] Enemy hit.
UnityEngine.Debug:Log (object)
[22:13:19] Enemy hit.
UnityEngine.Debug:Log (object)
[22:13:20] Enemy hit.
```

Testing the game before implementation of custom graphics.

Implementation of Software Solutions

Updated Data Dictionary

Identifier	Data Type	Number of Storage Bytes (MAX)	Scope	Description	Example
isGrounded	Bool		Private, object level	Based on whether the grounded hitbox is collided with the ground or another object, it returns either True or False.	True
facingRight	Bool		Private, object level	Based on whether the player has inputted left or right, if right, the bool returns True, and vice versa.	False
Speed	Float		Public, object level	Controls the speed at which the character moves.	50
jumpHeight	Float		Public, object level	Controls the height that the character jumps at.	30
MaxHealth	Int		Public, object level	Controls the maximum health of the character.	100
currentHealth	Int		Private, object level	Holds the current health of the character and can be affected by damage or healing.	35
attackDamage	Int		Public, object level	Controls the damage output of the character.	200
attackRate	Float		Public, object level	Controls the rate at which the character can attack, making attack spam impossible.	0.5
nextAttackTime	Float		Public, object level	Holds the next time which the character can attack. Is connected to the attack rate.	0f;
AttackRange	Float		Public, object level	Controls the range at which the character can attack.	50

Dead	Bool		Public, object level	Holds whether the character is dead or alive. Connected to the currentHealth.	True
MoveSpeed	Float		Public, object level	Controls the move speed of the enemy.	20
attackChase	Int		Public, object level	Controls the range at which the enemy will chase the player object.	100
STime	Float		Public, object level	Holds the current rate of spawning for the enemy spawner.	10
changeInTime	Float		Public, object level	Controls how much the spawn rate changes over time.	0.5
minusTime	Float		Private, object level	Holds how much the change in time increases, compounding.	1.5
changeRate	Float		Public, object level	Controls the rate at which the changes to the rate of spawning are changed over time.	3

Common modules

A few common modules are prevalent in my software solution. The Rigidbody2D module is the base for most of the physics in all of the playable characters and enemies and is controlled by the user inputs. Rigidbody2D velocity and magnitude are often used to control the speed and direction of the function. The BoxCollider2D is another important module used both in keeping the player on the ground and in the bounds of the arena, and also to check the collisions of enemy and player attacks. It is an important module used in the ground check, which draws a box in a range underneath the player to check whether the player is touching the ground or not. The animator module is an important module that not only controls the animation, allowing functions to play certain animations, but also allows the animations to call certain functions within the script, making things like the moment the player attacks to be timed with the frames of the animation.

Techniques in developing well-written code

Good Programming Practices

To ensure success throughout all steps in the Software Development Processes, good programming practices are useful to follow to avoid issues or problems that you may face along the road. Good programming practices include:

- Following the programming language naming conventions
- Implementing internal documentation
- Using version control software
- Creating regular backups of the program and documentation

Internal Documentation and Programming Language Conventions

Meaningful variable names

Meaningful variable names are important in your code. The names of your variables should be coherent and written properly so that it is easy to read and to understand. It can also lessen confusion between different variables if you use proper naming and spelling. The variables name should also describe what it is for and what it is doing. If you are also sending the code to someone else, it helps to make variable names something easy to read and understand for a person who has not read your code before. Variables in your code should be easy to identify and understand.

Examples of bad variable naming:

```
Camell = Trex

Bssadskon == True:

daduDDe = input(“whats your name bro”)
```

Examples of good variable naming:

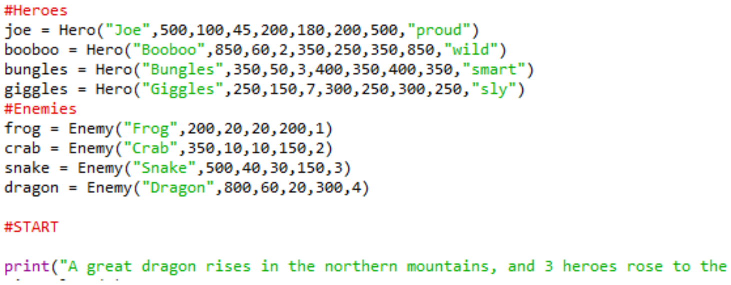
```
UserNumber = 3

FinalAddition = 3 + 6

Name = input(“What is your name?”)
```

Readability of Code

The readability of code is very important and goes beyond the meaningful naming of variables. All your code should be easy to read for not only yourself but other programmers that may have to read your code. Your code should be easy to read and understand. In the industry work can get passed to others so the ability for others to read your code should be nice and simple.

Comments

Comments are important in coding as they can help for the programmer to understand what each part of their code does. Comments do not have to be long and should quickly explain the things your code is meant to do. It can also show what you are trying to do behind lines of code as a person who does not know what you are doing can understand if you have comments in your code explaining what you are trying to do. In python comments using the hashtag (#) often come up as bright red so they stand out and are easy to see. This can help in reminding the programmer to return to a certain piece of code to work on it later. It can also help in debugging as it can highlight a certain problem in your code.



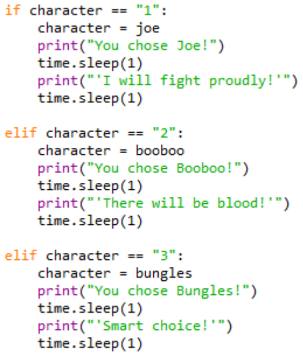
```
#Heroes
joe = Hero("Joe",500,100,45,200,180,200,500,"proud")
booboo = Hero("Booboo",850,60,2,350,250,350,850,"wild")
bungles = Hero("Bungles",350,50,3,400,350,400,350,"smart")
giggles = Hero("Giggles",250,150,7,300,250,300,250,"sly")
#Enemies
frog = Enemy("Frog",200,20,20,200,1)
crab = Enemy("Crab",350,10,10,150,2)
snake = Enemy("Snake",500,40,30,150,3)
dragon = Enemy("Dragon",800,60,20,300,4)

#START

print("A great dragon rises in the northern mountains, and 3 heroes rose to the
```

White Space

White space in coding is an empty area that takes up space between blocks of code. It can be used to represent the idea that two lines or blocks of code do different jobs. They are often used in this sense as sometimes large lines of code make begin to get confusing for the programmer. This is effective in debugging as it helps for the programmer to identify what parts do what in case one of them is not working correctly. Like how paragraph spacing in writing literature, white space in coding is important in separating different parts of the code.

Indentation

Indentation is particularly important as it helps for the programmer to understand which code falls under what. In python this is exceedingly important as the code indented under a statement shows what sequence that statement needs to run. In things like IF statements and LOOPS, it shows what code runs when a condition is met. Proper indentation using TAB in coding is important in avoiding syntax errors.



```
if character == "1":
    character = joe
    print("You chose Joe!")
    time.sleep(1)
    print("'I will fight proudly!'")
    time.sleep(1)

elif character == "2":
    character = booboo
    print("You chose Booboo!")
    time.sleep(1)
    print("'There will be blood!'")
    time.sleep(1)

elif character == "3":
    character = bungles
    print("You chose Bungles!")
    time.sleep(1)
    print("'Smart choice!'")
    time.sleep(1)
```

Coding to increase maintainability of code

Ensuring maintainability of code is important for your software solution because they allow the original developer or even a new developer who isn’t familiar with the solution to effectively maintain and add newer features to the software solution. Throughout my solution, a lot of my code is abstracted and allows for newer features to be added very easily

For example, important character stats like health, damage, speed, etc. are all public variables which can be accessed through Unity allowing quick and easy game balancing for future game updates. Different features like attacking, movement, jumping, etc. are all separate functions which are called in the Update function of Unity, which means that changing how the character will attack or changing the way the character will move can easily be done and implemented into the game. The moment the character attacks is a separate function within itself which can be called by animations in the game, and as previously my graphics were a placeholder attack animation, it was easy to add new animations by simply creating a new animation and applying the characters attack function into it. This allows new animations to be updated even after implementation or a full release.

The audio manager is also another abstracted way of adding and playing audio throughout the game. Coding in audio is incredibly difficult in C#, so having a script that controls all of the audio and allows the developer to simply drag and drop audio with customizable parameters and allows easy referencing of the audio manager in other functions, making audio in the game so much simpler. Simple drag and drop audio means that new audio files can be added easily post launch to the game without any hassle.

The fact that the player control character has customizable statistics, and that the character animations can be changed while still allowing functions like attack and different audio files mean that creating and adding new characters is entirely within the realm of reason. First, I created my character player script, and because of the changeable public variables, it means that I can apply different statistics to a copy paste of this character, and the custom animations mean that I can apply the attack function differently, which allows me to change the characters attack speed and range. This is what allowed me to create two characters with different play styles in this game. With this character template, it allows me to maintain the code and add newer characters in the future.

The enemy is created in the same way, with all its statistics and animations being customizable which can allow even different enemy types to be created after release.

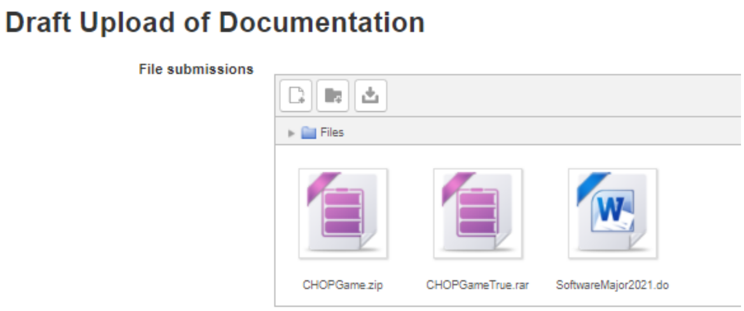
The choice of program being Unity is a big factor to the maintainability of code. Staying with the recommended choice Greenfoot would be hard to maintain after release, because Greenfoot is a very unpopular software. Unity on the other hand is one of the industry standard game creation software with a huge amount of support, meaning that even if the game were to be handed over to new developers, they would have no issues maintaining it because of C# and Unity not only being user friendly but simply being popular as a software.

The abstraction of the code and its functions, as well as the choice of programming language and software are the most important considerations of code maintainability.

Version Control Software and Backups

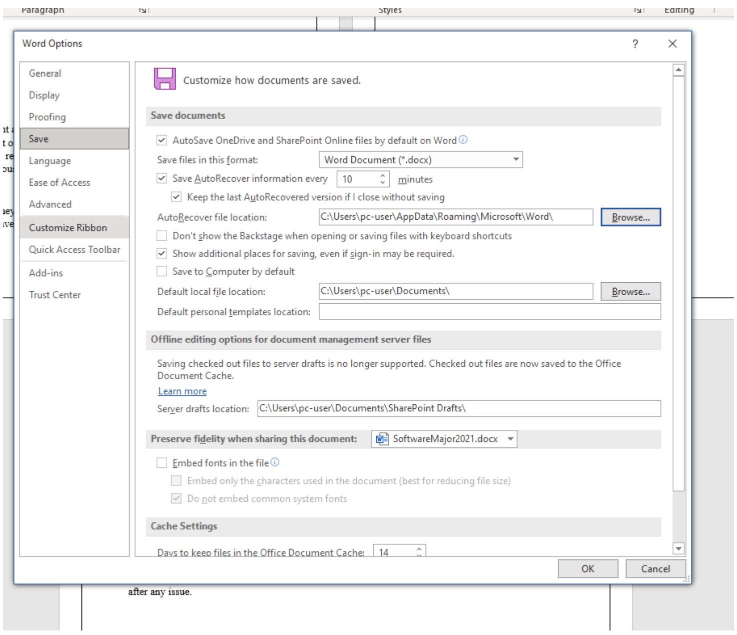
Version control is important so that the developer can backtrack to older versions which require information that was changed that proved to be correct for things like the portfolio. For the software, version control is important as well as the code and scripts used in the game are changed constantly, and if the developer were to find issues with the newer version, and hit a brick wall with the code, they could revert to the original version and either start the new features from scratch or simply find a new way to complete it. In my software, daily uploads to the teacher force us to constantly upload the latest versions of our software solution, so than in the event of issues, the latest version of the game and the portfolio can be recovered.

In the case of my project, I had issues with my game corrupting, and thanks to the uploads to the teacher, I was able to recover the older versions of my game so that I would not have to start from scratch. In Microsoft Word, in the event of a crash of the program, backup files are constantly made by the software while working which ensure that a version of the word file can always be recovered after any issue.



Draft Upload of Documentation

File submissions



Customize how documents are saved.

Save documents

☑ AutoSave OneDrive and SharePoint Online files by default on Word

Save files in this format: Word Document (*.docx)

☑ Save AutoRecover information every 10 minutes

☑ Keep the last AutoRecovered version if I close without saving

AutoRecover file location: C:\Users\pc-user\AppData\Roaming\Microsoft\Word\

□ Don't show the Backstage when opening or saving files with keyboard shortcuts

☑ Show additional places for saving, even if sign-in may be required.

□ Save to Computer by default

Default local file location: C:\Users\pc-user\Documents\

Default personal templates location:

Offline editing options for document management server files

Saving checked out files to server drafts is no longer supported. Checked out files are now saved to the Office Document Cache.

Learn more

Server drafts location: C:\Users\pc-user\Documents\SharePoint Drafts\

Preserve fidelity when sharing this document: SoftwareMajor2021.docx

□ Embed fonts in the file

□ Embed only the characters used in the document (best for reducing file size)

☑ Do not embed common system fonts

Cache Settings

Days to keep files in the Office Document Cache: 14

after any issue.

Errors During Development

While implementing the software solution, it is common to face a wide range of errors while programming. It is important however to recognise these errors in order to reduce the mistakes created.

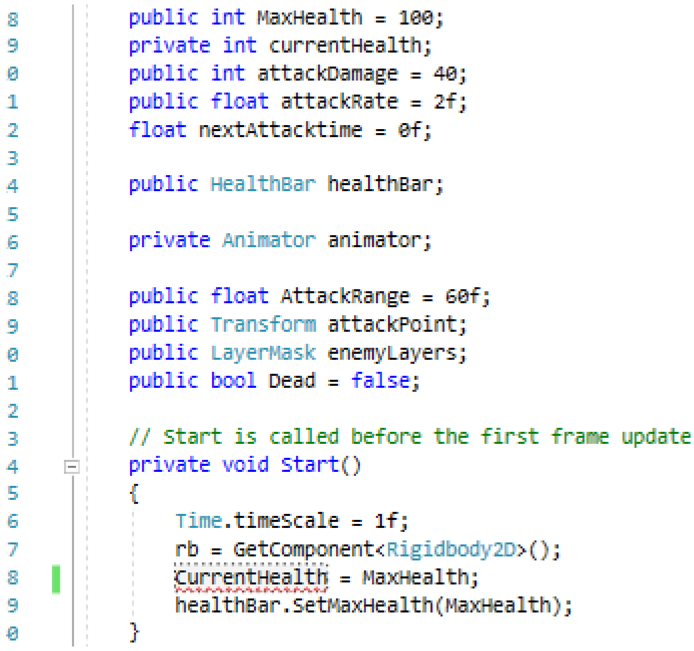
Syntax Errors

Syntax errors can occur when the programmer does a typo in their code. This can be something as simple as a wrong indentation, the missing of a colon, or just spelling something wrong. When the command is run, the error message for a syntax error will typically tell the programmer the line that the error is occurring in. In python, any kind of typo outside of quotation marks can result in the error message SyntaxError: invalid syntax.
e.g. of typos:

iff number == 3:

wyle bananas == Tru::

In my code, this was a common error to encounter. Often, I misnamed my variable names while programming, always forgetting that all my variables often started lowercase. In the caption below you can see the error being underlined in red by Visual Studio. This is useful as it often can recognise without running or compiling whether the written variable exists.



```
public int MaxHealth = 100;
private int currentHealth;
public int attackDamage = 40;
public float attackRate = 2f;
float nextAttacktime = 0f;

public HealthBar healthBar;

private Animator animator;

public float AttackRange = 60f;
public Transform attackPoint;
public LayerMask enemyLayers;
public bool Dead = false;

// Start is called before the first frame update
private void Start()
{
    Time.timeScale = 1f;
    rb = GetComponent<Rigidbody2D>();
    CurrentHealth = MaxHealth;
    healthBar.SetMaxHealth(MaxHealth);
}
```

Runtime Errors

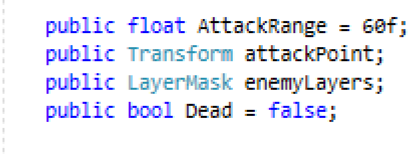
Runtime errors are errors that occur while the program is running. The program can quit unexpectedly due to runtime errors. Runtime errors can occur when the program can read and understand the code but cannot execute it. For example, if you were to try to divide by 0 in Python, the compiler would be able to understand the code, but since it is not possible, the program would crash, and a runtime error would occur. Runtime errors can also occur when there is a lack of memory, or when you are running out of RAM, which is not a user error.

Thankfully, I never encountered this error because the math in my software solution is not very complex, and I often followed the resources and tutorials very closely, making runtime errors almost impossible for me to do.

Logical Errors

Logical errors are typically caused by the programmer. It is when a code is running perfectly fine, does not cause any runtime errors, and is all spelt correctly, but does not produce the correct result. This can be caused in python by using the wrong variable names, making the wrong calculations, etc. Logical errors are fixed the same way throughout all languages. The only way to fix Logical errors is by going into your code to find the problem and to fix it yourself.

In my software, I found my character attacking every single enemy on the stage at once. When trying to fix it myself, I was often suggested to remove the "static" variable between my enemies, which means that the damage taken from the player is shared amongst all enemies. This was not the case however, as none of my variables were ever static, and there never was a need for it to be, and the true issue was that my characters attack range was so high that not only did it attack all enemies on the screen, but the range was so big that I could not see it on the game view and recognise it as an issue. Luckily, I fixed it from a range in the triple digits to a range of 60, which made the enemy die in front of the player in a reasonable range.



```
public float AttackRange = 60f;
public Transform attackPoint;
public LayerMask enemyLayers;
public bool Dead = false;
```

Error Detection Methods

Sometimes errors are very difficult to find and remedy in your software solution. Very often, logic errors are difficult to find because since the developer themselves created the logic error, they may not see what is wrong with what they have written, so there are techniques to detect these errors without too much aid from the coding software.

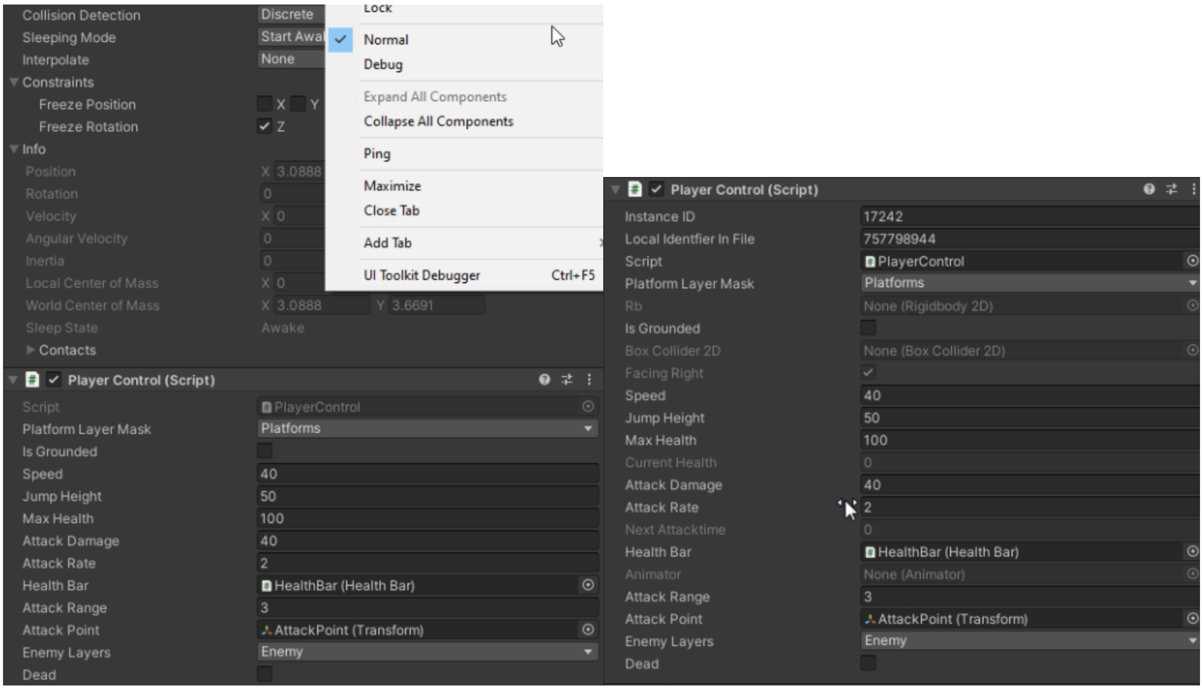
Peer checking is an effective way to find logical errors, as they help see a new perspective on your software which you may not have seen on your own, and because they are seeing everything at face value, they may be able to find errors in logic that you have been spending hours to find within moments. Asking peers to check code has helped me to solve the issues like the massive attack hitbox previously mentioned as they saw that the hitbox on the screen was too large.

Another method to check for issues is to desk check your algorithms. It's important to manually see how each process carries out its instructions, and if the wrong output were found while desk checking, you could see at which steps or repetitions the code begins to output the correct answer, which can help to remedy the issue as it shows exactly what part creates the issue.

A more technical method of error detection is using debugging statements which sends the outputs created by the process into the console. This was an effective method I used which helped me to recognise that the enemies were all being attacked by the player and losing health as I attacked and was not actually a glitch with the game or Unity itself.

Debugging Tools

Debugging tools are important in a software solution because they help to see under the hood of your software solution, seeing the variables at work while actually running it. In Unity itself, they allow a Debug option in the inspector, so that private variables can be viewed while playing to see things like health and spawn rates decrease. They helped to show that the scripts I created were working, and helped in solving issues in my software, like when my character never dropped health, I used the debug option in my inspector to see whether my character actually dropped health while being attacked, and it turned out I had two separate health variables by mistake.



Collision Detection	Discrete
Sleeping Mode	Start Awa
Interpolate	None
▼ Constraints	
Freeze Position	□ X □ Y
Freeze Rotation	☑ Z
▼ Info	
Position	X 3.0888
Rotation	0
Velocity	X 0
Angular Velocity	0
Inertia	0
Local Center of Mass	X 0
World Center of Mass	X 3.0888 Y 3.6691
Sleep State	Awake
▶ Contacts	

Player Control (Script)

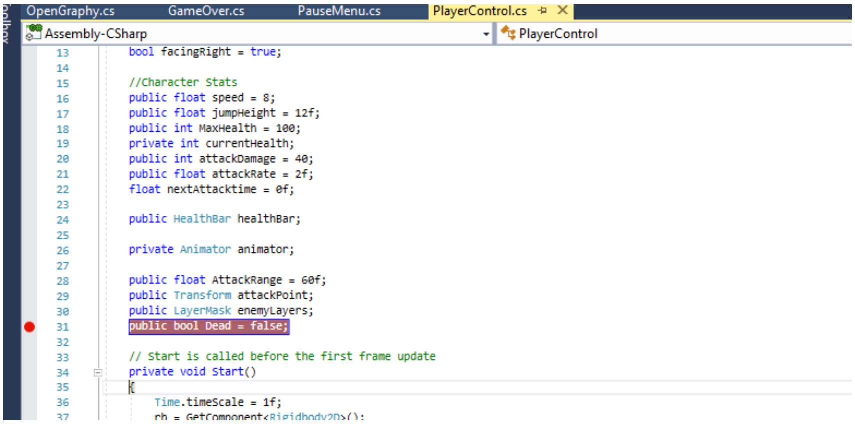
Script	PlayerControl
Platform Layer Mask	Platforms
Is Grounded	□
Speed	40
Jump Height	50
Max Health	100
Attack Damage	40
Attack Rate	2
Health Bar	HealthBar (Health Bar)
Attack Range	3
Attack Point	AttackPoint (Transform)
Enemy Layers	Enemy
Dead	□

Player Control (Script)

Instance ID	17242
Local Identfier In File	757798944
Script	PlayerControl
Platform Layer Mask	Platforms
Rb	None (Rigidbody 2D)
Is Grounded	□
Box Collider 2D	None (Box Collider 2D)
Facing Right	☑
Speed	40
Jump Height	50
Max Health	100
Current Health	0
Attack Damage	40
Attack Rate	2
Next Attacktime	0
Health Bar	HealthBar (Health Bar)
Animator	None (Animator)
Attack Range	3
Attack Point	AttackPoint (Transform)
Enemy Layers	Enemy
Dead	□

On the left, the normal inspector can be seen, but with the debug mode enabled for the inspector, hidden private variables like the current health and the facing right bool are viewable even through gameplay to see them change live to check whether inputs and outputs are working in your script.

Debugging tools are also available in Visual Studio. Clicking to the left of the line or statement marks it as a breakpoint. Pressing F5 runs the debugger until the encountered breakpoint. This helps to make it easier to look at variable values, to check whether a branch of the code is actually running properly and to see the behaviour of the memory.

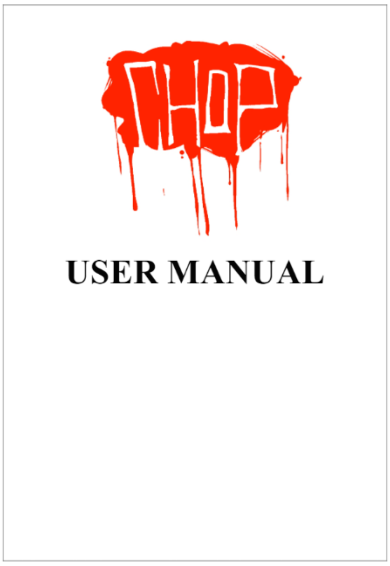


```
bool facingRight = true;

//Character Stats
public float speed = 8;
public float jumpHeight = 12f;
public int MaxHealth = 100;
private int currentHealth;
public int attackDamage = 40;
public float attackRate = 2f;
float nextAttacktime = 0f;

public HealthBar healthBar;

private Animator animator;

public float AttackRange = 60f;
public Transform attackPoint;
public LayerMask enemyLayers;
public bool Dead = false;

// Start is called before the first frame update
private void Start()
{
    Time.timeScale = 1f;
    rb = GetComponent<Rigidbody2D>();
```

Documentation of a software solution

User Manual

The user manual will be available in the game files with the .exe file, and is available as **Online Help** at the link https://www.yumpu.com/en/document/read/65761363/chopusermanual.



SDD 2021 Arnold Quiocho Major Work

Purpose of the Game

The game is an arena that constantly spawns enemies until the player dies. The goal of the player is to gain as much score as possible until the die. The player can choose between two characters, Slice, a fast-paced low damage character, or Cleave, a high damage, slow paced character.

Controls

A, D, Left Arrow, Right Arrow – Move left and right

Spacebar – Jump

Left Mouse – Attack

Installation

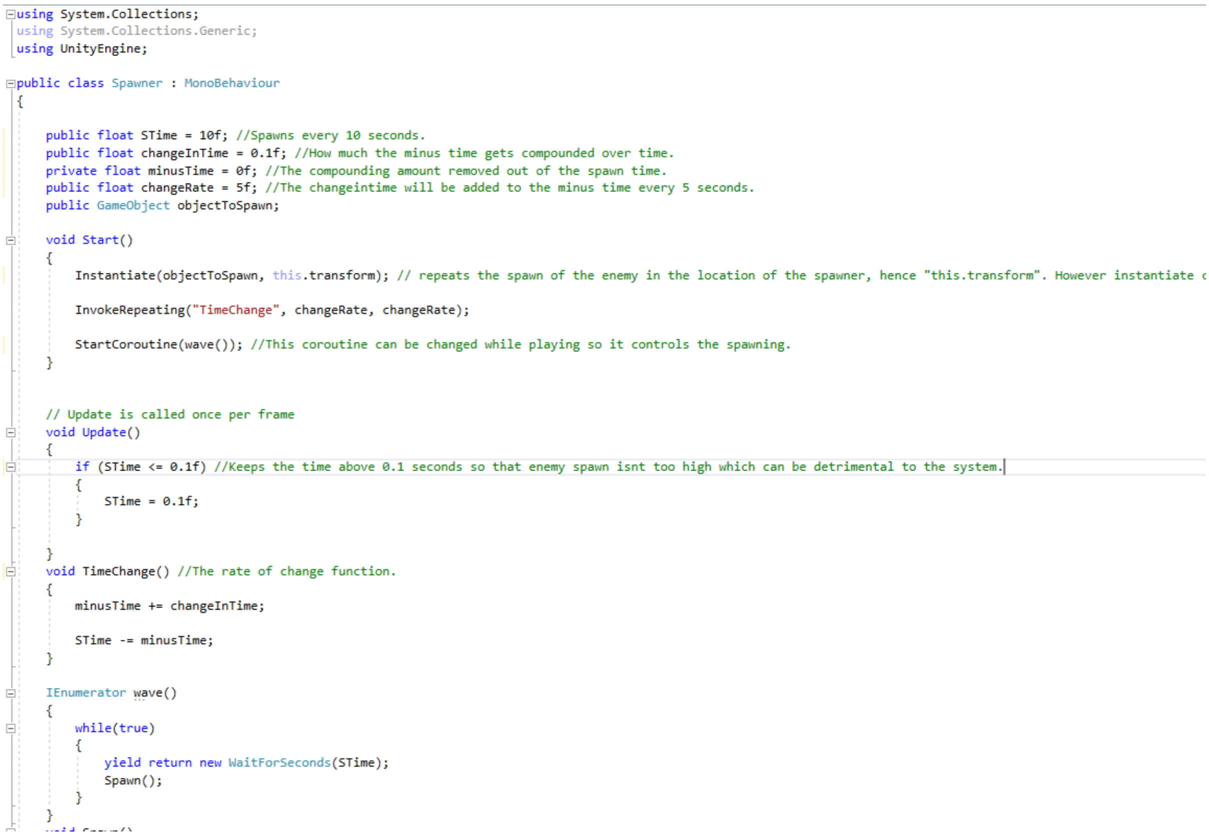
Upon receiving the ChopGame Zip file, simply unzip the file to access the game folder.

After unzipping, simply click on the ChopGame.exe file to launch the game.

CHOP.exe 7/8/2021 5:47 PM
UnityCrashHandler64.exe 6/12/2021 5:19 AM
UnityPlayer.dll 6/12/2021 5:19 AM
1

In Code Comments

Throughout my code there are in code comments available for easy understanding of the functions and processes.



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

public class Spawner : MonoBehaviour
{

    public float STime = 10f; //Spawns every 10 seconds.
    public float changeInTime = 0.1f; //How much the minus time gets compounded over time.
    private float minusTime = 0f; //The compounding amount removed out of the spawn time.
    public float changeRate = 5f; //The changeintime will be added to the minus time every 5 seconds.
    public GameObject objectToSpawn;

    void Start()
    {
        Instantiate(objectToSpawn, this.transform); // repeats the spawn of the enemy in the location of the spawner, hence "this.transform". However instantiate c

        InvokeRepeating("TimeChange", changeRate, changeRate);

        StartCoroutine(wave()); //This coroutine can be changed while playing so it controls the spawning.
    }


    // Update is called once per frame
    void Update()
    {
        if (STime <= 0.1f) //Keeps the time above 0.1 seconds so that enemy spawn isnt too high which can be detrimental to the system.
        {
            STime = 0.1f;
        }

    }
    void TimeChange() //The rate of change function.
    {
        minusTime += changeInTime;

        STime -= minusTime;
    }

    IEnumerator wave()
    {
        while(true)
        {
            yield return new WaitForSeconds(STime);
            Spawn();
        }
    }
    void Spawn()
```

System Documentation and Algorithms

System Documentation and algorithms can be accessed throughout this portfolio, documentation including DFDs, Storyboards, etc. and algorithms through pseudocode can be accessed. The algorithms and code itself can be accessed through the Unity game files before it is built which is in the given zip file.

Hardware Requirements

The hardware requirements are derived from the benchmarks done on a range of computer specifications, and due to the much more complex nature of the Castle Crashers game compared to my own, the minimum requirements will be the same. The minimum system requirements for the software solution are:

- OS: Windows 7
- CPU: Intel Core 2 Duo
- 1 GB RAM
- GPU: 256 MB Video Card
- 100MB of space
- 1280x720p Resolution Desktop Screen

The recommended or ideal system requirements for the software solution are:

- OS: Windows 7 or higher
- CPU: Intel Core i3 or higher
- 3 GB RAM or higher
- GPU: Intel HD Graphics or higher
- 100MB of space
- 1280x720p Resolution Desktop Screen

Testing the software solution

Evaluation of success

The success of the software solution is determined by how much of the specifications and criteria laid out by the user and the developer. Comparing the requirements completed to the original requirements are as follows (with the incomplete being red, half complete being orange, and fully completed being green):

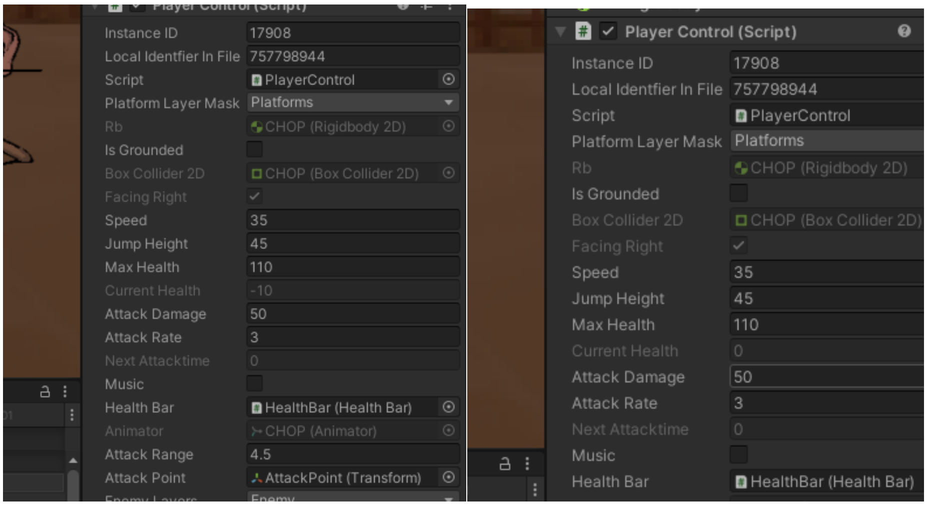
- At least a single stage or level.
- An object-oriented programming language.
- Support for both Mac and Windows.
- Combat with combos that are fun and easy to do.
- Basic left to right movement and jumping.
- A health and damage system.
- A pause menu that freezes the game.
- A start menu to start either the story mode or endless mode.
- An ending with high score.
- A settings menu with ability to change graphics settings, sound volume, blood and music volume.
- Interactive UI.
- Proper hitboxes.
- Time based attacks for the player and enemy.
- At least two characters.
- Original art, graphics and animation.
- Different enemy types to force the player into different strategies.
- A boss battle.
- An endless mode that shows online high scores.
- Audio through sound effects and music.

Judging by the large number of the criteria being completed, with even less being half complete or even incomplete, this means that the project is a success. All of the major game mechanics are available in the game, with the incomplete requirements having very little impact on the enjoyment of the game. The final evaluation of success however will be the evaluation of the client of the software, and the mark given for the whole major itself.

Solution testing at all levels

To check if my desk checks at the beginning were correct, I checked through gameplay if the variables changed with the expected output.

First I checked the health desk check. First when dying, I removed the function that set the health to 0 when its equal to or less than 0, to see if it would actually go under when dead. I saw the value go to -10 when the game over screen appeared, then when I added the function back in again, I saw the health at 0 at the game over screen. I tested this twice in case it was a fluke, but it still came up as 0.



Player Control (Script)

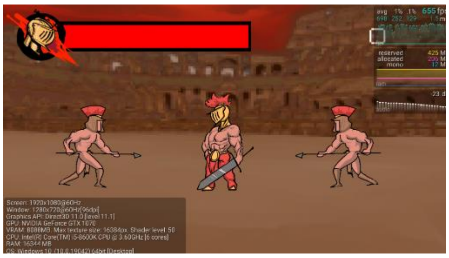
Instance ID	17908
Local Identfier In File	757798944
Script	PlayerControl
Platform Layer Mask	Platforms
Rb	CHOP (Rigidbody 2D)
Is Grounded	□
Box Collider 2D	CHOP (Box Collider 2D)
Facing Right	✓
Speed	35
Jump Height	45
Max Health	110
Current Health	-10
Attack Damage	50
Attack Rate	3
Next Attacktime	0
Music	□
Health Bar	HealthBar (Health Bar)
Animator	CHOP (Animator)
Attack Range	4.5
Attack Point	AttackPoint (Transform)
Enemy Layers	Enemy

✓ Player Control (Script)

Instance ID	17908
Local Identfier In File	757798944
Script	PlayerControl
Platform Layer Mask	Platforms
Rb	CHOP (Rigidbody 2D)
Is Grounded	□
Box Collider 2D	CHOP (Box Collider 2D)
Facing Right	✓
Speed	35
Jump Height	45
Max Health	110
Current Health	0
Attack Damage	50
Attack Rate	3
Next Attacktime	0
Music	□
Health Bar	HealthBar (Health Bar)

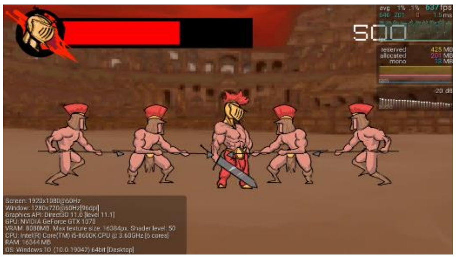
Check the debug mode in the game. The greyed out current health at game over is -10 without the function (left), but 0 with the function (right).

The next desk check I checked in the game was the spawn rate. I changed the value to 2 just like in the test, so it should be much faster. As seen by the amount of enemies at the start of the level to 20 seconds of gameplay, then 40, you see the spawn rate of the enemies increases over time.



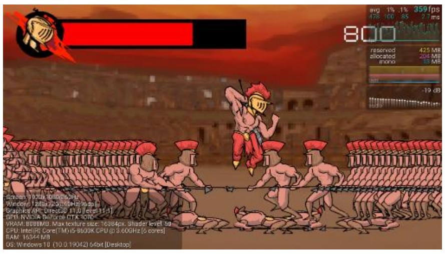
```
Screen: 1920x1080@60Hz
Window: 1280x720@60Hz[96dpi]
Graphics API: Direct3D 11.0 [level 11.1]
GPU: NVIDIA GeForce GTX 1070
VRAM: 8088MB. Max texture size: 16384px. Shader level: 50
CPU: Intel(R) Core(TM) i5-8600K CPU @ 3.60GHz [6 cores]
RAM: 16344 MB
OS: Windows 10 (10.0.19042) 64bit [Desktop]
```

Beginning



```
Screen: 1920x1080@60Hz
Window: 1280x720@60Hz[96dpi]
Graphics API: Direct3D 11.0 [level 11.1]
GPU: NVIDIA GeForce GTX 1070
VRAM: 8088MB. Max texture size: 16384px. Shader level: 50
CPU: Intel(R) Core(TM) i5-8600K CPU @ 3.60GHz [6 cores]
RAM: 16344 MB
OS: Windows 10 (10.0.19042) 64bit [Desktop]
```

20 Seconds in



40 seconds in

The most important test data used for this is the benchmarks from other players. It allowed me to set the limits on the complexity of the game, and thanks to the low requirements for the games maximum functionality, it required no optimisation or simplification in its code.

Post implementation review

Due to issues of communication, a post implementation review of the software solution is unavailable. While I have completed the major project in a reasonable time before the due date, because of issues related to COVID-19 and a sudden lockdown in the week of submission, an appropriate review of the overall software solution by the client was unable to be received.

However, peer reviews of the software solution are available due to the availability of peers, and informal communications through services like Discord which are commonly used between peers of the teenage age group. While a post implementation review from Mr El-Miski is unavailable, the following reviews from fellow peers Mario, Hashem, and Lachlan.

"Pretty epic, and solid. It's a good game, I like the fighting style, and the balancing of the game is good, however, it had no story, and the graphics were so good it should have a story. The game could use more enemies and characters, but everything on its own was very good."

- **Mario**

"Chop was a very energetic and intense game; the inclusion of character selection gave me different gameplay options opting for a good replay-ability factor. The inclusion of different traits for the characters also gave more variety in the gameplay loop."

- **Lachlan**

"The game is aesthetically pleasing and fun to play. Quite a simple game to grasp and understand. The two characters play styles were quite different however and took some time to figure out. Doesn't draw back from the fun I had though. 9/10 on steam, 8.5 on IGN"

- **Hashem**

Based on the peer reviews of the game, I could summarise that the main purpose of the game, its mechanics and its most stand out features are all recognized by the peer reviews. The positives of all reviews sign off on the specified requirements, the custom graphics, different play styles of different characters, and acknowledgement of the endless game mode and its gameplay loop prove that the games core mechanics even to a user outside of the client can be recognized and relate to the developer and client requirements. Even the requirements not met by the software are recognized by the outside users, with their only issues being enemy variety and lack of story. Because of the reviews indirectly recognizing the software requirements, the game has been signed of in lieu of Mr El-Miski himself.

Maintaining the Software Solutions

Changes that could be made in the maintenance of the software

The requirements of the software could change as time passes. A lot of the changes that should be made have been addressed by the missing software requirements and the issues brought up by the peer reviews. The most important possible changes for the software solution are:

- Increasing software functionality.
- Bug fixes.
- Compatibility issues.

To increase software functionality in the game, this means adding more features like different enemy types, more characters, and a story.

For bug fixes, this would include fixing issues like the issue of audio playing wrong during the pause menu (which is why they are removed), and other issues found throughout the game.

For compatibility issues, it would mean updating the game when new Unity updates are released so they are in line and allowing compatibility with Mac or Linux operating systems.

It's important to follow the documentation laid out by this portfolio when looking to add new features, or when locating the changes when maintaining the software solution. This would ensure that new content would not only be compatible, but also what must be changed to make the new content or fixes function correctly.

Documentation is important to change as well when new content is added or bugs are fixed to help maintenance of the software solution in the future, as changes continue to compound over time. It is important to follow the development cycle for newer features, so every feature including the DFDs, data dictionaries, IPO charts, etc. must be updated as the software is maintained and changed. It is also important to change the User Manual and installation guide as well if they are affected by the changes brought up by maintenance.