NEA

Name: Jez Snelson
Candidate Number: 1209
Centre Number: 62337

Contents

1 Analysis

1.1 Dungeon Crawlers

A dungeon crawl is a scenario in role playing games in which the main character navigates a dungeon environment often solving traps or fighting monsters to progress through the level. A video game or board game made up of predominantly dungeon crawls is considered to be a dungeon crawler.

Most dungeon crawlers have a fixed map that is the same every time which can lead to little replay value as it can be boring to replay the same map over and over.

1.2 The Problem

Dungeon Crawler style games can be boring and repetitive, this means they can have little to none replay value. Additionaly alot of Dungeon crawlers have a steep learning curve that makes it hard for new or casual players to fully enjoy them. These games are also very complex often demanding lots of time for a simple playthrough. In addition, Non-Computational Methods are inconvenient as they can take up alot of space, take a long time to set up and you cannot save your game state to pick it up later easily.

1.3 Stakeholders

1.3.1 Survey

I chose a set of questions in order to survey my stakeholders and help me find success criteria for the project to fulfill their needs.
1. How often would you say you play video games on a scale of 1-10 (1 being every other week 10 being every day)
2. Do you have any specific or requirements for this computer game?
3. How would you use this game?
4. Would you say you have the time to commit to learning a complex or unintuitive game?(yes,probably not,no)
5. How long would you say is your average gaming session?(1-5 hours)
6. Which different ways do you play video games?(multiple choice: controller, wasd, arrows)
7. Have you played any Dungeon Crawler games(e.g. Legend of Zelda, Binding of Isaac, Dead Cells, Hades)?
8. If not would you want to try a Dungeon Crawler Game?
9. Rank the features of classic dungeon crawlers you dislike the most(Lack of Replayability, Long Unskippable Cinematics, High length of time required for a playing session, The Learning Curve, The Difficulty)
10. Rank the features you think are most essential for the game to be enjoyable for you(Procedurally Generated Dungeons, Loot to Collect and utilise, Some Sort of skill tree, Co-Op mode, Puzzles, Hidden Areas)

1.3.2 Survey Results

Time available:
On average my stakeholders session length is around 2 hours for a single game. On average they play videogames almost every day however there is one that plays infrequently. Because of this I will have to try and make it easy to pick up without much you have to remember about previous sessions.
Most of my stakeholders do not have time to commit to learning a complex or unintuitive game and so I will have to make the game easy to pick up but still have complexities for those who want a challenge.
All controlling mechanisms where popular but WASD was the most so I will prioritise that.
50% of my stakeholders have played dungeon crawlers and so may be experienced with it but 50% have not so I should aim to make it a good introduction to the dungeon crawler genre with the potential of adding optional difficulty for those more experienced.

Disliked Features (Ranked most to least disliked):
1. Lack of replayability.
2. High length of time required for a playing session.
3. The Learning Curve.
4. Long Unskippable Cinematics.
5. The Difficulty.

Due to this I will focus on replayability through the use of procedural generation whilst still aiming to exclude the more disliked features.

Liked Features (Ranked from most to least liked):
1. Some sort of skill tree.
2. Hidden Areas
3. Procedurally Generated Dungeons.
4. Loot to collect and utilise (e.g. weapons).
5. Puzzles.
6. Co-Op Mode.

Because of this I will prioritise getting the more liked features done and exclude some of the less liked features from my success criteria.

1.3.3 About Stakeholders

Name	Description	How they will use my product
Samuel Vanderstelt-Hook	18 year old Male Sixth Form Computer Science Student, Casual Gamer who enjoys a wide range of games.	Sam will use my solution for casual gaming for fun as a break from his studies. He has stated needs for a game that is replayable and gives him a reason to come back to it.
Daniel Olde Scheper	18 year old Male A Level Computer Science Student	Daniel will use my solution as a way to relax from his A-Level Studies. He has stated needs for a fun, replayable and easy to pick up game.
Peter Dunn	17 year old Male College Student and aspiring hobbyist game developer.	Peter will use my solution as a form of entertainment after studies and as he loves Dungeon Crawl Style games. He needs a replayable game with an intuitive combat system.
Sadiya Shorkar	17 year old Female Student and Casual Video Game Enjoyer	Sadiya will use my solution as a form of casual entertainment for short sessions. Sadiya has seizures and so needs accessibility options like volume control and options for less vibrancy.
Penelope Castiau	18 year old Female Sixth Form Student, Avid Computer Gaming Enjoyer and Hobbyist Streamer.	Penny will use my product for entertainment purposes and to play on stream. Because of this Penny needs subtitles to make the game easy to follow for viewers.
Steff Stylianos	17 year old Female College Student and Game Developer	Steff will use my product to relax from studies. Steff needs a replayable game but also want it to be engaging.

1.4 Research

1.4.1 Existing Solutions

Edmund McMillen's The Binding of Isaac
Edmund McMillen created the popular dungeon crawler roguelike The Binding of Isaac and released it on Steam$_{(1)}$. This game was relatively unique as it had procedurally generated dungeons
using a system of rooms that tesalate with each other.

The procedurally generated dungeons consist of different shaped square based rooms that tesalate and are generated next to each other in a psuedo random fashion whilst obeying a set of rules. The mobs that spawn in each room can vary but there is usually only one or two enemy types per room and as you go up levels the amount of enemies and difficulty the pose increases. This system allows for every playthrough of the game to be different to the next with the same reccuring theme/difficulty which allows for lots of replay oppurtunity. This would be an appropiate way for me to fix the replayabilty issue.

I like the games simple UI design as it clearly indicates all the necessary parts. The Map also shows the basic stucture of the level without revealing too much.

However, the game has a couple issues that mean that it does not completely solve our problem. First is the steep learning curve that the game presents which, although to some is a welcome challenge, can put off new or less experienced players especially due to its roguelike nature meaning when you die you start from scratch. The game also has an unintuitive movement and fighting system as there is only really quad directional projectiles and a simple walking design which when combined contributes to the steep learning curve.

Figure 1: A screenshot of The Binding of Isaac UI and Map

Motion Twin's Dead Cells
Motion Twin created the roguelike dungeon crawler and metroidvania Dead Cells which is released on steam[(2)]. This game is known for its permadeath system and its procedurally generated dungeons.

The way Dead Cells uses procedural generation interests me as it allows for there to be some fixed attributes to the level whilst still allowing elements of randomness. The developers talk about how they do this in a video devlog[(3)], here the dev talks about his system of having a fixed structure for each level almost like a skeleton. This skeleton will include stuff like important rooms along the way and how much distance of rooms has to be between them. It then fills in all the spaces for rooms with one of the many handmade rooms made by the developers. After one room has been chosen for a spot this leaves less choice for the other spots as the rooms need to join and flow into each other properly and so as it chooses more of them the structure of the level is determined similar to the wave function collapse algorithm[(4)].

This style of generation allows for a unique experience each time whilst keeping a hand crafted and natural feel to the levels that is often lost in other techniques.

However due to the game being aimed at more hardcore gamers with it being part of the rouguelike genre it can often appear complex and offputting to newer players who dont like the idea of taking multiple runs just to have very little to show for it and not much forward progress in the game. Although the game is a side on game I think that I will use the idea of its procedural generation as inspiration in my product.

Nintendo's Legend Of Zelda Breath of the Wild
Nintendo created the open-world dungeon crawler which is released on the Nintendo Wii U and the Nintendo Switch$_{(5)}$. This game is known for its open world approach to dungeon crawlers as well as its easy to pick up nature for first time players.

The game starts with a tutorial that teaches players the mechanics of the game (combat, exploration, and resource gathering). This tutorial helps players into the world without overwhelming them, offering opportunities to learn at their own pace which helps reduce the steep learning curve of other games in the genre. The open-world nature of the game also adds to its replayability, allowing the player to take many different routes to complete the game. However, while the game's size and allows for alot of replayabilty, the volume of content and time required to explore everything can reduce its effectiveness as a game that can be picked up easily for shorter sessions. Its 3D world and complex systems are features that would be too tricky to implement within the scope of an A-level computer science project. It also does not fully fit the dungeon-crawler genre, particularly as it is less dungeon-focused.

I want to take inspiration from the open-world nature of the game to increase replayability aswell as its approach to tutorials in order to make the learning curve steeper. Ontop of this another feature I would like to take inspiration from is the intuitiveness of the combat system which is easy to learn but hard to master in particular its feature of being able to lock onto enemies.

Some features I will not be including are the 3D nature and the overall content heaviness aswell as the focus less on dungeon crawling as I believe these would be unnecesary features which would drive up the complexity of the solution both to make and run.

1.5 Limitations and Requirements

Requirement	Description	Justification
Hardware	PC or laptop with a Keyboard or Game Controller, minimum of 4GB RAM. For Windows/Linux: x86_32 CPU with SSE2 instructions, any x86_64 CPU, ARMv8 CPU. For Macos: x86_64 or ARM CPU. Integrated graphics with full OpenGL 3.3 support	These are the requirements for running an executable from Godot. The keyboard(WASD) or controller is needed as the input for the game.
Software	I will be using the Godot Game Engine and GDScript to program my game.	I will be using Godot as it is a good 2D game designer that is Free and Open-Source it changes less often than alternatives such as Unity. Ontop of this I have prior experience in Godot and GDScript.
OS Limitations	For Native Exports: Windows 7 or newer, macOS 10.13 or newer, Linux distribution released after 2016 For Web: Firefox 79, Chrome 68, Edge 79, Safari 15.2, Opera 64	Godot can export easily to any of these platforms and more accessibility is good and I can also export a HTML5 version to be hosted in a website such as https://www.itch.io.
General System Limitations	A visually or auditory excellent experience	I do not have the experience with shaders or music and sound effects to add these features to the game in this time and it would make the game requirements higher.

1.6 Features

1.6.1 Essential Features

Feature#	Feature	Description	Justification
1	Player Movement and Controls	The player will control movement using the WASD keys for up, left, down and right respectively. Alternatively they will use the left control stick of a controller.	This will be used tp navigate around the Dungeon environment and WASD was the most popular control mechanism for the stakeholders with controller close behind. I will also include mouse buttons as a non-mandatory addition.
2	A Basic Combat System	The combat system will consist of a primary weapon (melee, magic or ranged) on mouse-1/1 key/X button and a sheild or secondary weapon on mouse-2/2 key/Y button. I will have to implement projectiles and hitboxes for both the player and enemies.	A basic combat system is essential as it will provide the main difficulty and entertainment within the game.
3	Dungeon Environment	The Dungeon Environment will consist of different shaped rooms with different purposes(e.g. boss room, chest room and shop room.) with hallways connecting inbetween them and a starting room.	A Dungeon Environment is essential as it is the environment the player will play in.
4	Different Enemies	The Enemies will consist of a variety of enemies that attack the player with different patterns and have different looks and animations.	This is essential as it will add variety to the gameplay and each enemy will provide a challenge to the player.
5	Appearance and Animations of the Player	The Player will have a recognisable appearance aswell as animations for all its actions such as walking and fighting	This is essential as it lets you know where your character is on screen aswell as giving life to the actions the player is performing.
6	Login System	Users will be able to login in order to save and reload their progress. The login system will use a username and password with the details being encrypted and stored in an external database. Their will be options for signing in or creating a new account aswell as resetting your password.	This is an essential feature as saving progress is essential for making the game replayable.
7	User Interface	A Simple UI that shows status indicators like health, weapons being used, enemy health and magic points.	This would allow the player to be aware of the characters health and give them the necessary information.

1.6.2 Desireable Features

Feature#	Feature	Description	Justification
8	Weapons and a more Advanced Combat System.	A system of weapons where you can get them from boss drops and potentially shops and a combat system with normal, charged (based on how long you hold down) and special attacks (using a special key).	Different weapons will allow each player to have a playstyle more customized to them and will allow for the player getting stronger as they progress more. An advanced combat system will allow for a more smooth and enjoyable fighting experience.
9	Skill Tree	A skill tree to unlock unique skills/abilities and get better at using existing skills/weapons. You would gain points from playing the game and can then put them into different areas in order to create a customized character build	This would further allow the player to choose their own play style and add an element of replayability where you can try going for a different build each time you play. This was also requested by the stakeholders.
10	Procedurally Generated Dungeons	The Dungeons would be procedurally generated whilst keeping some amount of structure (e.g. the same amount of distance between posses and key rooms). This would happen through many similar small room sections that can be slotted together in order to make a full dungeon.	This would create a more engaging game which is different each time you play it and therefore increase replayability exponentially as the different combinations of room increases. This was also requested by the stakeholders.
11	Hidden Areas	Secret areas that can be unlocked through wasy such as progressing further in the game and coming back or through puzzles/fake walls. Could have secret loot or bosses.	This feature was highly requested by the stakeholders and would allow for more time spent having fun in the game through finding these areas.
12	Inventory Sysetm	An Inventory to be opened with the E key or the + button through which you will manage equipped weapons, key items, skills and more.	An Inventory System is an essential feature if we want to add more weapons/weapon types and a skill tree.
13	Settings and Volume Control	A settings page to control the volume of noises aswell as the vibrancy of colours.	One of the Stakeholders has requested this as a feature to help the game be more accessible to them.
14	Difficulty Levels and Hardcore Mode	A Difficulty level selector which allows the user to up the difficulty(damage the enemies do etc) and a Hardcore Mode which switches the game to a roguelike format with seperate save state to the normal game.	50% of the stakeholders are experienced with Dungeon Crawlers so in order to help the game still be reasonably challening for them I will add a difficulty toggle.

1.7 Success Criteria

Criteria #	Abstraction	Success Criteria	Justification
1	Players to be able to control and move the player using both the WASD keys and a controller.	1.1 W key - Forward 1.2 A key - Left 1.3 S key - Backward 1.4 D key - Right 1.5 Q key - Dash 1.6 Left Control Stick directional movement corresponds to player movement.	These Criteria need to be met for the character to be controllable by the player. These specific controls where preffered by the stakeholders.
2	Players to be able to have different weapons and attack with them.	2.1 mouse-1/1 key/X button - Primary Attack 2.2 mouse-2/2 key/Y button - Secondary Attack 2.3 Add a basic melee sword 2.4 Add a basic ranged bow and projectiles 2.5 Add a basic magic staff and projectiles 2.6 Add a basic magic staff with area of effect attacks 2.7 Add a hitbox for the player 2.8 Add a health bar for the player 2.9 Add the ability to lock facing an enemy 2.10 Make sure all attacks go in the correct direction	These criteria need to be met for a basic combat system to create the main difficulty and entertainment throughout the game
3	A Dungeon environment for the character to walk around and different rooms	3.1 Walls that you cannot walk through 3.2 Floor of the Dungeon 3.3 Interactive chests for loot 3.4 Seperate Boss, Chest, Monster and Shop Rooms 3.5 A room Door that only opens on a certain condition 3.6 A Dungeon Environment built out of the rooms and corridors	These Criteria will provide the environment within which the game is played.
4	Different Enemies for the player to face including bosses	4.1 Enemy Sprites 4.2 Enemy Pathfinding Abilities 4.3 Enemy sight range 4.4 Enemy hitbox 4.5 Enemy health tracking 4.6 Melee Enemies 4.7 Projectile Enemies 4.8 Boss Enemies with different attack combinations	These Criterie need to be met in order to provide enemies in order to provide the challenge throughout the game.

5	Appearance and Animations of the Player	5.1 Player Sprite 5.2 Walking Animation 5.3 Player sprite turns to face the direction of movement 5.4 Melee Animation 5.5 Magic Animation 5.6 Bow Animation	These Criteria will provide the visual animations so the player knows where they are attacking and moving
6	Login System	6.1 Password Hashing Algorithm 6.2 SQL Table to store username and hashed password pairs 6.3 Ability to create a new account with unique username 6.4 Validation of Usernames ($1 \leq chars < 15$) 6.5 Validation of passwords (One Special Character, One Number, At least 8 Characters, One upper and lower case character) 6.6 Input Sanitisation (Removing any escape chars for SQL before sending the command) 6.7 Ability to log in with an exisiting account and correct password 6.8 Ability to reset password (If admin account) 6.9 A general login form which links the other forms	These Criteria will provide a Login system in order to allow multiple users to login whilst keeping user data secure and accessible by that user.
7	User Interface	7.1 Health Bar 7.2 Magic Points Bar 7.3 Display of the weapon being used 7.4 Popup display with enemy health over their head when they get damaged 7.5 ability to switch between weapons	These Criteria allow the key information to be displayed to the user aswell as the user being able to change what weapon they have equipped

8	Weapons And a More Advanced Combat System	8.1 Different Styles of melee, magic and ranged weapons 8.2 Item pick up 8.3 Boss Drops 8.4 Shop System that appears throughout levels 8.5 Charged Attacks (based on how long you hold down) 8.6 Special attacks	These Criteria would allow for a more complex combat system to increase differences in the way you play
9	Skill Tree	9.1 UI Menu for the skill tree (Some skills required before others unlocked). 9.2 Different Branches (Melee, Ranged, Magic, Defense) 9.3 Experience system. 9.3.1 Experience gained after killing enemies/bosses 9.3.2 Different experience amounts required for different skills 9.4 Ability to unlock skills 9.5 Ability to reset your skill tree	These Criteria will fulfill a stakeholder desire and will allow the game to have more complexity and replayability
10	Procedurally Generated Dungeons	10.1 Creating requirements for each level to satisfy 10.2 Creating differnt room sections/rooms to peice together 10.3 Creating the algorithm to generate which room sections are slotted together where. 10.4 Create an algorithm to peice the sections together to create a fully playable level. 10.4.1 Level's generated satisfy length requirements 10.4.2 Level's generated contain all the special rooms needed (chest room, secret rooms, etc.)	These criteria will help to add a ton of replayability to the game aswell as being requested by stakeholders
11	Hidden Areas	11.1 Add mechanics to get into the secret rooms (breakable walls, climbing vines, keys, etc.) 11.1.1 Add a hammer to break walls with 11.1.2 Add climbing gloves which you need in order to climb vines 11.2 Add secret Boss and Treasure rooms for behind these obstacles.	These Criteria will make the game more engaging as well as being requested by the stakeholders
12	Inventory System	12.1 UI for Inventory 12.2 Storage of Extra weapons and key items (keys, armour, charms, etc) 12.3 E key to open up the inventory 12.4 Ability to switch out what Weapons, Armour and charms are equipped.	These criteria will allow for more complexity in the game and more customized and diverse playthrough options which allows for more replayability

13	Settings and Volume Control	13.1 Settings UI with buttons for each setting 13.2 Ability to control the volume 13.3 Ability to control the vibrancy of colours in the game.	These criteria have been requested by stakeholders as a way to increase the accessibility of the game to them
14	Difficulty Levels and Hardcore Mode	14.1 A slider for difficulty in the settings menu 14.2 Increasing difficulty based on the slider 14.2.1 Increasing enemy health 14.2.2 Decreasing player health 14.2.3 Increasing number of enemies 14.3 A Hardcore mode at maximum difficulty with a seperate save state to the normal game. 14.3.1 roguelike features (permadeath, resource management, etc)	These criteria will allow those of the stakeholders who have more experience in dungeon crawlers aswell as those that have played the game more to up the difficulty and it increases the replayabilty through different challenges.

1.8 Computational Methods

2 Design

2.1 Overview

2.1.1 Global Variables

I have a couple of main global variable scripts Global, Inventory, Database etc.

Source	Identifier	Data Type	Justification

2.1.2 Folder Structure

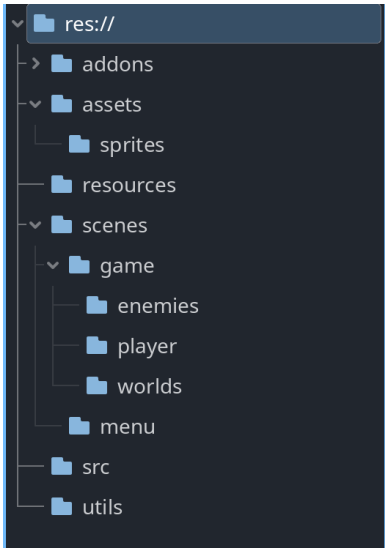

Figure 2: Folder Structure

I chose this folder structure as it will allow me to clearly define where all the different parts of the game are aswell as easily being able to access the closely related parts.
The assets folder will contain all of the external assets, sprites, spritesheets and audio.
The resources folder will contain all of the items (weapons, armour, keys and charms) that I will make to be included in the game.
The scenes folder will contain all the scenes for the menu and the game sorted into their respective folders.
The src folder will contain all of the preloaded scripts for the game.
the utils folder will contain any testing or debugging scripts/scenes to help with the development process.

2.1.3 Naming Convention

For naming I conventions I will adopt the naming conventions already used in godot for ease of integration, readability and consistency with documentation.
The naming conventions are as follows.

Type	Convention	Info
File Names	snake_case	yaml_parsed.gd
Class Names	PascalCase	YAMLParser
Node Names	PascalCase	
Functions	snake_case	
Variables	snake_case	
Signals	snake_case	Past tense ”door_opened”
Constants	CONSTANT_CASE	

2.2 Database Design

I will be using an SQL Database in order to store the data about my users.

2.2.1 ERD

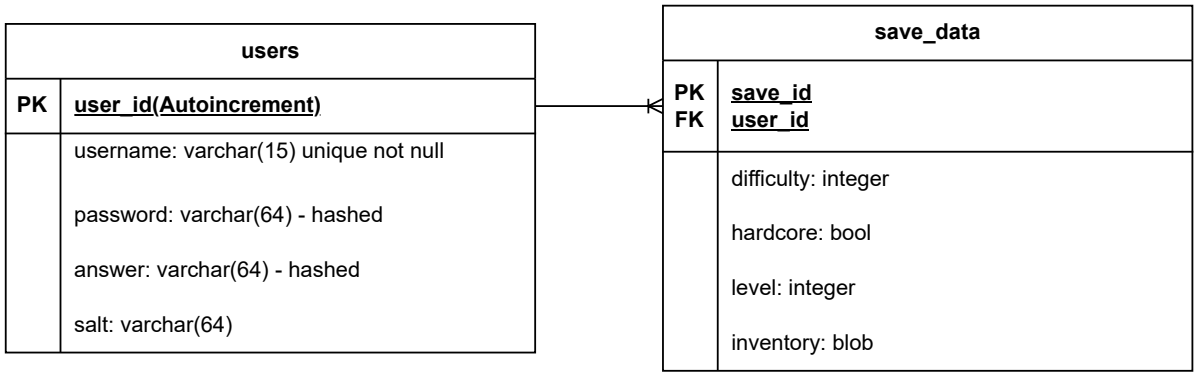

Figure 3: Database Design

Figure 2 shows the Database Design:
The users table will be the main table containing all the login details.
Each user will be able to have multiple save instances which will be stored in save data.
Upon designing the inventory I have decided that I will split the inventory into the stored items which I will use a seperate table to store with the item_path as a primary composite key with the save _id aswell as storing the equipped items in the save_data table. I have also decided to split the composite key in the save_data table and just have the save_id as the primary key autoincrementing.

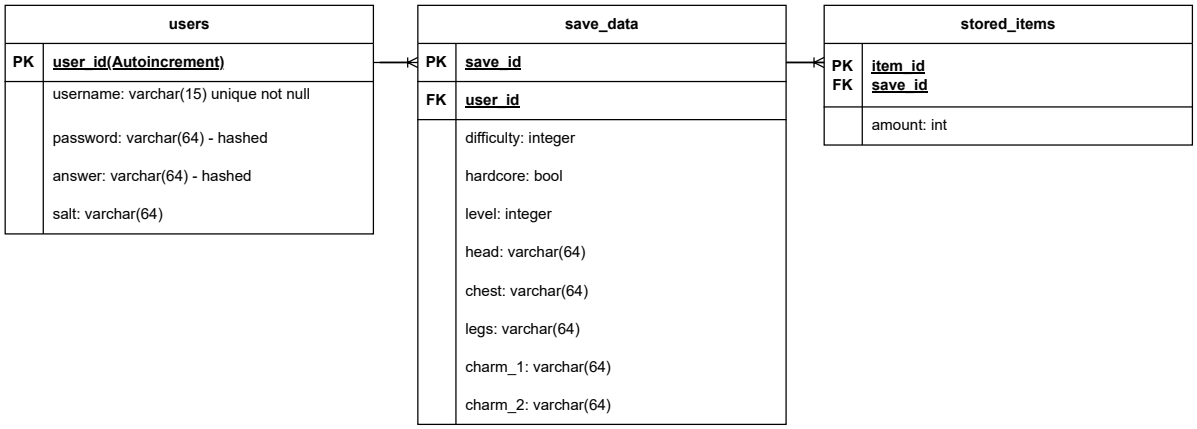

Figure 4: Database Design

2.2.2 Database Naming Conventions

The naming conventions I will adopt for the database is as follows.

Abstract	Convention	Examples	Justification
Tables	Plural snake_case	users,save_data	SQL is case insensitive so with CamelCase it cann't tell the difference between undervalue and underValue
Fields	Singular snake_case	inventory_content, username	
Keys	singular snake_case_table_id	user_id, save_data_id	

2.2.3 SQL Queries

I have to write Queries for each of the actions I want to do.

Name	Description/Justification	SQL

create_table_users	Create's a table for users if it does not exist.	`CREATE TABLE IF NOT EXISTS users (`
		`  user_id INTEGER PRIMARY KEY AUTOINCREMENT,`
		`  username VARCHAR(15) NOT NULL,`
		`  password VARCHAR(64) UNIQUE NOT NULL,`
		`  salt VARCHAR(64) NOT NULL,`
		`  answer VARCHAR(64) NOT NULL`
		`);`

get_user_data	Returns the user data assuming it exists. If it doesnt it will return null.	`SELECT * FROM users` `WHERE username = ?;`

add_new_user	Inserts a new user into users with username, password, challenge question answer and salt	`--Assume hashed password and answer`
		`INSERT INTO`
		`users(username,password,answer,salt)`
		`VALUES (?,?,?,?);`

reset_password	Changes a users password	`--Assume hashed password and answer`
		`UPDATE TABLE users`
		`SET password = ?`
		`WHERE username = ?`

create_table_save_data	Create's a table for save_data if it does not exist.	`CREATE TABLE IF NOT EXISTS save_data (`
		`  save_id INTEGER AUTOINCREMENT,`
		`  FOREIGN KEY (user_id) REFERENCES users(user_id),`
		`  difficulty INTEGER,`
		`  hardcore INTEGER,`
		`  level INTEGER,`
		`  head VARCHAR(32),`
		`  chest VARCHAR(32),`
		`  legs VARCHAR(32),`
		`  charm_1 VARCHAR(32),`
		`  charm_2 VARCHAR(32)`
		`);`

add_new_save_data	Adds new save data for a user.	`INSERT INTO`
		`save_data(user_id,difficulty,hardcore,level)`
		`VALUES (?,?,?,?);`

get_save_data	Get's the save data with a specific user_id and save_id	`SELECT * FROM save_data`
		`WHERE user_id = ?`
		`AND save_id = ?;`

get_user_save_data	Get's the save data for all entries with a specific user_id	`SELECT level, hardcore FROM save_data` `WHERE user_id = ?;`

update_save_data	updateSave	`UPDATE save_data`
		`SET`
		`  head = ?,`
		`  chest = ?,`
		`  legs = ?,`
		`  charm_1 = ?,`
		`  charm_2 = ?,`
		`  level = ?`
		`WHERE`
		`  user_id = ?`
		`AND save_id = ?;`

Name	Description/Justification	SQL
create_table_stored_items	Create's a table for stored_items if it does not exist.	`CREATE TABLE IF NOT EXISTS stored_items ( item_id INTEGER NOT NULL, save_id INTEGER NOT NULL, PRIMARY KEY(item_id,save_id), FOREIGN KEY(save_id) REFERENCES save_data(save_id) );`
update_stored_item_amount	Update's a specific person's stored items to increase the amount of something stored (assumes it is stored)	`UPDATE stored_items SET amount = amount + ? WHERE item_id = ? AND save_id = ?;`
get_stored_item_amount	Get's the amount of an item being stored	`SELECT amount FROM stored_items WHERE save_id = ? AND item_id = ?;`
add_stored_item	Adds an item to the stored_items	`INSERT INTO stored_items(save_id,item_id,amount) VALUES (?,?,?);`
count_stored_items	Count's the number of items stored for a save_id	`SELECT COUNT(*) FROM stored_items WHERE save_id = ?;`
remove_stored_item	Removes an item from stored_items	`DELETE * FROM stored_items WHERE save_id = ? AND item_id = ?;`
get_slot_value	Gets the file path of the item equipped in the slot	`SELECT ? FROM save_data WHERE save_id = ?;`
set_slot_value	Sets the value of the slot to the file path	`UPDATE save_data SET ? = ? WHERE save_id = ?;`

I will use godot's *query_with_bindings*() function in order to substitute in the bindings for the ?s in the queries. This is useful as it automatically performs input sanitisation so that the system isn't vulnerable to SQL injection.

2.2.4 Algorithms

login():
The login function will be used to find if the user exists and then check the hashed password if it does.

```
def login(username,password):
    query_result = get_user_data(username) #Getting user data
    if len(query_result) == 0: #If user doesnt exist
        return "InvalidUsernameError"
    if hash(password) == query_result["password"]: #Checking password hash against stored hash
        return True
    return "IncorrectPasswordError" #If password doesnt match
```

add_user():
The add_user function will be used to generate salt for the user check if the username is unique and add the user.

```
def add_user(username, password, answer):
    salt = gen_salt() #Generating new salt
    hashed_password = hash(password,salt)
    hashed_answer = hash(answer,salt)
    if not add_new_user(username, hashed_password, hashed_answer, salt): #Tries to add user
                                                                          with hashed password and answer
        return "InvalidUsernameError" #If user cannot be added then the username must be
                                                                            invalid
    return True
```

reset_password():
The reset_password function will be used to check if the username is valid, fetch the user data and then check if the hashed answer is the same as the stored answer before updating the stored password.

```
def reset_password(username, answer, password):
    query_result = get_user_data(username) #Getting user data
    if len(query_result) == 0: #If user doesnt exist
        return "InvalidUsernameError"
    if hash(answer) == query_result["answer"]: #Checking the answer hash against the stored
                                                hash
        reset_password(password,username)
        return True
    return "IncorrectAnswerError" #If answer doesnt match
```

2.3 Login System

2.3.1 Activity Diagram

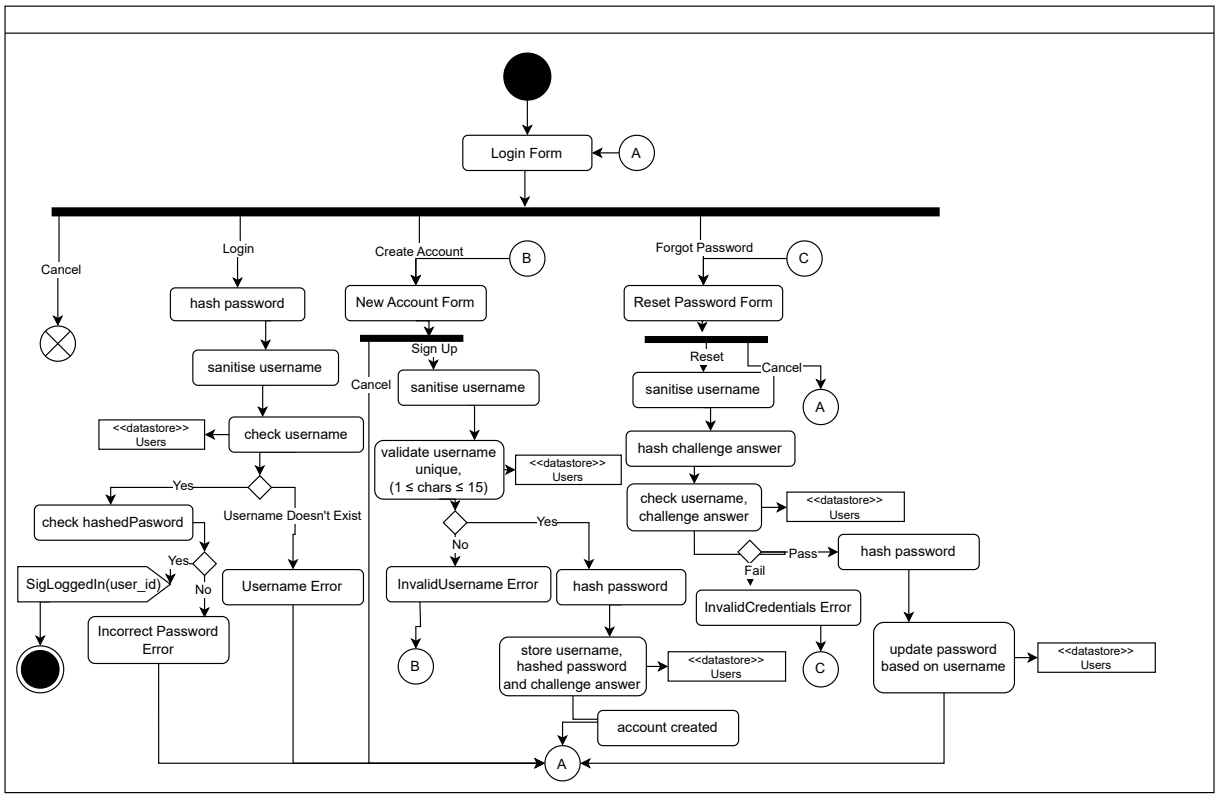

Figure 5: Activity Diagram for login forms

The login form will allow users to create accounts aswell as login with an existing account and reset a password. Upon successful login the user will be redirected to the GAME system.

2.3.2 Algorithms

hash():

```
def hash(password: str, salt: str):
    hashedPassword = password
    #Repeating a consistent but unpredicatble amount of times
    #On even rounds the password is sandwidged on odd rounds the salt is sandwidged
    #Alternating the use of sha256 and md5 but making sure to end on sha256 so the hash is a
                                                        predictable length.
    for x in range(1,6*len(password)+1):
        if x\%2 == 0:
            hashedPassword = sha256(md5(salt[x:]+hashedPassword+salt[:x]))
        else:
            hashedPassword = md5(sha256(hashedPassword[:x]+salt+hashedPassword[x:]))
          return hashedPassword
```

genSalt():
The genSalt function uses random processes to generate a salt string to be used in the hashing of the password and challenge question.

```
def gen_salt():
    salt = "string"
    x = randint(5,10)
    for i in range(2**x):
        salt = hash(salt,sha256(str(i)))
    return salt
```

removeForm():

```
def changeForm(form1: scene,form2: scene):
    self.remove_child(form1)
    form  = form1.instantiate()
    self.add_child(form)
```

2.3.3 Mockup Forms

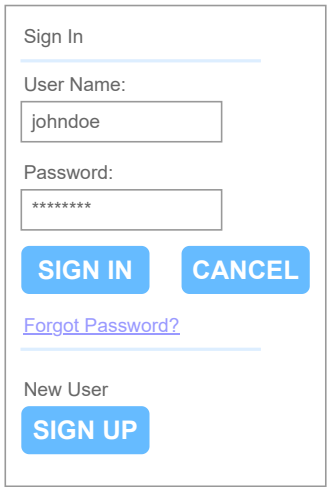

Figure 6: Login Form

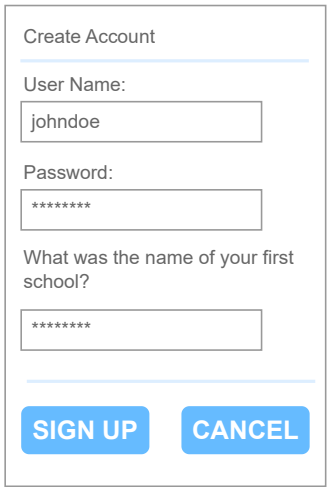

Figure 7: New Account Form

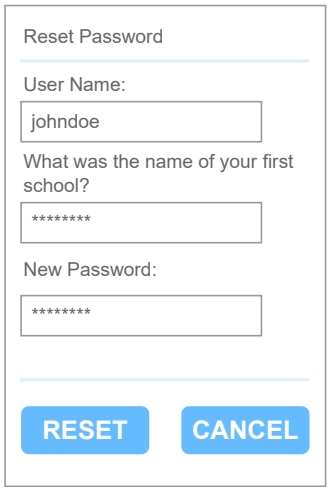

Figure 8: Reset Password Form

These forms would be used in order to create an account, reset your password and login.
The Password and Challenge question entries would be starred for privacy.

2.4 Item Design

I will implement the different item types using Godot's resource system. This will allow me to define properties that all items of the same type will share and I can use inheritance to allow classes to derive from a parent class.
The resource system is useful as it is reusable throughout scenes and scripts and it can easily be saved and loaded from disk.
The types of items I will aim to implement will be different weapon types, charms/trinkets/amulets, armour and keys.

2.4.1 Class Diagram

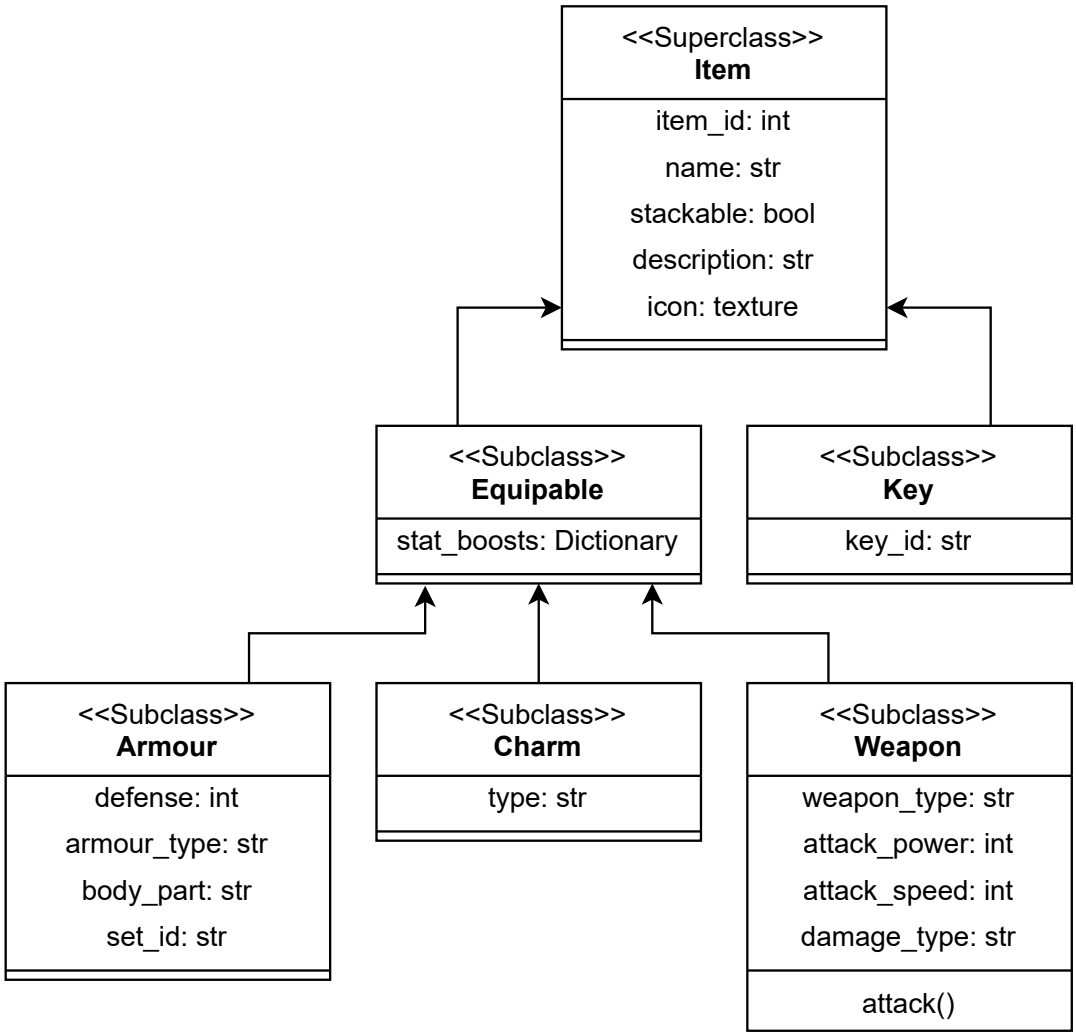

Figure 9: Player Class Diagram

2.4.2 Algorithms

attack():
I will have a number of different weapon types that will have different attacks.
I will implement the attack cooldown through the use of a CooldownTimer(timer node) attached to the player.
Melee:
I will implement the melee attack by creating a variable size hitbox(area2D) scene that can be instantiated as a child of the player in order to detect enemies that would be attacked in the range specified in the resource.

```
def attack(owner: node):
    #Load the hitbox scene
    hitbox_scene = load(hitbox_scene_path)
```

```
    hitbox_instance = hitbox_scene.instantiate()
    hitbox_instance.range = range
    hitbox_instance.damage = damage

    #Add child
    owner.add_child(hitbox_instance)

    #Hitbox lasts for half a second
    sleep(0.5)

    hitbox_instance.queue_free()
```

Ranged:
My Bows and ranged magic weapons will shoot out projectiles.

```
def attack():
    #Load the projectile scene
    projectile_scene = load(projectile_scene_path)
    projectile_instance = projectile_scene.instantiate()
    projectile_instance.damage = damage

    #Add Child
    owner.add_child(projectile_instance)
```

2.5 Inventory Design

My inventory design will cover two main parts the equipped items and the item storage. I will have a maximum inventory size script variable so that we can still display all the items in the inventory and a max stack size for stackable items.

2.5.1 Stored Items

I will implement the stored items through a dictionary that stores the item resource and the quantity of it. I will implement add and remove item functions. I will also add a max inventory size.

2.5.2 Equipped Items

I will implement the equipped items through a dictionary where the keys are the slots and the values are what is equipped in that slot. I will also add a equip function to equip an item and an unequip function to unequip the item in a slot.

2.5.3 Clarification

Upon further thought I have decided it is best to use SQL tables instead of dictionaries and use SQL queries to manage the inventory.

2.5.4 Algorithms

add_item():

```
def add_item(item_id: file_path, amount):
    if count_stored_items(save_id) > max_inventory_size: #Full Inventory
        return "FullInventoryError"
    else:
        #Checks if you can stack the item and either adds a new entry or stacks it
        if get_stored_item_amount(save_id,item_id) and item.stackable: #If the item is in
                                                                       the database and stackable
            update_stored_item_amount(amount, item_id, save_id)
        else:
            add_stored_item(save_id, item_id, amount)
    return True
```

remove_item():

```
def remove_item(item: Resource, amount: int = 1):
    if get_stored_item_amount(save_id, item_id): #If the item is in the database
        if get_stored_item_amount(save_id, item_id) < quantity: #Not enough items
            return "ItemQuantityError"
        if get_stored_item_amount(save_id, item_id) == quantity: #Exactly enough items
            remove_stored_item(save_id, item_id)
        else:
            update_stored_item_amount(-amount, item_id, save_id) #Removes the ammount of that
                                                                  item
        return True #Indicates that it was successful
    return "ItemQuantityError" #Indicates that there is an item quantity error
```

unequip_item():

```
def unequip_item(slot: str):
    #Checks if there is an item to unequip
    if get_slot_value(slot, save_id) != null:
        item = get_slot_value(slot, save_id)
        add_item(item, 1) #Adds the item back to the stored_items
        set_slot_value(slot, null, save_id) #Sets the slot back to null
        return True
    return True #If not item in slot it is unequipped
```

equip_item():

```
def equip_item(item: Resource):
        #Checks if the item is Equipable
        if not(item.is_class(Equipable)):
                return False
        #Gets the slot to equip it into
        if item.is_class(Armour):
                slot = item.body_part
        elif item.is_class(Weapon):
                slot = "weapon"
        else:
                equipped = false
                #Tries both charm slots
                for slot in ["charm1","charm2"]:
                        if get_slot_value(slot, save_id) == null and not(equipped):
                                set_slot_value(slot, item, save_id) #Equips item
                                remove_item(item) #Removes from stored_items
                                equipped = True
                if not(equipped):
                        unequip_item(slot)
                        set_slot_value(slot, item, save_id) #Equips item
                        remove_item(item) #Removes from stored_items
                return True
        if slot in equipped_items:
                unequip_item(slot)
                set_slot_value(slot, item, save_id) #Equips item
                remove_item(item) #Removes from stored_items
                return True
        return False
```

2.6 Player Character

This is my design for the physical player character and sprite.

2.6.1 Composition

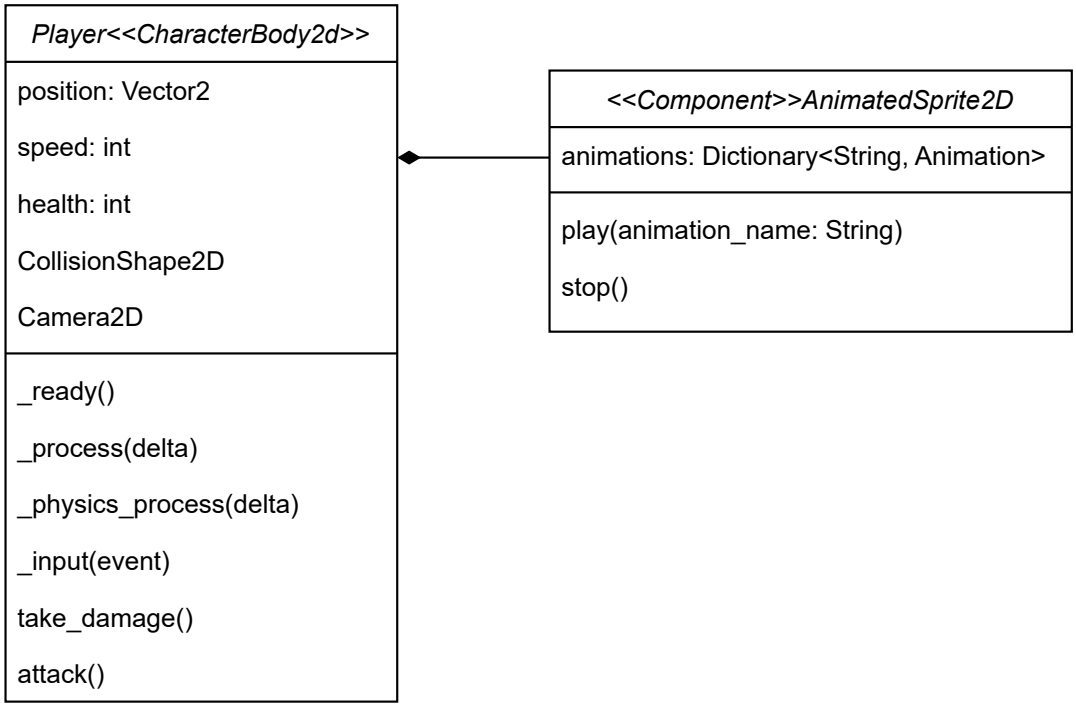

Figure 10: Player Class Diagram

The root node of the player which will contain all the child nodes will be Godot's CharacterBody2D as this will allow for a user controlled physics body. It will then have child nodes of CollisionShape2D(for collision detection), AnimatedSprite2D(for an animated character sprite) and Camera2D(for the player's view window to be centered on).
I have chosen to store the speed and health variables within the player class as they will reset/ be recalculated based of the equipment equipped.

2.6.2 Algorithms

_ready():
The _ready() function gets called whenever the player is instantiated in a scene and so it will be used to setup variables and the environment based on existing stuff.

```
def _ready():
        #Inventory calculates the speed based on any modifiers equipped.
        speed = Inventory.calc_speed()
        #Global Script calculates the health based on the player level and any modifiers
                                                    equipped.
        health = Global.calc_health()
```

_physics_process(delta):
The _physics_process(delta) function gets called every frame where delta is the time since the last fram and is usually used to deal with movement and physics processes.

```
def _physics_process(delta):
   direction = Input.get_vector("left", "right", "up", "down")
   velocity = direction * speed
```

```
    move_and_slide()
```

_input(event):

```
def _input(event):
    if event.is_action_pressed('attack'):
        attack()
```

3 Development

3.1 Database Development

I used a global autoloaded script database.gd in order to implement all of my functions for handling the database. Upon testing the functions I realised that the reset_password query was incorrect as it says UPDATE TABLE instead of just update.
I added all the prepared queries as private variables with strings in order to use db.query_with_bindings to sanitise and substitute inputs aswell as run the queries. This function would output wether the query succeeded or failed.
I then could use db.query_result in order to get the results of the query.

3.1.1 _ready()

```
func _ready() -> void:
    db.path = "res://game_data.db"
    db.open_db()
    if not db.query(_create_table_users):
        print("Error: users table unable to be created")
        return

    if not db.query(_create_table_save_data):
        print("Error: save_data table unable to be created")
        return

    if not db.query(_create_table_stored_items):
        print("Error: stored_items table unable to be created")
        return
    print("DONE")
```

Figure 11: _ready

In the database script db is declared using $SQLite.new()$ which is a wrapper class. I use the script to load the database and make sure all the necessary tables are present.
I also made it so that the database is closed when the script exits the tree so as to make sure all the data is saved properly.

3.1.2 Hashing

```
#Function for generating salt
func gen_salt() -> String:
    var salt = "string"
    var x = randi_range(5,10)
    for i in range(2**x):
        salt = j_hash(salt,str(i*randi_range(1,10)))
    return salt
```

Figure 12: gen_salt

```
#Function for hashing a password or challenge answer
func j_hash(string, salt):
    var hashedString = string
    #Repeating a consistent but unpredictable amount of times
    #On even rounds the password is sandwidged on odd rounds the salt is sandwidged
    #Alternating the use of sha256 and md5 but making sure to end on sha256 so the hash is a predictable length.
    for x in range(1,6*len(string)+1):
        if x % 2 == 0:
            hashedString = (salt.substr(x,hashedString.length()-x)+hashedString+salt.substr(0,x)).md5_text().sha256_text()
        else:
            hashedString = (hashedString.substr(0,x)+salt+hashedString.substr(x,hashedString.length()-x)).sha256_text().md5_text()
    return hashedString
```

Figure 13: hash

The hash and gen_salt functions implementation followed the pseudocode pretty faithfully apart from the fact I decided to not hash the number turned into a string as the salt doesnt have to be a certain length for the code to work. I also decided to times the number by a random integer to increase randomness and the number of possible salts.

3.1.3 Login Functions

```
func login(username,password):
    db.query_with_bindings(_get_user_data,[username]) # Getting user data
    if len(db.query_result) == 0: # If user doesnt exist
        return "InvalidUsernameError"
    var user_data = db.query_result[0]
    var hashed_password = j_hash(password,user_data["salt"])
    if hashed_password == user_data["password"]: # Checking password hash against stored hash
        current_user_id = user_data["user_id"]
        return true
    return "IncorrectPasswordError" # If password doesnt match
```

Figure 14: login

This algorithm is a copy of the design algorithm just using godot's relevant functions instead. I further saved the current_user_id for ease of future queries.

```
#Function for creating a user
func add_user(username, password, answer):
    var salt = gen_salt() #Generating new salt
    var hashedPassword = j_hash(password, salt)
    var hashedAnswer = j_hash(answer, salt)
    if not db.query_with_bindings(_add_new_user,[username,hashedPassword,hashedAnswer,salt]): #Tries to add user
        return "InvalidUsernameError" #If user cannot be added then the username must be invalid
    return true
```

Figure 15: add_user

This algorithm is a copy of the design algorithm just using godot's relevant functions instead.

```
func reset_password(username, answer, password):
    db.query_with_bindings(_get_user_data,[username]) # Getting user data
    if len(db.query_result) == 0: # If user doesnt exist
        return "InvalidUsernameError"
    var user_data = db.query_result[0]
    var hashed_answer = j_hash(password,user_data["salt"])
    if hashed_answer == user_data["answer"]: # Checking the answer hash against the stored hash
        db.query_with_bindings(_reset_password,[password,username])
        return true
    return "IncorrectAnswerError" # If answer doesnt match
```

Figure 16: reset_password

This algorithm is a copy of the design algorithm just using godot's relevant functions instead.

3.2 Login System Development

I used godot's inbuilt label, button and line edit node's in order to construct my forms. To each form I added an extra label in order to display Errors to the user.
I linked the buttons pressed signals to a script in order to determine what happens when the button is pressed and used variables to fetch and store the data from the line edit nodes.
I used node2ds in order to create groups of the nodes for more organisation and I kept the form layout mostly the same without some of the fancier unnecesary design elements from the mockup forms.

3.2.1 Login Form

Figure 17: Layout

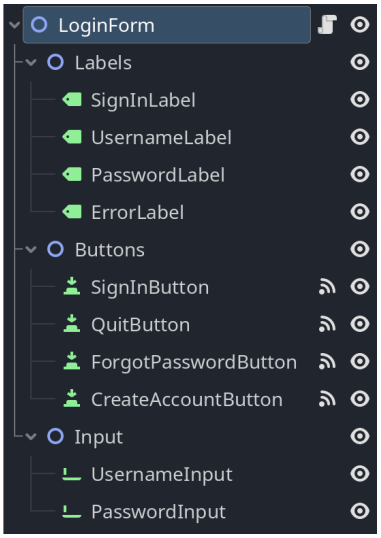

Figure 18: Structure

```gdscript
func _on_forgot_password_button_pressed() -> void:
    get_tree().change_scene_to_file("res://scenes/reset_password_form.tscn")


func _on_create_account_button_pressed() -> void:
    get_tree().change_scene_to_file("res://scenes/menu/create_account_form.tscn")


func _on_quit_button_pressed() -> void:
    global.quit()
```

Figure 19: button_pressed functions

These button functions are pretty simple as I only need to change scene or quit the game.

```gdscript
func _on_sign_in_button_pressed() -> void:
    var username = $Input/UsernameInput.text
    var password = $Input/PasswordInput.text
    var success = database.login(username, password)
    if not (typeof(success) == TYPE_BOOL and success == true):
        if success == "InvalidUsernameError":
            $Labels/ErrorLabel.text = "Invalid Username"
        elif success == "IncorrectPasswordError":
            $Labels/ErrorLabel.text = "Incorrect Password"
    else:
        get_tree().change_scene_to_file("res://scenes/menu/save_menu.tscn")
```

Figure 20: _on_sign_in_button_pressed

This is the function for when the sign in button is pressed it fetches the data and tries to login, displaying any errors it gets. If the login is successful theen it switches the scene to the save_menu scene.

3.2.2 Reset Password Form

Figure 21: Layout

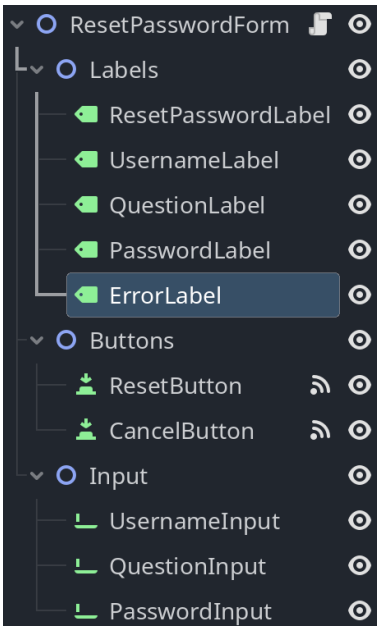

Figure 22: Structure

```gdscript
func _on_cancel_button_pressed() -> void:
    get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
```

Figure 23: _on_cancel_button_pressed

This button function is pretty simple as I only need to change scene back to the login form.

```gdscript
func _on_reset_button_pressed() -> void:
    var username = $Input/UsernameInput.text
    var answer = $Input/QuestionInput.text
    var password = $Input/PasswordInput.text
    var success = database.reset_password(username, answer, password)
    if not (typeof(success) == TYPE_BOOL and success == true):
        if success == "InvalidUsernameError":
            $Labels/ErrorLabel.text = "Invalid Username"
        elif success == "IncorrectAnswerError":
            $Labels/ErrorLabel.text = "Incorrect Answer"
    else:
        get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
```

Figure 24: _on_reset_password_button_pressed

This is the function for when the reset password button is pressed it fetches the data and tries to reset the password, displaying any errors it gets. If the reset is successful then it switches the scene to the login_form scene.

3.2.3 Create Account Form

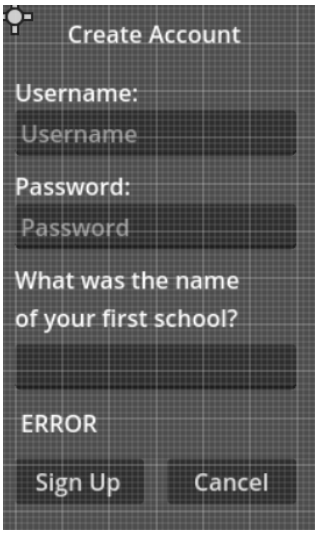

Figure 25: Layout

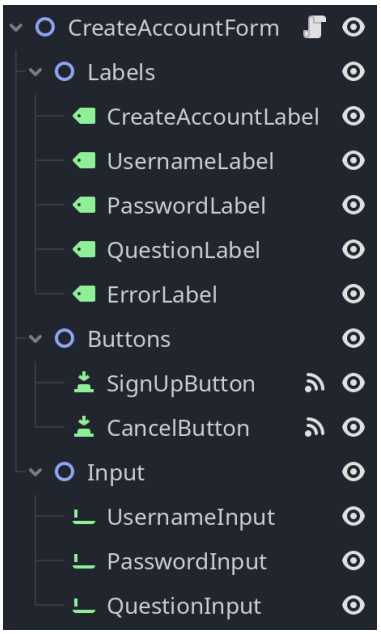

Figure 26: Structure

```gdscript
func _on_cancel_button_pressed() -> void:
    get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
```

Figure 27: _on_cancel_button_pressed

This button function is pretty simple as I only need to change scene back to the login form.

```gdscript
func _on_sign_up_button_pressed() -> void:
    var username = $Input/UsernameInput.text
    var password =  $Input/PasswordInput.text
    var answer = $Input/QuestionInput.text
    var success = database.add_user(username, password, answer)
    if not (typeof(success) == TYPE_BOOL and success == true):
        if success == "InvalidUsernameError":
            $Labels/ErrorLabel.text = "Invalid Username"
    else:
        get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
```

Figure 28: _on_reset_password_button_pressed

This is the function for when the create account button is pressed it fetches the data and tries to create the account, displaying any errors it gets. If the reset is successful then it switches the scene to the login_form scene.

3.3 Item Development

I will use resource scripts in order to implement the item classes and I will export the variables so that when I create new resources I can set the values.
In order to export the armour_type, body_part, charm_type, weapon_type and damage_type I will use an enum as it can only take one of the values in the list. This means the variables will take the form of an integer instead of a string.

3.3.1 Folder Structure

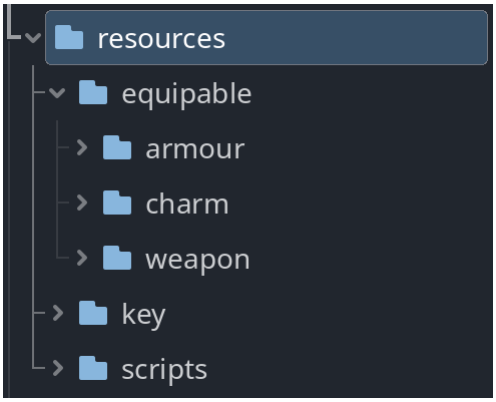

Figure 29: Folder Structure

I added more folders in order to organise the items into their groups aswell as keeping the resource scripts in a scripts folder.

3.3.2 Item

```
1  extends Resource
2
3  class_name Item
4
5  @export var stackable: bool
6  @export var description: String
7  @export var icon: Texture
```

Figure 30: Item Script

Figure 31: Item Exports

This shows the Item script and exported variables which I can set for each instance of that class including instances of classes that inherit from item.
I chose to remove the item_id as it seemed complicated to autoincrement it and enforce uniqueness and so I will store the file path in the item_id column in the database instead of an integer and so I updated the create_table_stored_items query in order to allow that.

3.3.3 Equipable

```gdscript
extends Item

class_name Equipable

@export var stat_boosts: Dictionary = {}

func _init():
    stackable = false
```

Figure 32: Equipable Script

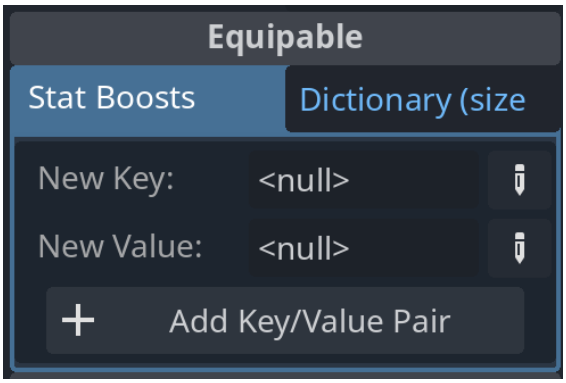

Figure 33: Equipable Exports

This shows the Equipable script and exported variables which I can set in any instance of this class or classes that inherit from it. I am using a dictionary to store stat boosts where the key is the stat and the value is the boost and these pairs can be added through the inspector. I set stackable to false by default as Equipable items will not be stackable.

3.3.4 Armour

```gdscript
extends Equipable

class_name Armour

@export var defense: int
@export_enum("Light", "Heavy") var armour_type: int
@export_enum("Head", "Chest", "Legs") var body_part: int
@export var set_id: int
```

Figure 34: Armour Script

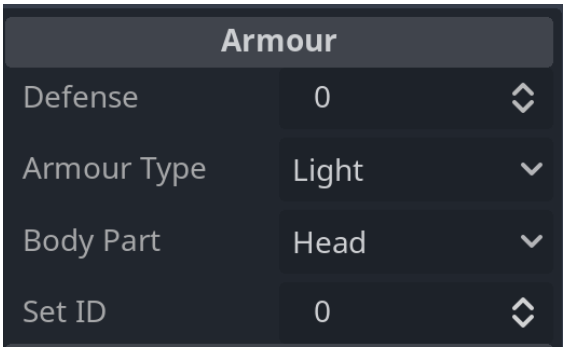

Figure 35: Armour Exports

This shows the Armour script and exported variables which I can set in any instance. I used an enum to represent the types of armour and body parts which it can be equipped on.

3.3.5 Charm

```gdscript
extends Equipable

class_name Charm

@export_enum("Ice", "Fire", "Cursed", "Divine","Poison") var charm_type: int
```

Figure 36: Charm Script

Figure 37: Charm Exports

This shows the Charm script and exported variables which I can set in any instance. I used an enum to represent the different charm types as you can only have one of them.

3.3.6 Weapon

```gdscript
extends Equipable

class_name Weapon

@export_enum("Ranged", "Magic", "Melee") var weapon_type: int
@export var attack_power: int
@export var attack_range: int
@export_enum("Ice", "Fire", "Cursed", "Divine","Poison") var damage_type: int
```

Figure 38: Weapon Script

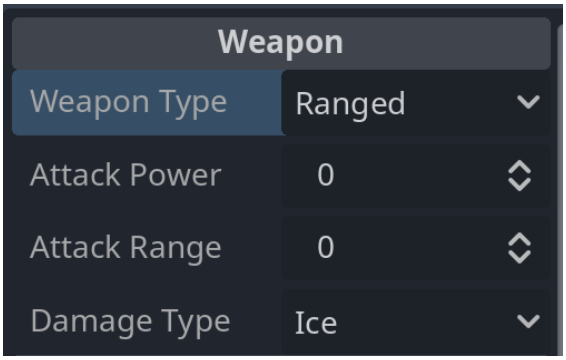

Figure 39: Weapon Exports

This shows the Weapon script and exported variables which I can set in any instance. I used an enum to represent the different weapon types and damage types so you can only select one

3.3.7 Key

```gdscript
extends "res://resources/scripts/item.gd"

class_name Key

@export var key_id: String
```

Figure 40: Key Script

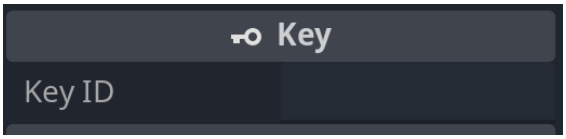

Figure 41: Key Exports

This shows the Key script and exported variables which I can set in any instance. The key ID will correspond to a door id and unlock that door.

3.4 Inventory Development

I used a seperate autoloaded script inventory.gd in order to implement the inventory functions.
I added functions to the database script in order to utilise the current_save_id script variable so that I dont have to input the variable every time I want to run a save_data or stored_items query. The functions also return the query result.

3.4.1 Add Item

3.4.2 Remove Item

3.4.3 Unequip Item

3.4.4 Equip Item

4 References

REF#	Date	Topic/Abstract	Type	URL or BOOK reference	How I used this
1	1/6/24	Research/ Existing Solutions	video games store, online	Steam (The Binding of Isaac)	One of the exisiting solutions I researched.
2	15/6/24	Research/ Existing Solutions	video games store, online	Steam (Dead Cells)	One of the existing solutions I researched
3	15/6/24	Research/ Existing Solutions	youtube video, online	Youtube (Motion Twin)	A dev log for an existing solution.
4	15/6/24	Research/ Existing Solutions	blog, online	robertheaton.com	An existing algorithm I researched.
5	25/11/24	Research/ Exisiting Solutions	video games store, online	Nintendo Store (Breath of The Wild)	One of the existing solutions I researched