Team Project Report

Underwurlde

`

Joseph Chotard - jpc844@student.bham.ac.uk - 1935844

Christos Efstathiou - cxe806@student.bham.ac.uk - 1819605

William Matson - wxm856@student.bham.ac.uk - 1875756

Eni Segun - exs802@student.bham.ac.uk - 1885902

Archie Watson - agw828@student.bham.ac.uk - 1914028

Introduction

For this project we have decided to create a top-down action-adventure game. Drawing inspiration from popular games in the genre, such as The Legend of Zelda and Enter the Gungeon, we aim to create a product which will pay homage to these predecessors but also give us scope to try new and creative things with the formula. Typically an action-adventure game of this type sees the player traversing a "dungeon" made up of a sequence of rooms which can each be progressed through by acquiring the required key(s) to unlock the next. With a strong emphasis on exploration and finding creative ways to beat puzzles and enemies, we aim to have captured the spirit of the genre and provided a few alterations to keep things fresh and exciting.

In order to ensure we didn't go off track with this, one of the first things we did as a team was to research the genre and find out what many of the games offered - playing games such as The Legend of Zelda: Link's Awakening on Nintendo Switch and watching Let's Plays of several games on YouTube, we educated ourselves on what makes these games what they are. Over the last term we have created "The Saga of Adlez", here's how we did it.

Requirements

Our requirements are annotated using MoSCoW criteria (**M**ust, **S**hould, **C**ould, **W**ould), in order to display which requirements took priority while developing

	Requirement	Priority
1	**User must be able to control one player character through use of mouse and keyboard**	**M**
1.1	W, A, S, D and space keys must be able to be used to move the player around the map with a constant speed	**M**
1.1.1	The W key will make the player face upwards and move in that direction with vector (0,1)	**M**
1.1.2	The A key will make the player face left and move in that direction with vector (-1,0)	**M**
1.1.3	The S key will make the player face downwards and move in that direction with vector (0,-1)	**M**
1.1.4	The D key will make the player face right and move in that direction with vector (1, 0)	**M**
1.1.5	The Space key should apply a force to the player (equivalent to moving at triple speed) in whatever direction they are facing to create the effect of a “dash”	**S**
1.2	The player will be able to cycle through which weapon they have selected through pressing the Q key	**M**
1.3	The player will be able to reduce the health of enemies according to which weapon they have selected by pressing the E key	**M**
1.4	The player may be able to increase their health up to a maximum value of 500 by pressing the H key and removing 1 healthpack from their inventory	**C**

1.5	The player must be able to pause the game by pressing the Esc key	**M**
1.5.1	In single player the pause button will stop all enemies from moving and attacking and stop any timers being used	**M**
1.5.2	In Multiplayer the host can pause the game for themselves and all connecting players (the same as in single player)	**M**
1.5.3	All Multiplayer clients should not be able to pause the game but should be able to access their individual pause menu the same way	**S**
2	**In Multiplayer at least 1 other player will be able to connect to the host via UDP**	**M**
2.1	The game will use a Client-Server model for multiplayer	**M**
2.2	More that 1 players should be able to connect to the host at once	**S**
2.3	The server will be authoritative and will handle the processing of inputs and enemy/object behaviour which will all then be sent to the clients	**M**
2.4	All singleplayer features of the game will still be accessible in multiplayer	**M**
3	**All Players and Enemies have a health total**	**M**
3.1	Enemies and players will have a health total	**M**
3.2	When an entity's health total is less than or equal to 0 they will die	**M**
3.3	A player may be able to restore their health by consuming a health pack item	**C**
3.4	All attacks will reduce the target entity's health total by a specified amount	**M**

3.4.1	The amount of health reduced should be specified by the weapon used	**S**
3.5	The Player's health total should be displayed in the UI at all times	**S**
3.6	A Boss' health total may be displayed in the UI when a boss is present	**C**
4	**Every Level will have enemies**	**M**
4.1	Enemies have a health total (see 3.1)	**M**
4.1.1	Enemies should have different health totals based on what type of enemy they are	**S**
4.2	Enemies will attempt to follow the player at a constant speed	**M**
4.2.1	Speed may be specified by the type of enemy	**C**
4.3	Enemies will attack the player	**M**
4.3.1	Basic enemies should damage the player on contact with a 2 second cool down time between attacks	**S**
4.3.2	Some enemies may deal more damage based on what type of enemy they are	**C**
4.3.3	Boss enemies should have unique attack types which challenge the player	**S**
4.4	The game must include at least 1 boss at the end of the level - harder enemies which have unique movements and behaviours	**M**

5	**The Game will consist of a series of “rooms”**	**M**
5.1	Every room should be at least 200 x 200 pixels in size	**S**
5.2	Each room will have an entrance and an exit point	**M**
5.3	Every entrance and exit will consist of a door	**M**
5.3.1	A door can be opened when the player has collected enough “keys” from the level	**M**
5.3.2	Doors will act like walls until they are opened	**M**
5.3.3	Once opened, approaching a door will take the player to the next room	**M**
5.4	A level must consist of at least 3 rooms	**M**
5.5	Players should be able to return to previous rooms	**W**
5.6	Normal rooms will contain at least 1 “key” item	**M**
5.7	The final room of a level will be a boss room which contains the boss enemy for the level	**M**
6	**The Game will end and return the player to the main menu**	**M**
6.1	After the boss is defeated a “relic” item will appear at the boss’ location, picking this up will end the game	**M**
6.2	The game must end when all players die	**M**
6.2.1	In multiplayer when a player dies, if other players are alive, they should be able to be revived to 50% health allowing them to keep playing. This is achieved by a teammate standing next to them for a few seconds	**S**

6.2.2	If no players are still alive in any mode, the game will end	**M**
6.3	The game can be exited any time via the pause menu	**M**
7	**There will be items that the Player can pick up**	**M**
7.1	The player must be able to pick up items by walking into them or pressing a button	**M**
7.2	The player should be able to pick up and store several different types of items: - Health packs - heal the player when used with the H key - Keys - used to open doors when the right quantity acquired - Weapons - adds a new weapon to the player - Relic - dropped after boss is defeated, collecting will end the game - Coins - add to the total amount of the in-game currency, it can be spent later on skins through the in-game shop	**M**
7.3	Using certain types of items will remove them from the game world (for example using a key will reduce the number of keys you have by 1)	**M**
8	**The Game will have a clear and comprehensive menu system**	**M**
8.1	The game will open onto a main menu with buttons to start a new game, continue the game, create a multiplayer game, join a multiplayer game, view settings and exit the application	**M**
8.2	The settings menu will allow you to mute game audio, view controls and change game difficulty	**M**
8.2.1	The game may allow you to change the difficulty of the game - changing health and damage totals for various entities	**W**
8.2.2	The game may allow you to change the controls used to play the game	**W**

8.3	All of the necessary screens to navigate the menus and access game modes must exist (Multiplayer host screen, client joining screen, etc)	**M**
8.4	A pause menu will be implemented which allows you to mute the game and exit to the main menu	**M**
9	**The Game must have a UI containing all the information a player needs while playing**	**S**
9.1	The player's health total should be displayed at all times	**S**
9.2	All of the weapons the player currently has should display and it should be clear which one they have selected	**S**
9.3	The number of keys the player has must be displayed	**M**
9.4	The number of health packs the player has should be displayed	**S**
9.5	The game could include a minimap for navigation	**W**
9.6	The number of coins the player has should be displayed during the game and in the Main menu	**S**
10	**The Game will have a soundtrack which plays while the application is open**	**M**
10.1	The soundtrack should be dynamic and change based on the game state	**S**
10.1.1	When in the menus background music will play	**M**
10.1.2	When playing the game in any mode background music will play	**M**
10.1.3	When the player enter combat different music should play	**S**
10.2	The music will be possible to turn on and off through the settings menu	**M**

11	**It will be possible for users to save the game**	**M**
11.1	The game state will be saved into a database	**M**
11.2	When a user returns to the game they will be able to resume by loading the database save	**M**
11.2.1	The user's health will be stored and loaded from the database	**M**
11.2.2	All items the user has will be stored and loaded from the database	**M**
11.2.3	All enemies and objects in the game world will have their properties stored and be loaded from the database	**M**
11.3	The user should have the option to start a new game without loading a save	**S**
12	**The Game must be capable of running on any modern Windows, Linux of Mac OS system**	**M**
12.1	The game should run at 30fps at all times	**S**

Software Design

Architecture

For this project, our main goal was to make the code as modular as possible in order to allow us all to work together and add features as necessary. Of course, we also placed great emphasis on managing the complexity of the project and encouraging reusability. To this end, we chose to adopt a Component Architecture as that allowed us to make independent, replaceable and reusable components. We chose this architecture for its simplicity and its power. It was also the architecture used to make the FXGL library we were using, this allowed for better integration between our code and the library.

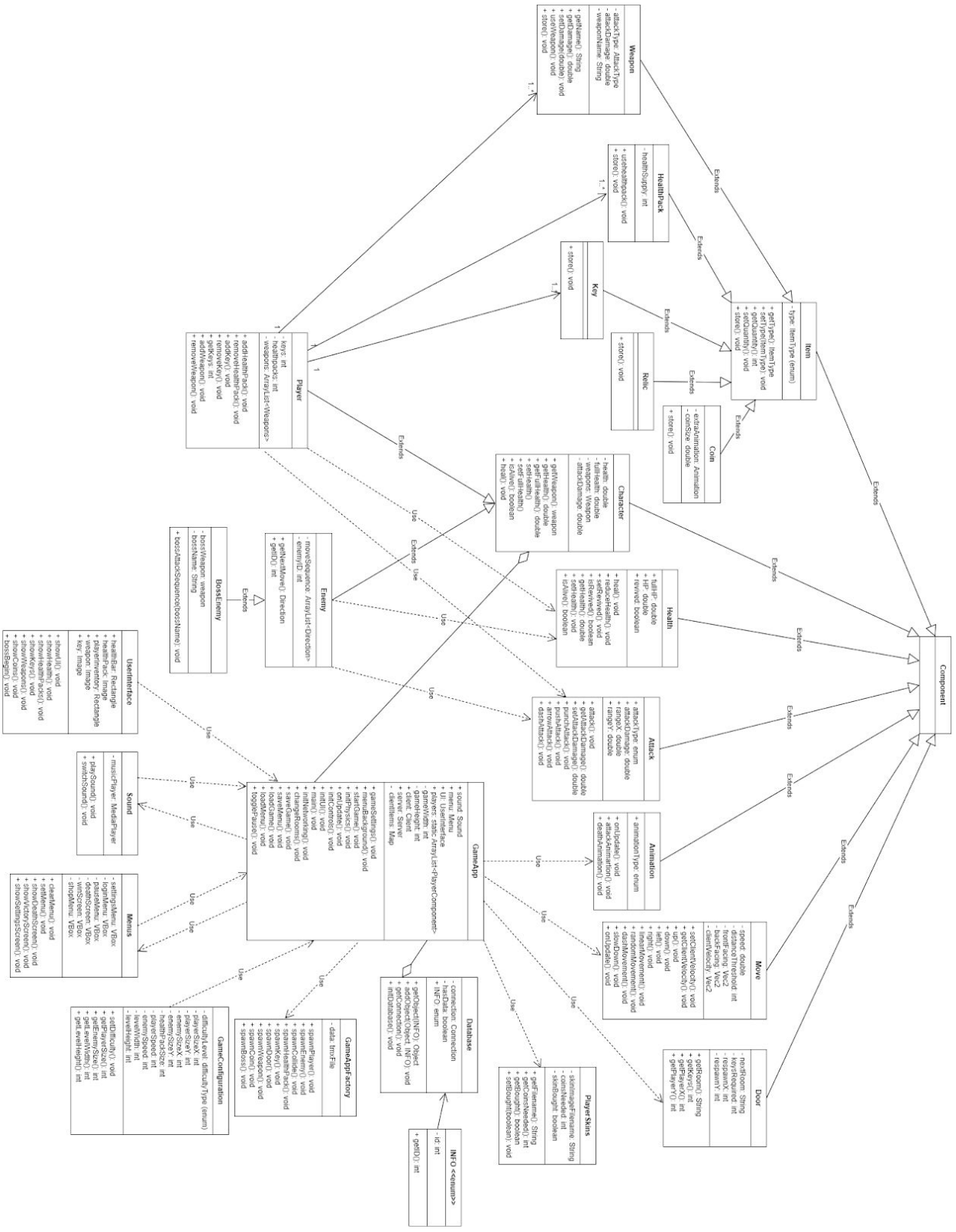
Essentially our game's execution flow is handled by the GameApp class which holds the GameScene which, in turn, holds Entities (Players, Enemies, Healthpacks,..) that are controlled by their individual components: this is where the component-based architecture comes into play. The Entity class is simply there to create the components it is given and update them. For example, the player entity has a PhysicsComponent, PlayerComponent, MoveComponent, AttackComponent, HealthComponent, AnimationComponent, CollidableComponent and a DepthComponent. Each of these components is responsible for a particular ability of the entity.

The only component in that list that is unique to the player entity is the PlayerComponent, because of this, our Component Architecture clearly achieved the goal of being modular and reusable.

All these entities are then rendered onto the view along with the map. To maximise modularity of the game, the .tmx files contain the maps along with the position of the entities. Therefore, maps can be added as needed in the game without the need to change any code or even reload the game as maps are loaded "on the go".

On the next page is a class diagram of our project where we can visually see the modularity behind the components.

Class Diagram

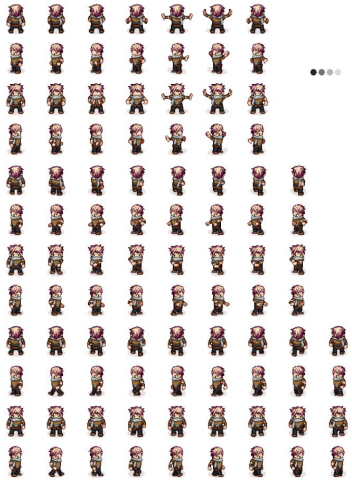


Art Style & Map Design

Throughout our game we have ensured that both the art style and map design represent a consistent approach to creating an overarching theme for our game, which we have chosen to be Medieval Fantasy.

Player Sprites

In order to maintain consistency throughout, we have used the same style of sprite sheet for all player sprites and enemy sprites. For this, a sprite sheet generator has been used (https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator/). We have chosen this method of creating our sprites (as opposed to manually designing them ourselves) because it gives us a high level of customizability, the ability to make numerous sprites very quickly, and ensures that they all conform with specific sprite sizes. Additionally, this generator provides an entire sprite sheet, containing numerous different animations for the character (these are individual images then we then form into animations within our code). These animations in particular would have taken a very long time to design and develop ourselves, and we believe our time was better spent elsewhere. The base player skin and all base enemy skins fit our game theme very well; the player skin illustrates a typical protagonist, with the character in medieval style clothes but also maintaining a look that subtly hints at a mythical persona. Additionally, the enemy sprites all depict fantasy creatures, such as orcs. There are multiple enemy types, each with a different sprite (as well as behaviour), we have done this to ensure that the game does not get stale, as fighting the same enemy over and over again might provide boring gameplay.



An example of a sprite sheet used

The base player sprite

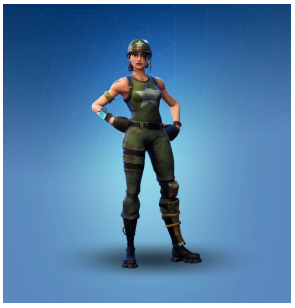
An example of an enemy sprite

We have developed the sprite shop to allow the user to buy new sprites (skins) and change their ingame appearance. This adds to the enjoyment of the game, as players will strive towards unlocking more skins in order to “show off” their new appearances. As a result, while the original player sprite fits the theme well, the more

expensive skins tend to fit this theme less. This is intended, the player needs to feel rewarded for putting in the extra effort to achieve the higher coin balances. A unique and bespoke player skin, one where it is obvious it stands out from the theme of the game, will feel more worthy of this effort than one that fits the theme of the game and therefore doesn't stand out.

A good example of this would be in Fortnite (a game developed by Epic Games). As you can see below, the base player has a scavenger like model, this fits the theme of the game well (looting the map in order to find weapons to kill other players). Within the shop there are also military style models, once again fitting the theme fairly well, which can be bought for relatively cheap prices. However, the more expensive and sought after models (which cost significantly more) are all more unique and bespoke as previously described. A nutcracker costume and an astronaut do not fit the theme of the game, however this means they are more exciting to the user and are therefore willing to spend more money on these skins as opposed to the military ones. We have taken inspiration from this and as a result created a similar experience within our game.

The base skin in fortnite



An example of one of the military skins, which can be bought for a low price

Two of the more expensive skins in fortnite

An example of the player sprites available in the shop, the more expensive skins are the more unorthodox ones

The two boss enemies in the game have specific sprites which aim to reflect their abilities and personality. The Fallen skin shows a “corrupted” version of the player, with the skin being identical to the base player skin except with a much darker tone. This fits the character because it’s abilities are identical to the user, except more powerful. For example, The Fallen can shoot arrows, just like the player, except it can shoot three at once (and more spread out when on less than 50% health). Gilgamesh alternatively has full golden armor and this is designed to portray strength and therefore intimidate the user. This portrayal of strength is reflected in the character’s abilities, with it having a long sword, with which it can deal a large amount of damage to the player.

The sprite used for The Fallen

The sprite used for Gilgamesh

In-Game Item Pickups

Within the game, we have added the ability to pick up certain items while traversing maps, specifically health packs, weapons, keys and coins. The key requirement of the art style of these sprites was for them to make it absolutely clear to the user what the sprite refers to, with this being a higher priority to us than fitting the theme of the game well (which we still believe they do). For example, the health pack has a medical cross on it, and therefore it should be obvious to the user what this is. For weapon pickups we have not assigned a specific sprite, instead deciding to simply visualise them as a red square. We have chosen to do this because there is only one weapon pickup within the game, and as such it would be difficult to source a sprite

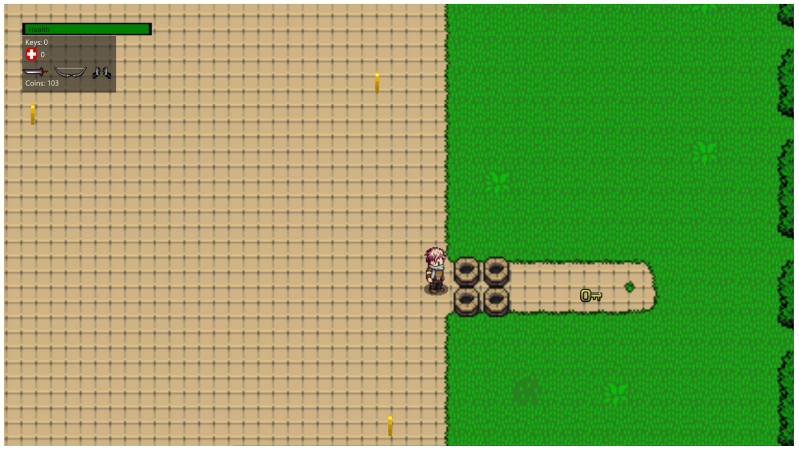
image which refers to the specific weapon. We could have created an endless number of different weapons (for example multiple bow types which deal different damage), however we did not believe that this was a good use of our time; our time was better used elsewhere. As such, we have only one weapon pickup in order to demonstrate that this function is possible, and easy to implement if we were to add more weapons to the game.

Examples of the sprites used for in-game pickups

Map Design

Within our game we have developed maps with a number of crucial requirements: the maps must have a main goal (which is to find the keys scattered around the map in order to progress to the next level); they must be challenging; they must fit with the theme of the game; and each one must be unique.

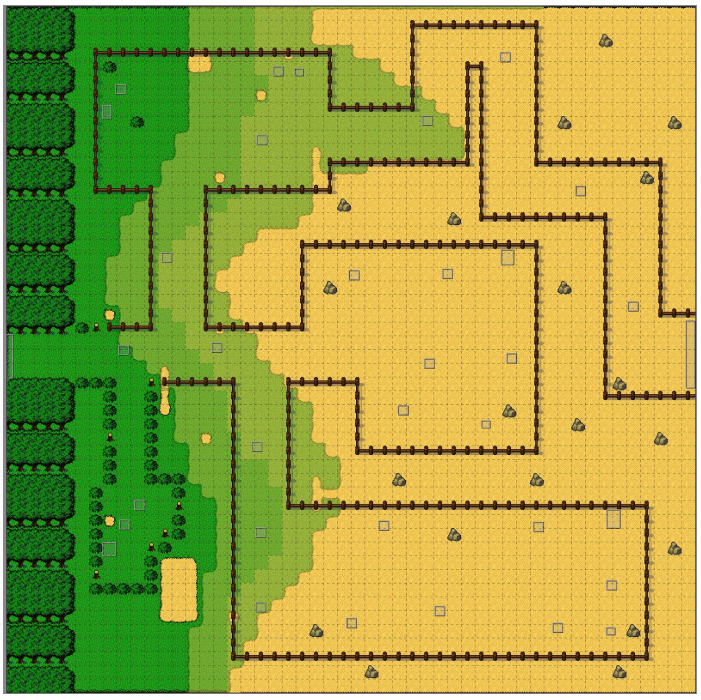
We believe the maps we have developed meet all these criteria points. All the maps contain keys, often in discrete locations that ensure the player has to explore most of the playable area. For example, this key below is located behind a barrier, however the user has to figure out that this barrier is avoided if you dash over it. Furthermore, these maps are littered with enemies for the player to fight and defeat. This, along with the challenge of finding some keys, ensures these maps are challenging.



An example of a hard-to-obtain key

The overall style of these maps ensures that there is a sense of uniformity, none of the maps contain any elements which don't fit in with this overall style. The maps have been designed so that moving between them is a clean transition. The player begins in a grassland biome for the first two "rooms", they then move into a transition room, where the ground becomes drier, and eventually turns into a desert biome. Following this, the player encounters the end of the desert, and reaches a beach with which it becomes the final biome, which is a water biome. We believe that this

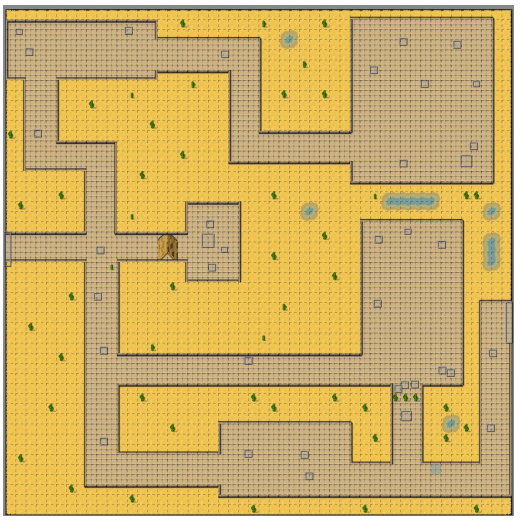
smooth transition between biomes is much more visually appealing and more suited to the game than moving between two completely different looking maps. This method ensures that playing through the game feels more like a story.



An example of how maps provide a smooth transition between different biomes

The actual playable area of the maps is mixed, in certain areas it is more open (such as the first level and the start of the beach level) whereas in other areas it is more restricted, with halls and rooms making it more confined. This mix of areas ensures that the gameplay doesn't become stale, as constantly moving through dungeons can be tedious. Furthermore, the final boss fight is located within a tight room. This ensures that the player is forced to confront the boss directly, as they are unable to escape or move far out of the range of this boss, also demonstrating that our rooms are implemented such that their size is completely dynamic.

The more confined areas of our maps take inspiration from numerous games, including Pokemon Mystery Dungeon (developed by Spike Chunsoft). Our maps share the same concept as these, where there are narrow lanes and larger rooms to discover items in. These types of games all have a similar style of dungeon, and therefore we followed this industry standard.



As you can see, our maps (right) follow a similar principle to the maps in Pokemon Mystery Dungeon (left, the white outline shows the overall map).

Sprite	Sprite Source locations
Player and enemy sprite sheets	https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator/#
Coins	https://free-game-assets.itch.io/free-game-coins-sprite
Arrows used in the shop	https://cdn.pixabay.com/photo/2020/02/11/07/54/arrows-4838663_1280.png
Tilesheet	https://opengameart.org/content/tinyslates-16x16px-orthogonal-tileset-by-ivan-voirol
Healthpack	http://pixelartmaker.com/art/2ede25037e61e96
Key	https://opengameart.org/content/golden-and-silver-key-game-item

Interface

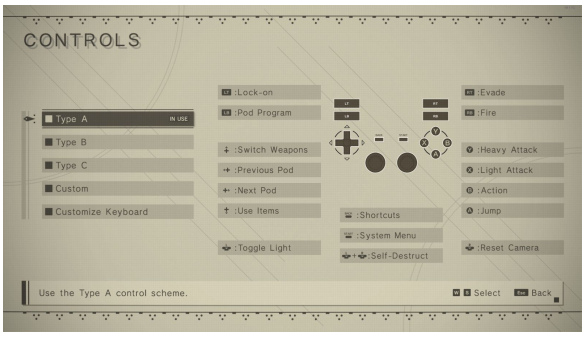
Our interface targets simplicity and ease of use while maintaining a cohesive look and feel - everything should be clear what it means and where you might find it. We have drawn inspiration from how other games layout their menus and UI in an attempt to follow industry best practices in order to give a level of familiarity to the interface.

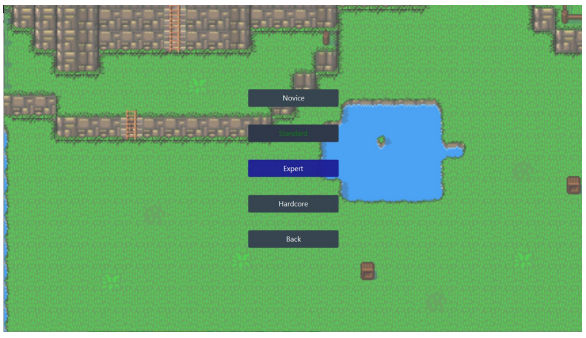
Menus

One of the main principles of HCI is that the interface should be consistent. In order to comply with this principle all of our menus follow the same visual style - all buttons look alike, the same font is used throughout the game and the same effects occur when interacting with menu items. In addition to the visual design's consistency, the naming of buttons and menus is also kept the same throughout (no screen is ever called 2 different things) such that navigation is easy.

Taking an example from NieR: Automata (A game by Platinum Games and Square Enix), looking at the menu of the game we can immediately see some examples of how a good menu should be laid out.

As can be seen from NieR (left), every item on the menu has a clear name which makes its function obvious. Notice also the effect of the underline on the continue button (where the mouse is) in order to give the user visual feedback of which option they are currently selecting and indicating that it can be clicked. As for our game (right), we have followed a similar concept where when you hover over a button on the menu it will darken and the text will become green to give some visual feedback to the user.





Continuing with the example of NieR: Automata, this screen on the left is accessible in the game's settings. We have taken similar inspiration from this to provide even more visual feedback to the user. This screen demonstrates that when selecting out of a list of options a message saying "IN USE" displays - we have used a similar concept for our Difficulty menu in the game settings where whichever difficulty mode you have selected, the relevant button will highlight blue to indicate the choice as shown in the image of our game (right).

In-Game HUD

In terms of the in game HUD we have designed this to echo how other games implement their UI with a clear focus on corners and edges such as to obstruct the gameplay itself as little as possible. In the left image of Link's Awakening we can see the health in the top left and items in the top right. We have followed a similar pattern where health is in the top left hand corner but in order to keep the UI concise (and because we have more screen to use than a Nintendo Switch) we have also put the counters of other UI elements in the top left as well to indicate what items the player has as well - this has been done as then the player need only look in one location to find all of the information they need with the goal of distracting less from gameplay. For ease of use we have included the words health and keys (the 2 most key quantities) in the UI in order to make it clear what they are referring to. In the case of "health packs" we have also used the same icon as the health pack pickup to indicate their quantity in the UI which should make things more clear to the user.
In design we opted to make the UI behave dynamically to give more visual cues to the user based on the current state of the game. Here is a screenshot from later into the game:

In this screenshot you can immediately gather various bits of information from the UI just at a glance. In order to give more visual feedback when important events occur a few UI items change, for instance a boss health bar and name gets displayed when there is a boss on the screen in this case “Gilgamesh”. You can also see that the player is on very low health as the health bar in the top left has turned red giving a good contrast and indicating that the UI wants the user’s attention now. Finally, different to the previous screenshot, you can see that there are more weapons in the player’s inventory as the bar has expanded and a new icon has appeared - switching between them indicates which weapon is selected at any time by colouring in that weapon, when a weapon is not in use its icon is dimmed and becomes black and white.

Software Engineering

When brainstorming the project we came up with a few software engineering principles we would adhere to in order to speed up the development process and improve cooperation between members of the team. The first principle we focussed on was the single responsibility principle: each class and function only has one clear responsibility which means that our classes aren't overloaded with functionality and can be reused in many different situations. This also means that our code is a lot more readable which is something we're looking for when we could potentially all be using the same classes/functions.

Another important principle we followed was the Liskov substitution principle in that every child class can replace the parent class if need be. This was important to us simply for the sake of code readability which really was one of our main goals.

Finally, to make our code as modular and extensible as possible we tried, as much as possible, to follow the Dependency Inversion Principle in order to abstract low-level features such as Database manipulation from the save feature. This would allow us to, should we need to, change from SQLite to MySQL or even a NoSQL database such as firebase. Of course, this also applies to network or graphical functionality. Not only does this allow modularity it also means we can delegate tasks better and split the workload more evenly.

Finally, the last principle we followed wasn't related to our software itself but to our meetings. We decided that we should meet, in person, at least once a week. This was important to us because we wanted to make sure we were all on the same page regarding what we were working on and what we wanted to focus on. With the school shutting down, we couldn't meet in person over the last couple of weeks but we still made sure to attend a group call through Discord at least once a week to finalise the report and test the game.

Risk Analysis

During this project we had to take, and overcome, some risks associated with our software design approach, as well as our development methodology. For instance, the Component Architecture we used was a great idea initially, since everyone could work on their own individual component classes, but then problems occurred later on when trying to make all of the components work with each other. The Attack Component, for example, needed the other entities to have the Health Component, since attacking would cause the other entities to reduce their HP - this issue was noticed when entities began attacking walls, which of course have no HealthComponent. We overcame these problems, by making sure the attack method only affected the surrounding entities which actually had the Health Component.

Another risk we had to take was with the data transfer over network. The protocol that was used to exchange data between machines was UDP, which allowed us to transfer large amounts of data, but without any security. Anyone could probably intercept the connection quite easily, but since no important user information is exchanged, no harm could really be done. Additionally, by choosing UDP over TCP we could transfer more data faster, but there was no guarantee that the data would reach its destination. The trade off for ease and speed here outweighed the potential risks for our purposes.

Consequently, the single responsibility principle we used had many benefits in our software engineering architecture, however, we ended up with a lot of small classes with not a lot of code in them. This kept individual classes cleaner and easier to read, but on the other hand it spread out the classes and created many unnecessary connections, sometimes leading into “spaghetti” code. This was a product of trying to create a very simple design to begin with, leading to redesign of certain components before their implementation in order to adapt to the current state of the code base.

Additionally, we had to mitigate the risk of feature creep throughout the development of the game. Feature creep refers to a scenario where, as people get more invested in the game, they become more likely to want to add features above and beyond the original scope of the project, which consumes significant amounts of development time. We managed this by ensuring that all features we added were outlined by the initial requirements and if we wanted to add anything more we would have to meet as a team and weigh up if it was worth the time cost of adding it to the product.

Finally, another major risk with the product was time management and the individual availability of each team member. As the project had tight time constraints, and everyone had other important commitments, it was hard to allocate a clear time plan for the project in advance. In addition to this, any time scales we created would be “best case” and not necessarily representative of reality or what we should expect. Fortunately, we managed to get the product completed within schedule, however several factors did hamper our original time plans over the course of development, not least the current COVID-19 crisis.

Evaluation

After finishing the project, we have created an amazing and enjoyable top-down dungeon adventure game, which was based on other famous games. It has some really nice textures, that make it look more lively and the gameplay is very simple, so that anyone can play it. Not only is it simple to play, but it is also quite interesting, with the variety of enemies and bosses you can fight, as well as the adventurous levels you can explore. We believe that the game turned out better than what we initially expected.

Strengths

The biggest strength of our game is its extensibility. We can add or update the textures, the maps and even the entities whenever we want, that means we can have newer enemies with new textures and abilities, bosses with different fighting styles and more maps to explore. Furthermore, the game can be a bit challenging, but that keeps it interesting and also entertaining. If a player is stuck on a difficult level, their friends can join and help them progress. In addition to this, the shop we included in the game is definitely another reason why people will keep playing it, so that they can collect their favourite skins. We also feel that one of the greatest strengths of the project has been succeeding in creating what we feel is a rather ambitious game with a lot of features and depth to it.

Weaknesses

Even though the game has many strong points, it also has some weaker ones. For example, the Clients who connect to the Host player, their gameplay is not as smooth as the Host's. It is still playable, but there is definitely some small noticeable delay, especially when you compare them next to each other. Moreover, the game does not cope well when the network connection is not reliable. For instance, if a player disconnects due to a bad connection, the game might randomly crash on that player's machine, but the host and the other players can continue playing normally. The enemies' pathing could also be improved, we experimented with an A* algorithm and it did work but the performance hit was too great as our map is not grid based so there were a lot of routes to process when you had multiple enemies in the map - this, of course, did not make it to the final version of the game.

What has been learned?

During the developing phase of our project, we have all learnt a lot of new things that might be useful in the future. To be more precise, Git was one of the things we struggled with at first, but we slowly all learnt how to use it effectively to complete the game. Additionally, we all worked on some areas we had no experience in, like networking and databases, but after doing a lot of research we managed to complete them. Furthermore, we definitely gained a lot of knowledge in FXGL's library, due to the fact that we used it quite a bit in order to render our game.

What could have been done better?

After completing the game and having careful consideration for some of our earlier decisions and the knowledge we gained throughout the project, we found some areas that could have been done more efficiently. Multiplayer, for example, could have been implemented very differently. Instead of having the Clients sending their key inputs to the Host and having the Host do all the calculations, we could make the Client do some calculations as well, such as player movement and picking up items. This would undoubtedly make the gameplay more smooth on the Client side, but it would be more complicated to implement, since the Host and the Client would constantly be exchanging entity data, which could cause some data clashing. We also would, with more time, have liked to create multiple levels for players to choose from rather than just 1 long level, but this would not be difficult to implement at all, all that would change is the menu would need some more buttons to load you onto a different starting map.

Summary

In conclusion, we believe this project has been a great success as we have succeeded in our goal of creating the game which we wanted to create and quite an ambitious one at that. We started out with establishing a clear idea in our minds and then through hours of hard work on each of our sections and collaboratively polishing and putting it all together we have managed to make the final product which we wanted to make, capturing the essence of the games which inspired it - true to our vision and objectives.

The game we have produced feels polished and is the product of every team member's skills and creativity. It achieves the effect of being a simple game to understand and play while fulfilling all of the requirements of the task and still achieving a lot of depth. From the varied enemy types, boss fights and reviving your friends in battle to the pickups and ability to buy new skins for your player character, we have tried to capture a lot of attention to detail within the final product and hope that it shows.

"The Saga of Adlez" has been nothing short of a joy to make and, hopefully, is equally as enjoyable to play.

Teamwork Evaluation

During the development of this game we have successfully worked well as a team, and we believe this has been instrumental to the completion of the project. We decided from the beginning not to set strict roles between us, and instead decided to be more relaxed in deciding what tasks individuals completed. The network side was the one exception, as Christos was the lead developer in regards to this portion of the game.

Strengths

One key strength of this task is the fact we didn't set strict roles. Apart from networking, as previously stated, we split certain tasks between us based on individual elements, as opposed to overall categories. For example, William took charge of developing the main menu interface, however we did not assign him to just develop all user interface elements. This ensured that the sections of code we wrote did not feel too disjointed from one another. As we all contributed to certain areas of the code we are all confident in each individual section. This, in our opinion, increased efficiency, as we did not have to wait for other members to complete code for us to continue with our own development. The development of the shop is a good example of this, Archie took charge of developing this feature. This feature required significant menu interface development in order to implement, and he was able to implement these features himself, in such a way that it fits uniformly into the original menu interface, instead of waiting for William to do it (which would have been the case if William had been assigned specifically to create all user interface features).

Additionally, communication throughout the team was successful, and helped us ensure everyone knew what tasks were currently in development and those that needed to be implemented. We created a Trello board which allowed us to monitor progress. Every task was placed onto this board and individual members could assign themselves to these tasks. They could then update the group of their progress through the comment feature; once completed these tasks were placed in the "review" section, and then finally the "complete" section. As a result, every member of the group could easily see everything currently in development, and avoid working on the same thing as someone else by accident. Furthermore, we maintained regular meetings either once or twice a week in order to keep everyone up to date on our progress. These were very helpful, and often we used these as an opportunity to merge all our code together.

Weaknesses

One potential weakness of our teamwork is that, as we did not have distinct roles, we found it more difficult to keep a clean git repository. Each major task within the development had its own branch, however as multiple people may have been working on the same task at the same time, these branches weren't really used by individual people. Instead multiple people may have been working in the same branch at the same time, and this on a few occasions created some difficult merge conflicts that could not wait until a group meeting to deal with. It may have been easier for us to have our own branches that were separate from one another, as this would have caused less frequent conflicts.

Teamwork Conclusion

Overall, we believe we worked very well as a team, and were we to develop another game we would follow the same development path. While there were certain issues with the way dealt with development, the benefits far outweigh these negatives and as a result allowed us to create a comprehensive and successful game.

Test Report

Test Plan

To test the networking of our game, we decided to test individual features and compare our observations to the expected result. There is a table that logged the outcomes of the network testing so we could ensure that the features that the game was supposed to implement were still accessible in multiplayer mode.

To test the wider features of the game we performed User Testing, using a questionnaire to evaluate the user experience and overall usability of the game. The questions we asked allowed us to make sure that the requirements that we outlined at the start of the project were still met by the final product.

Throughout development, we also implemented various debugging tests to ensure that methods we added gave us our expected outcomes. This coincided with one of the primary principles we followed for development using git - no features should be merged into the dev or master branches if they have not been thoroughly tested to ensure we would not be introducing any new issues to the existing code base.

Testing through development

Throughout development we strived to ensure that everything we implemented was fully tested before it made its way into our Dev branch such that no problems would be introduced if someone else was to start on a new feature and needed a clean Dev branch to start off - this greatly reduced problems we had to solve down the line.

One approach to testing could have been through the use of JUnit tests, however, we decided that these would slow down development time by taking away from the time we would be working on features. Given the nature of a game, it was quite easy to come up with a strategy to test features, as and when they were added, which could be done by eye rather than implementing test methods.

The general process here was that while developing the first version of a feature, extensive debugging should be used in order to check every value that the program sees to see if they are what should be expected, where applicable apply conditional blocks in order to validate the values. This ensured that as we were creating any given feature we would have a clear view of what was going on within the methods we used and could easily see the impact on the wider code base. Once this step is complete any unnecessary logging and print statements should be stripped away and comments should be created in order to explain the functionality.

Next came playtesting, when making a game it is imperative that the feature works in terms of gameplay and feels good to a user. The principle of this testing takes an abstract view of the system as a whole and puts it into practice - the way that it will actually be used in reality. Where a feature has a direct dependency on another feature, they should be tested in collaboration with each other in order to ensure that neither causes an unexpected behaviour in the other.

An example of this in practice would be for the saving feature of the game. This feature saves data about the game world to a database so throughout development, debugging was done to ensure that the database was getting the right information and the right information was being read from the database. When this was done, the feature was integrated with the Menus of the game where when a player quits to menu from a single-player game or a game in which they are the host the data is saved to the database. In addition to this, when selecting to start a game from the main menu the player receives the option of a New Game or Continuing from their save - this is also true for multiplayer. In order to check this worked as intended several things were done in sequence:

1. Create a game New Game, move to a new location and quit to the main menu
2. Continue the game, are you in the location you were in before?
3. Move again then quit the game to menu
4. Continue the game from the main menu, are you still in a new position?
5. Quit the game to the main menu
6. Start a New Game, have you reset to the original location?
7. Pick up some items in the game and stop where some enemies are on screen, quit to the main menu
8. Continue the game from the main menu, are the enemies in the same location? Do you still have your items?
9. Quit to the main menu again
10. Start a New Game, have you reset completely again?
11. Move then quit to the main menu
12. Continue the game as a multiplayer host, are you in the same location?
13. Move and quit the game
14. Continue from single-player this time, is the save still intact?
15. Quit the game and start a new game as a host in multiplayer, have you reset?
16. Quit to the main menu, join a multiplayer game as a client
17. Move and then quit to the main menu again
18. Continue in single-player or multiplayer - did the game save before? (Expected behaviour is to not save a client's game)
19. Start a new game, move and then quit the application
20. Open the game, continue in single player, is everything as it was before you quit the application?
21. Quit to the main menu
22. Start a new game, head into a new room
23. Quit to the main menu
24. Continue the game, are you in the new room? Does everything look right? Are all the enemies and items where they were before? Do you still have your items?
25. Quit to the main menu
26. Start a New Game, has everything reset?

This is an example of one of the more complex cases that we have. Once all of these checks have been carried out and you are confident that the feature fulfils the expected behaviour it is ready to be tested by another member of the team who should check out the branch and test the feature until they are also satisfied that it works. When this is done, the branch is ready to be merged - if the feature is big, wait until next Merge Wednesday so the team can work together on it.

Following this methodology, we ensured that features were developed quickly and achieved high standards of quality before entering the core codebase.

User Testing

We asked 3 individuals a series of questions so we could find out how well the features we implemented were received by possible users of the game. The people we interviewed were novice game players, one of which has played a few mobile games and one used to own a Nintendo DS when younger, as well as someone who plays games frequently at an amateur level with friends online.

Test Results

Feature Tested	Question Asked	Response
Gameplay	On a scale of 1-5, how likely are you to play this game again in your free time?	Novice 1 – 4 Novice 2 – 5 Amateur – 3
	On a scale of 1-5, how likely are you to recommend this game to a friend?	Novice 1 – 4 Novice 2 – 5 Amateur – 4
	On a scale of 1-5, how easy was it to understand the rules/aim of the game?	Novice 1 – 5 Novice 2 – 5 Amateur – 5
	On a scale of 1-5, how would you rate your overall experience?	Novice 1 – 5 Novice 2 – 4 Amateur – 4
	On a scale of 1-5, how engaging was the game?	Novice 1 – 5 Novice 2 – 5 Amateur – 4
	Is there anything you would improve about the game?	Novice 1 – more levels/rooms Novice 2 – no, it's fine as is Amateur – more complex levels with boss rooms in between, time limit for each room or whole game
User Interface	On a scale of 1-5, how easy was it to use the controls?	Novice 1 – 5 Novice 2 – 5 Amateur – 5

Animations & Graphics	On a scale of 1-5, how clear/easy is it to see the different objects in the map?	Novice 1 – 5 Novice 2 – 4, the item dropped by the boss could use a bit more contrast to the map it's in Amateur – 5
	On a scale of 1-5, how smooth are the player animations?	Novice 1 – 5 Novice 2 – 5 Amateur – 4, seen better but thought it's very good still
	On a scale of 1-5, how smooth are the enemy animations?	Novice 1 – 5 Novice 2 – 5 Amateur – 4, same as above
	On a scale of 1-5, how appealing and smooth are the room transitions?	Novice 1 – 5 Novice 2 – 5 Amateur – 5
	Have you experienced any issues with the animations in the game?	Novice 1 – no Novice 2 – no Amateur – no
	On a scale of 1-5, how clear were the graphics in the game? Did you experience any pixelation of images/characters?	Novice 1 – 5, no Novice 2 – 5, no Amateur – 5, no
Audio Integration	On a scale of 1-5, how appealing are the sounds that are integrated in the game?	Novice 1 – 4, the music is a little repetitive at points but it works Novice 2 – 5 Amateur – 4, would like some sound for attacks
	On a scale of 1-5, how easy was it to turn off the music in the game?	Novice 1 – 5 Novice 2 – 5 Amateur – 5
Menu Navigation	On a scale of 1-5, how easy were the menu icons to use?	Novice 1 – 5 Novice 2 – 5 Amateur – 5
	On a scale of 1-5, how easy was the menu to navigate?	Novice 1 – 5 Novice 2 – 5 Amateur – 5

Multiplayer Connectivity	On a scale of 1-5, how easy was it to join or create a multiplayer game?	Novice 1 – 5 Novice 2 – 4, would be nice to have a lobbying system Amateur – 5
	Did you experience a broken connection between players while engaging in multiplayer gameplay?	Novice 1 – no Novice 2 – yes, unexpected disconnect Amateur – no
	Did you experience any lagging that disrupted the game while engaging in multiplayer gameplay?	Novice 1 – no Novice 2 – no Amateur – yes
Database	Did you experience any issues resuming the game after returning to the main menu?	Novice 1 - no Novice 2 - no Amateur - no

Network Testing

We have conducted extensive network testing on all scenarios which can arise and their impacts on the game. This ranges from the various game creation and joining scenarios to the more visual side of what is actually seen by each player.

Test Results

Testing Method	**Expected Result**	**Actual Result**
Create a new Multiplayer game and have 1 Client join the Game.	Game creation is successful and the Client can join and play successfully.	Game creation is successful and the Client can join and play successfully.
Create a new Multiplayer game, change room and then have 1 Client join the Game.	Game creation is successful and the Client can join and spawn in the same room as the Host.	Game creation is successful and the Client can join and spawn in the same room as the Host.
Create a new Multiplayer game and have 3 Clients join the Game.	Game creation is successful and all Clients can join and play successfully.	Game creation is successful and all Clients can join and play successfully.
In a Multiplayer game with multiple clients, each client can move their own character through the normal control scheme and it will update for all other players	Player will press W, A, S, D and Space keys and movement will occur as expected from single player games	Player will press W, A, S, D and Space keys and movement will occur as expected from single player games
In a Multiplayer game with multiple clients, each client can attack enemies	Player will press the E key with their equipped weapon near an enemy, the animation for the attack plays and the enemy's health decreases - the enemy is also pushed back	Player will press the E key with their equipped weapon near an enemy, the animation for the attack plays and the enemy's health decreases - the enemy is also pushed back
In a Multiplayer game with multiple clients, each player can take damage	A player approaches an enemy and within a second they see their health bar decrease in size	A player approaches an enemy and within a second they see their health bar decrease in size
When joining a Multiplayer game, all entities on the	The Client joins the host's game and can see the	The Client joins the host's game and can see the

server are loaded for the client	healthpack, weapon and key items - enemies spawn in the map and move correctly as per their positions on the server	healthpack, weapon and key items - enemies spawn in the map and move correctly as per their positions on the server
In a Multiplayer game with multiple clients, players can pick up items correctly	A player picks up the key item and all players see their number of keys increase and the sprite disappear. A player picks up the health pack, all players see the sprite disappear and that player sees their number of health packs increase, the health pack can be used by pressing H. A player picks up the weapon, all players see the sprite disappear and that player has the weapon in their inventory which can be switched to by pressing Q and used by pressing E	A player picks up the key item and all players see their number of keys increase and the sprite disappear. A player picks up the health pack, all players see the sprite disappear and that player sees their number of health packs increase, the health pack can be used by pressing H. A player picks up the weapon, all players see the sprite disappear and that player has the weapon in their inventory which can be switched to by pressing Q and used by pressing E
In a Multiplayer game with multiple Clients, check all the Clients can see all the animations (i.e. walking, shooting bow, attacking, boss animations, arrows)	Animations play on all the machines' screens.	Animations play on all the machines' screens.
In a Multiplayer game with multiple Clients the Host player changes room.	All players spawn in the next room and keep their items.	All players spawn in the next room and keep their items.
In a Multiplayer game with multiple Clients, 1 player changes room.	All players spawn in the next room and keep their items.	All players spawn in the next room and keep their items.
In a Multiplayer game with multiple Clients, the Host pauses the game.	All Clients get the "Player 1 Paused the Game" menu screen, they cannot move and all the entities stop moving in the background.	All Clients get the "Player 1 Paused the Game" menu screen, they cannot move and all the entities stop moving in the background.
In a Multiplayer game with multiple Clients, the Host	All Clients get Player 1 Paused the Game" menu	All Clients get Player 1 Paused the Game" menu

pauses the game and then unpauses.	screen and then it is removed and they can move again and all the entities are active again.	screen and then it is removed and they can move again and all the entities are active again.
In a Multiplayer game with multiple Clients, a Client pauses the game.	Just that Client gets a pause menu screen, but everything runs normally in the Background and all the other players can still play.	Just that Client gets a pause menu screen, but everything runs normally in the Background and all the other players can still play.
In a Multiplayer game with multiple Clients, a Client disconnects.	Client goes to the Main Menu and all the other machines see the Player disappear and can still continue playing the game normally.	Client goes to the Main Menu and all the other machines see the Player disappear and can still continue playing the game normally.
In a Multiplayer game with multiple Clients, the Host ends the game.	Host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background.	Host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background.
In a Multiplayer game with multiple Clients, a Client disconnects and then reconnects.	Client can reconnect, but they are a new Player with no items and full HP.	Client can reconnect, but they are a new Player with no items and full HP.
In a Multiplayer game with multiple Clients, the Host ends the game, then starts it again and all the Clients join the game.	The Host goes to the Main Menu, then creates a new Multiplayer game successfully. All the other Clients connect and play successfully, but as new players with no items and full HP.	The Host goes to the Main Menu, then creates a new Multiplayer game successfully. All the other Clients connect and play successfully, but as new players with no items and full HP.
In a Multiplayer game with multiple Clients, 1 player dies.	That player kneels down on everyone's screen and gets the “You have died” menu screen on their screen and they cannot move, but all the entities are still moving in the background. The enemies will stop following this player.	That player kneels down on everyone's screen and gets the “You have died” menu screen on their screen and they cannot move, but all the entities are still moving in the background. The enemies will stop following this player.

In a Multiplayer game with multiple Clients, a player dies (player A) and then another player (player B) approaches them for a few seconds.	Player A kneels down on everyone's screen and gets the “You have died” menu screen on their screen and they cannot move, but all the entities are still moving in the background. After a few seconds of player B standing near player A, Player A’s death screen disappears and the player can move again with half the HP of their full HP. Player A also stands up on all the machines’ screens. The enemies start following this player again if they are in range.	Player A kneels down on everyone's screen and gets the “You have died” menu screen on their screen and they cannot move, but all the entities are still moving in the background. After a few seconds of player B standing near player A, Player A’s death screen disappears and the player can move again with half the HP of their full HP. Player A also stands up on all the machines’ screens. The enemies start following this player again if they are in range.
In a Multiplayer game with multiple Clients, all players die.	All the players get the “All Players are Dead” menu screen and the game is over.	All the players get the “All Players are Dead” menu screen and the game is over.
In a Multiplayer game with multiple Clients, the Host ends the game and then presses Continue game.	The host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background. The Host continues the game, in the same position, with the same HP and items.	The host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background. The Host continues the game, in the same position, with the same HP and items.
In a Multiplayer game with multiple Clients, the Host ends the game and then presses Start New game.	The host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background. The Host starts a new game on the first level with no items.	The host goes to the Main Menu and all the Clients get the “Server Ended” Menu screen and all the entities stop moving in the background. The Host starts a new game on the first level with no items.

Problems Encountered

- Sometimes a Client might randomly crash, especially if the connection is not reliable, particularly when changing rooms. That player can still reconnect, but they are going to get a new player and the old player stays in the background not moving.
- There is definitely a noticeable delay on the Client side lending to the fact that everything is processed on the server.

Appendix

This Appendix will detail the coding standards and practices our team has adopted over the course of this project.

Variables

We have ensured that all of the variables in this project are named unambiguously and clearly indicate what they will do from the name at a glance. They should all be named using camel case, starting with a lowercase letter (for example gameActive and totalEnemyAttackTime). There are a few exceptions to this rule where it makes more sense to use another convention, for instance the app HEIGHT and WIDTH constants are named in uppercase.

Variables should also not be global unless the benefit of them being global outweighs all other factors - for example core features like UI which are declared and initialized in GameApp are set to public as they are used widely and there is largely no drawback to making them public, whereas the properties of entities should only be possible to edit via the appropriate getters and setters or other methods in that class.

All variables should be used for one purpose alone - it can be confusing and cause errors if one variable is used for one thing and then another sometime later.

Methods

Method names should be written in camel case, starting with a lowercase letter (for example togglePause() and clearEntityLists()). A getter's name should begin with the word get, a setter's name should begin with the word set.

Methods should be named appropriately based on what they do, it should be obvious from a glance what the method does without needing much external context. Where code would be duplicated a method should be made instead if it makes sense to do so.

Methods should be private or protected by default, a method can be public when it is:

- A getter
- A setter
- Functionality of that class which is called elsewhere

Classes

Class names should start with an uppercase letter, if the name is made up of multiple words then inner words should also start with uppercase letters. Generally class names will be nouns which simply describe what the class is. Classes should have 1 clear purpose and not deviate from that purpose - for example UI handles the User Interface, Menus handles the game Menu, GameAppFactory is only for spawning entities and GameConfig is for global properties of the game.

Components should be named in a similar convention to classes. Typically they will consist of 2 or 3 word names where the final word is always Component to indicate they are components. Each Component should have only related functionality within them which makes sense for all entities incorporating that component. For instance MoveComponent handles movement of all entities, ArrowComponent handles the behaviour of arrow entities specifically and the BossComponent extends EnemyComponent and contains the behaviours of each of the bosses. Components should be extended like this when there is functionality which only specific variants of the entities incorporating that component need - for example SwordProjComponent extends ArrowComponent with behaviours unique to Sword Projection attacks.

Other Code Related

- There will be no use of GOTO statements
- Proper indentation should be used at all times for readability
- Function arguments should be separated with both a comma and a space following it
- Errors should be prevented without throwing and catching exceptions wherever possible, exceptions to this include dealing with asynchronous methods and network errors

Documentation

All methods should be appropriately documented. As a minimum a method should have JavaDoc for its parameters and return types with their names. Getters and setters may keep minimal documentation unless their usage must be specified or they perform some calculation in addition to simply returning or setting a value. More complex methods should have documentation briefly explaining their usage and behaviour as a minimum with the most complex functions requiring a proper explanation.

Git Usage

New features being worked on should be separated into new branches off of the latest version of dev. All changes must be pushed to the remote repository often so that all members of the team have access to your code as soon as it is ready. No code should be added to Dev or Master when it contains any known errors to avoid damaging the code base.

When merging, if/when conflicts arise, seek the person who made code you are overwriting to ensure that you are not introducing new issues. We used the concept of “Merge Wednesdays” as a team so that any big merges we had to do in a week would all be ready for our Wednesday meeting where we were all there to work through issues together.

Master should only be pushed to for fully working iterations of the product.

Personal Reflections

Joseph Chotard's Reflection

This project has been an overall very positive experience which I really enjoyed from start to finish. I think one of the factors which contributed the most to making this game an enjoyable project to work on is the fact that my teammates were all extremely friendly and hard working. I really feel like we agreed on all our ideas and put in equal amounts of work which made for an enjoyable project. Not only was everyone contributing equally in the work, but we also all took part in regular meetings where we all contributed ideas towards the development and helped each other with issues that we might have encountered. An example of this teamwork is our "Merge Wednesdays" where we all met up to help each other merge tested branches into dev.

With this game, I've learned a lot of techniques and tools that will be invaluable in my studies and future career in technology. For example, I use to use git regularly however it was generally on individual or 2 person projects and this was the first time I was really exposed to more complicated merges or rebases, I now feel a lot more confident using git. I also learned a lot about java and software project management with maven. I had to learn how to build jar files as we had a few problems with the pre-built version of the library we were using. This means I had to learn about more complex dependency management withing maven. As I was responsible for database integration I learned a lot about SQL and no-SQL databases and how different database drivers work in Java, I eventually settled on SQLite as it was the easiest to use with an embedded database and fit our needs.

Throughout this project, I was responsible for various features. Most notably, the map loading and how to specify entity positions within maps. I also implemented the view which follows the player around along with movable physics components like the crates. Another big feature I worked on was the database integration that I worked on with Christos: my idea was to reuse a lot of the networking code but instead of getting the information from the network to load it from the database, so I hid the database driver behind some abstraction to make the network code reusable. I worked on other odd features here and there helping out whenever others needed help. Once again coming back to my first paragraph, I really feel like everyone helped each other out on different features which means we all have quite a good grasp about how most of the game's features work.

All of these factors combined is why I really enjoyed working on this interesting project with some equally interesting and hard-working teammates. I really feel like I'm submitting this project knowing the inner working of every aspect of the code thanks to the rest of the team.

On a final note, I would like to thank Tom for helping us out throughout the project by giving us ideas and encouraging us through his enthusiasm and support.

Christos Efstathiou's Reflection

This project was probably the most fascinating assignment I had so far during my studies and I had a lot of fun doing it. From the start, I could tell that my teammates were all very smart and excited for this project. We came up with many clever ideas for our game and we actually designed the architecture quite quickly, which meant we had more time to work on the implementation face. Everyone in the team was very organised and we always kept in contact to discuss important details about our game. I have no complaints about my teammates, they were all amazing and we all gave our best efforts on this assignment.

My personal contribution was mostly focused on implementing and taking care of the network. I chose this requirement for the project, because I thought it was important to start it as soon as possible. Initially, I spent about two weeks researching and trying out simple ways to exchange data over UDP between two local instances of the game. I decided to go with a Client-Server architecture, where all the calculations happened on the Server and the Client just sent over the input key presses. This was the easiest way to implement multiplayer and since I had no experience in networking, I did not want to overcomplicate our code and not have a working game by the end of the project. For the following weeks I mostly helped my teammates with some small stuff, which I would later need to find a way to implement them in multiplayer. Overall, I am very happy with the way multiplayer works.

Some of the most challenging stuff I had to work, included sending over animations from the Server to the Client. Since the entities on the Client were only changing HP and position, they did not have any walking or attacking animations. For this part my teammate William helped me a lot and after one week, we found an easy and efficient way to send over animations.

Towards the end of the assignment, I helped with the room transitions and database saving and loading methods. Since I was using similar methods to store the entities' data for the network, I was the perfect person to do this task. I knew exactly how and what to store, as well as how to reconstruct the entities using their respective data.

In conclusion, it was a very fascinating and exciting project to work on. I met and worked with some amazing people and I made some new friendships. We have also managed to create a fantastic game, which definitely exceeded my expectations.

William Matson's Reflection

I have enjoyed this team project on the whole, working with my team has been great and honestly I couldn't have asked for a better group to work with. After meeting for the first time we very quickly all got onto the same page and knew what we wanted to create. Everyone in the team has been ready and willing to take responsibility for their own areas within the product but also not afraid to ask each other for help where needed.

From doing this project I learned a lot about working on developing in a team, one way in which we best handled this was through use of a Trello board which was incredibly useful for keeping track of what still needed doing and what various team members were working on. I also have become a lot more experienced with usage of git over the course of the project through having to deal with branches and merging which is relatively new for me. I also learned a lot more about how to manage a large project in Java using various packages and libraries as well as working with Maven.

My personal role in the team has been a bit of everything. Initially a lot of my role was, as the person most familiar with the games we were drawing inspiration from, to aid the idea process and direct our ideas. I then started to work on sound and UI in order to try and create some core aspects of the experience for a new user picking up the game and to create the feel of other games in the genre.

After finishing up on that I started doing various bits of work across almost all of the different areas of the game. I implemented a lot of the framework for animations but then began, my favourite part of the whole project, making boss fights. I created the first boss pair programming with Archie as he made a lot of the component which handles attacks so it made sense that we would want to make some updates to that area as well while creating the new, challenging enemies. I then went on to create the second boss more independently, I would have liked to get round to making a third boss but other areas of the game took priority.

My role then effectively for the last few weeks became adding (and fixing) essential features and polishing aspects of the game such as creating the end game condition, improving the in-game HUD and pair programming with Christos a lot to make the multiplayer portion of the game feel more polished.

To sum up, I have enjoyed working on this team and the opportunity to work closely with the other members of the team on a large variety of areas has been fantastic as I have been able to become very familiar with all aspects of our game.

Eni Segun's Reflection

The development of a game was something that seemed very daunting at the beginning of the project, however I have thoroughly enjoyed working with every member of the team over the past 11 weeks to create a game that we all envisioned at the start. I feel like the team worked well because we all agreed on what we wanted to create, and we were happy to implement ideas that individuals had about extra features to make the game more fun and interesting. The use of Trello to keep track of what features we wanted to implement and how far each part of the game was in terms of development was extremely useful. We also used Facebook Messenger to keep up with each other on a day-to-day basis if anyone had made further developments or had any issues. The use of these platforms helped me learn the importance of keeping every member of the team updated on current developments, which helps improve efficiency and reinforces the importance of communication between all of us.

I have learnt a lot from all of the team members throughout the project, including more about git repositories and the use of different branches and the git timeline, as well as managing lots of classes in a large java project in a way that keeps them well integrated. I also gained a lot of knowledge on FXGL and how it is used to create projects and games.

As for my personal contribution to the game, I started with looking into the different designs for the maps that were used to create the levels. We used tmx files for the maps, as this was well integrated into FXGL, so I looked into different designs for the rooms and how we could have different objects and paths throughout the map. After this I also moved onto working on transitioning between rooms. I had found this quite challenging to begin with, but with the help of my team we were able to integrate the room transitions into the game in a way that allowed for a lot more flexibility and just worked better overall. I also worked on putting keys in each room and creating more map designs, which could have been used as an extra feature that changed rooms depending on the difficulty selected and even allowing players to design their own rooms to play in the game if we had the time to do so, as discussed during our week 6 prototype demonstration.

Looking back on how I could have improved, I think I would have liked to push myself more out of my comfort zone and been more confident in terms of exploring different aspects of the game and working on the development of them, as well as sharing more research that was done into different areas of the project that were not related to the levels and rooms that were created and used. However, I have learnt so much and enjoyed working with everyone in my team so much that I am sure I would be able to apply what I have learnt to future projects and would like to look into developing more games to improve and expand the range of skills I have learnt.

Archie Watson's Reflection

The development of this game has been a fascinating experience for me, one that I have thoroughly enjoyed. My programming skills and abilities have certainly improved as a result.

A key skill that I have been able to develop throughout this assignment is teamwork. I have never used tools such as Trello before, and rarely have I been required to take part in weekly meetings. Throughout this project I continually updated the areas of the Trello board I was responsible for. This was particularly useful when reviewing work; I would place my tasks into the "review" section and await feedback (while at the same time reviewing other's code). These also provided ample opportunity to merge our code together through Git. I have never used Git before, and as a result of this assignment I have significantly improved my ability to use this tool.

As a team, I believe we all worked really well together. Every member of the team worked hard to ensure we developed a well-polished game, with all aspects of it developed to a high standard. Everyone contributed significantly to the success of this game, and I could not have asked for any more from any individual.

In regards to my own role within the team, I primarily took charge of a significant portion of the game logic, while also getting involved in other areas of the game later on in the development. Initially, I developed an A* algorithm for enemy pathfinding, along with a "partial" variant. While neither of these ended up in the final build, the movement component we eventually used took significant inspiration from these. From here I began to focus the majority of my effort into the attack and health features of the game, developing different attack types (such as using a sword and shooting arrows with a bow) as well as their intended effects to player and enemy health. I enjoyed this area of the project significantly, and it felt very rewarding to program these elements into the game. In addition to this, I also thoroughly enjoyed the challenge of co-developing the first boss fight alongside Will, with which I primarily dealt with the unique attacks that the boss had.

Developing the ingame shop, as well as a handful of the maps on offer, allowed me to get a bit creative, which is slightly out of my comfort zone. I believe I created some nice looking, and well planned, maps which are entertaining to play and fit the theme of the game well. I also believe the shop I developed works well, and it was fun to create the skins to go along with it. Other features I developed include implementing coins and health packs, and some UI features like the difficulty menu.

Overall, I believe that the development of this game was very successful, and that I contributed significantly to that. I have been able to develop my skills throughout and am now in a much stronger position if I were to develop more games in the future.