NEA

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

Contents

1 Analysis **1**
- 1.1 Dungeon Crawlers . . . 1
- 1.2 The Problem . . . 1
- 1.3 Stakeholders . . . 1
  - 1.3.1 Survey . . . 1
  - 1.3.2 Survey Results . . . 2
  - 1.3.3 About Stakeholders . . . 3
- 1.4 Research . . . 4
  - 1.4.1 Existing Solutions . . . 4
- 1.5 Limitations and Requirements . . . 7
- 1.6 Features . . . 8
  - 1.6.1 Essential Features . . . 8
  - 1.6.2 Desireable Features . . . 9
- 1.7 Success Criteria . . . 10
- 1.8 Computational Methods . . . 13

2 Design & Plan **14**
- 2.1 Overview . . . 14
  - 2.1.1 Global Variables . . . 14
  - 2.1.2 Folder Structure . . . 14
  - 2.1.3 Naming Convention . . . 14
- 2.2 Database Design . . . 15
  - 2.2.1 ERD . . . 15
  - 2.2.2 Database Naming Conventions . . . 16
  - 2.2.3 SQL Queries . . . 16
  - 2.2.4 Algorithms . . . 20
  - 2.2.5 Testing Plan . . . 20
- 2.3 Login System . . . 21
  - 2.3.1 Activity Diagram . . . 21
  - 2.3.2 Algorithms . . . 21
  - 2.3.3 Testing Plan . . . 22
  - 2.3.4 Mockup Forms . . . 22
- 2.4 Save Select System . . . 23
  - 2.4.1 Algorithms . . . 23
- 2.5 Item Design . . . 23
  - 2.5.1 Class Diagram . . . 24
  - 2.5.2 Algorithms . . . 24
- 2.6 Inventory Design . . . 25
  - 2.6.1 Stored Items . . . 25
  - 2.6.2 Equipped Items . . . 25
  - 2.6.3 Clarification . . . 25
  - 2.6.4 Algorithms . . . 25
  - 2.6.5 Testing Plan . . . 27
- 2.7 Player Character . . . 28
  - 2.7.1 Composition . . . 28
  - 2.7.2 Animations . . . 28
  - 2.7.3 Help Screen . . . 28
  - 2.7.4 Algorithms . . . 28
- 2.8 Weapon Hurtbox . . . 30
  - 2.8.1 Algorithms . . . 30
- 2.9 Dungeon Environment Design . . . 30
  - 2.9.1 Generic Tiles . . . 30
  - 2.9.2 Chests . . . 30
  - 2.9.3 Doors . . . 30
  - 2.9.4 Algorithms . . . 30
- 2.10 Projectile Design . . . 31
  - 2.10.1 Overview . . . 31
  - 2.10.2 Ranged . . . 31

2.10.3 Area Of Effect . . . 31
2.10.4 Algorithms . . . 31
2.11 Enemy Design . . . 32
2.11.1 Overview . . . 32
2.11.2 Composition . . . 32
2.11.3 Navigation . . . 32
2.11.4 Animation . . . 33
2.11.5 Projectile Enemies . . . 33
2.11.6 Algorithms . . . 33
2.12 UI . . . 35
2.12.1 Display Strings . . . 35
2.12.2 Inventory UI . . . 35
2.12.3 Game UI . . . 35
2.12.4 Algorithms . . . 36
2.13 Procedural Generation Design . . . 37
2.13.1 Overview . . . 37
2.13.2 Graph Design . . . 37
2.13.3 Room Design . . . 37
2.13.4 Algorithms . . . 37

3 Development & Testing **39**
3.1 Database Development . . . 39
3.1.1 _ready() . . . 40
3.1.2 Hashing . . . 40
3.1.3 Login Functions . . . 41
3.1.4 Testing . . . 42
3.2 Login System Development . . . 43
3.2.1 Login Form . . . 44
3.2.2 Reset Password Form . . . 45
3.2.3 Create Account Form . . . 45
3.3 Save Select System . . . 47
3.4 Video Testing (Login Forms) . . . 48
3.5 Item Development . . . 48
3.5.1 Folder Structure . . . 48
3.5.2 Item . . . 48
3.5.3 Equipable . . . 50
3.5.4 Armour . . . 50
3.5.5 Charm . . . 50
3.5.6 Weapon . . . 51
3.5.7 Key . . . 51
3.5.8 Revision . . . 52
3.6 Inventory Development . . . 52
3.6.1 Add Item . . . 52
3.6.2 Remove Item . . . 52
3.6.3 Unequip Item . . . 53
3.6.4 Equip Item . . . 53
3.6.5 Testing . . . 54
3.7 Hurtbox Development . . . 57
3.7.1 Layout . . . 57
3.7.2 Script . . . 58
3.8 Player Development . . . 58
3.8.1 Layout and Structure . . . 58
3.8.2 _physics_proccess(delta): . . . 59
3.8.3 Player Animation . . . 59
3.8.4 Player Attack . . . 60
3.9 TileMap Development . . . 60
3.10 Video Testing (Movement and TileMap) . . . 61
3.11 Dummy Development . . . 61
3.12 Stakeholder Feedback 1 . . . 61
3.12.1 Login System and Save Menu . . . 61

3.12.2 Test Scene . . . 62
3.13 Dash Development . . . 63
3.14 Dungeon Environment Development . . . 63
3.14.1 Door . . . 63
3.14.2 Chest . . . 64
3.15 Video Testing (Chests and Doors) . . . 65
3.16 Projectile Development . . . 65
3.16.1 Ranged . . . 65
3.16.2 Area . . . 67
3.17 Magic Weapon Development . . . 67
3.17.1 Ranged . . . 68
3.17.2 Area Of Effect . . . 68
3.18 Enemy Development . . . 68
3.18.1 Layout and Structure . . . 68
3.18.2 General Script . . . 69
3.18.3 _physics_process(): . . . 70
3.18.4 take_damage(damage, damage_type): . . . 71
3.18.5 Animation . . . 71
3.19 Video Testing (Player Attacks and Enemies) . . . 72
3.20 UI Development . . . 72
3.20.1 Inventory UI . . . 72
3.20.2 Game UI . . . 74
3.20.3 Enemy UI addition . . . 75
3.20.4 Help Menu . . . 76
3.21 Video Testing (UI) . . . 76
3.22 Tutorial Development . . . 76
3.23 Stakeholder Feedback 2 . . . 77
3.23.1 Dropped Features . . . 77
3.23.2 Tutorial . . . 78
3.23.3 Weapons . . . 78
3.23.4 Menus . . . 78
3.23.5 Inventory . . . 79
3.24 Procedural Generation Development . . . 80
3.24.1 Graph . . . 80
3.24.2 Room . . . 82
3.24.3 Testing . . . 84
3.24.4 Level 1 . . . 86

4 Evaluation **88**
4.1 Success Criteria . . . 88
4.2 Stakeholder Feedback 3 . . . 98
4.2.1 Save Menu . . . 98
4.2.2 Levels . . . 98
4.2.3 Inventory and Items . . . 98
4.2.4 Maintenance . . . 98
4.3 Final Evaluation . . . 99

5 References **100**

6 Code Listings **101**
6.1 resources/scripts . . . 101
6.2 scenes . . . 102
6.2.1 game/enemies . . . 102
6.2.2 game/player . . . 105
6.2.3 game/projectiles . . . 109
6.2.4 game/worlds . . . 111
6.2.5 menu . . . 114
6.3 src . . . 116
6.4 utils . . . 122

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. I want to take inspiration from the simplicity of ui in order to help my game be intuitive.

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. These are some of the features I will not include in my product opting to instead try for an easier approach by allowing scaleable difficulty and oct-directional movement and attacks.

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. Because of these advantages I will take heavy inspiration from this style of procedural generation for my level generation in order to make them more unique.

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 aswell as aiming to forgo some of the games more complex or challenging mechanics such as permadeath in order to create a more accessible game.

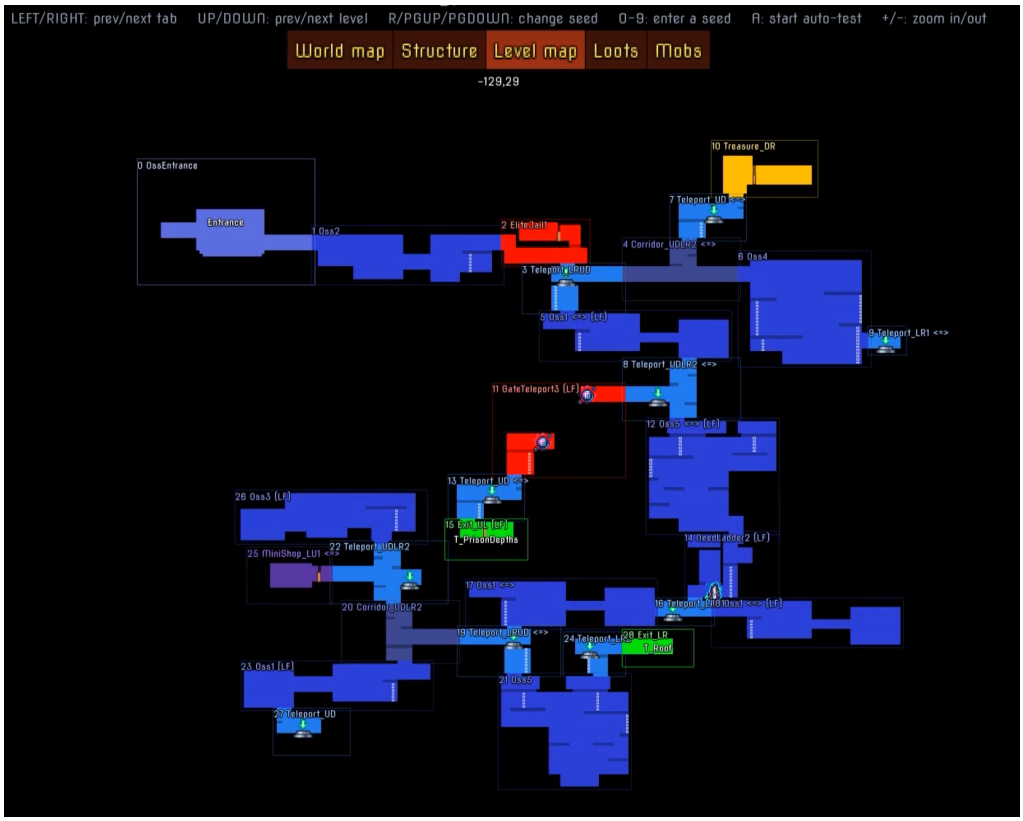


Figure 2: Dead Cells Level Generation Example

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 games 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 allow different routes through my 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.

Figure 3: BOTW tutorial map

This map shows how the BOTW tutorial has the different "shrines" placed to help the player learn the basic mechanics of the game.

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 Q will trigger a dash. Alternatively they will use the left control stick of a controller.	This will be used to navigate around the Dungeon environment and WASD was the most popular control mechanism for the stakeholders with controller close behind. The controls of my game aim to follow the stakeholder feedback aswell as the general controls that seemed to be favoured in my research
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. The existing solutions all have at least a basic combat system as one of the driving forces for progress through 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. All the existing solutions had dungeon environments as this is an essential part of a 'dungeon crawler'.
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 as I saw it used during my research.
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 tells you about where your character is aswell as what they are doing even if the animations are basic like in Binding of Isaac.
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. All the Exisiting solutions I looked at either had login systems or used an existing login system (e.g. steam) in order to manage seperate user saves.
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 communicate necessary information for playing the game as I found through the Binding of Isaac UI.

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 as I saw through my research into Dead Cells and BOTW.
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 and can be seen in Dead Cells.
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 and was used in Dead Cells to allow for greater replayability.
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 can be seen in alot of existing solutions and would allow for more time spent having fun in the game through finding these areas.
12	Inventory System	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. It can be seen in BOTW and less complex in Dead Cells.
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 slider.

1.7 Success Criteria

Criteria #	Abstraction	Success Criteria	Success Indicators
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.	WASD/Left Stick direction - Move in that direction Q - Faster movement in direction player is facing
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 Make sure all attacks go in the direction the player is facing	Attacks are triggered when their corresponding controls are pressed. Melee attacks affect all enemies within range in the direction the player is facing causing them to lose health. Projectiles launch on a ranged attack and travel in the direction the player is facing Area of effect attacks spawn an area around the player that slowly damages enemies that come into it. Enemy attacks cause player health to go down. Player health accurately displayed on a health bar in the UI
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 and Monster 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	Ability to walk around the dungeon environment and remain contained by it. Ability to open chests and recieve a specific quantity of random loot from a pool. A level built out of specific purpose built rooms and corridors.
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	Enemies have distinct and visually recognisable sprites with smooth animations. Enemies navigate around walls and obstacles and follow the player. Enemies detect the player within a certain range and react. Player attacks are registered and decrease enemy health. When an enemies health runs out it will die. Melee enemies attack the player within close range.

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 Dash Animation	The Player has a distinct and visually recognisable sprite with smooth animations for walking, melee attacks and others. The direction of the player changes based on last direction moved.
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 Input Sanitisation (Removing any escape chars for SQL before sending the command) 6.6 Ability to log in with an exisiting account and correct password 6.7 Ability to reset password (With challenge question) 6.8 A general login form which links the other forms. 6.9 Ability to delete an account.	Uses a strong hashing algorithm with salting. Username password pairs are stored in an SQL table. Users can only create an account if the username is unique and between 1 and 14 characters. Preventation of SQL injection attacks. User's can login with credentials. User's can reset their password. A general login form links to registration, password reset and logging in.
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	Displays Player health and MP accurately and updates dynamically. Clearly indicates which weapon is being used. Clearly displays enemies health when they get damaged. Allows switching between weapons.
8	Weapons And a More Advanced Combat System	8.1 Different Styles of melee, magic and ranged weapons 8.2 Boss Drops 8.3 Shop System that appears throughout levels 8.4 Charged Attacks (based on how long you hold down) 8.5 Special attacks	Distinct different weapon styles, levels and dynamics. Defeated Bosses drop unique or rare items. Shops can appear throughout levels. Holding down attack button increases power of attacks. More powerful secondary special attacks.
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	A user-friendy menu displaying skills and prerequesites. Seperate branches for Melee, Ranged, Magic and Defense skills. Players gain xp from defeating enemies and bosses. Different skills requiring different amounts of XP to unlock. Players can spend XP to unlock skills. Players can reset and redistribute points in the skill tree.

10	Procedurally Generated Dungeons	10.1 Creating requirements for each level to satisfy 10.2 Creating different 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.)	Each level generated meets specific preconditions. Different Room sections are designed to be peiced together dynamically. An algorithm places room sections together to form a level layout. Generated levels are fully playable and contain all rooms needed.
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.	Players can access secret rooms using specific methods. A hammer item allows players to break walls. Players need climbing gloves to scale vines. Secret areas contain unique bosses or loot.
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. 12.5 Ability to add or remove items from the inventory. 12.6 SQL table to store inventory contents	A clear and intuitive menu for managing items. Players can store extra weapons and items to be saved in their inventory. E Key - Open Inventory UI. Players can swap weapons/armour. Players can remove items from their inventory. Inventory state persists even when game closes.
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.	A clear and accessible menu with buttons for different settings. Players can adjust the volume of each source of noise in the game. Allow users to adjust colour intensity along with accessibility needs.
14	Difficulty Levels and Hardcore Mode	14.1 A slider for difficulty in create save 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) 14.4 SQL table to store different saves	A difficulty slider in the save menu. The game adjusts difficulty by increasing enemy health and damage and increasing the number of enemies. A hardcore mode with permadeath that can be toggled in the save creation menu. Saves persist even when game closes.

1.8 Computational Methods

Method	Description	Application to My Project
Thinking Abstractly	Removing unnecessary detail to focus on what's important	I will make general classes for items/enemies to reduce repeated code. I will use godot which automatically abstracts away the complexity behind all of the node types so that you get the functionality without dealing with the complexity. I will be using prexisting libraries for sha256 and md5 algorithms within my hashing for the passwords and challenge question answers.
Thinking Ahead	Planning for future stages or needs	I will design systems that can be expanded later, like adding new weapons. I will also preload scripts for quicker access when I need them.
Thinking Procedurally	Breaking tasks into clear steps	I will write algorithms to manage the combat system including taking damage and managing hitboxes and hurtboxes. I will write algorithms for the database to make sure all the data is handled and stored securely. I will bind specific inputs to specific functions such as attacking, moving the player, pulling up menus etc.
Thinking Logically	Using reasoning to control program flow	I will use event signals and that will be handled in order to get movement directions. I will be using loops throughout my hashing and salting in order to add complexity and reduce the chance of reverse engineering.
Thinking Concurrently	Running multiple processes at once	Player and enemy movement will happen at the same time using Godot nodes aswell as being able to pause certain threads and keep some active for a pause menu. The physics processes within godot will run on seperate threads to the rendering. The _ready() method is used as an initializer before the physics process loop which both run seperately from the rendering.
Problem Recognition	Identifying key criteria of my solution	I noticed replayability is poor in dungeon crawler games so I will use procedural generation to help fix this. I also recognise that some dungeon crawlers arent as accessible to new players and so I will aim to solve this. I also recognise that by representing rooms as nodes and using graph theory to for level structure we can use existing algorithms such as depth first search to help with procedural generation.
Backtracking	Returning to previous states/function calls to aid algorithms or find another solution	I will use backtracking in graph traversals for procedural generation in order to fully proccess all nodes in the graph even when reaching a dead end.
Heuristics	Using rules of thumb to guide decisions	In the pathfinding for enemies I will use godot's implementation of mesh A* which uses heuristics to speed up pathfinding by guiding it in the right direction.
Visualisation	Representing information graphically	I will use visualisation to visualise my classes and algorithms aswell as data flow within my design of the solution. Godot uses visualisation in order to help place all the different nodes throughout the scene and visualise the completed scene and how it will look.

2 Design & Plan

2.1 Overview

2.1.1 Global Variables

I have a couple of main global scripts global.gd, inventory.gd and databse.gd, this is because they will provide key functions that will be needed in throughout different of parts of the game. These scripts will be automatically loaded at the start of the game using godot's autoload feature. The global variables are the variables within the scripts that will be used throughout the game by external scripts.

Source	Identifier	Data Type	Justification
global.gd	difficulty	Integer	Needed for deciding enemy health/damage .
global.gd	current_level	Integer	Needed for deciding enemy health/damage.
database.gd	current_user_id	Integer	Used for SQL queries after having logged in.
database.gd	current_save_id	Integer	Used for SQL queries after having selected a save.

2.1.2 Folder Structure

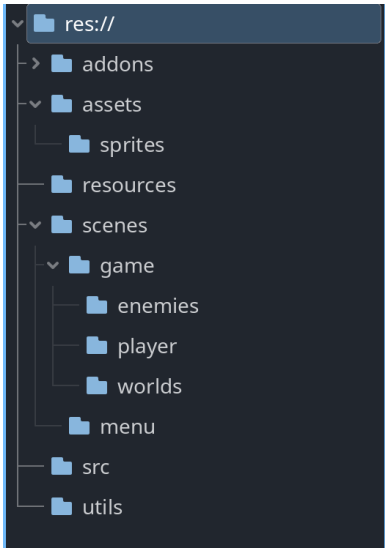


Figure 4: Folder Structure

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. This will aid in the development process as the different parts will be easily navigable and findable.
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. This will allow for a more seamless development integrating my features and godot's features without getting confused.
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 and their game saves.

2.2.1 ERD

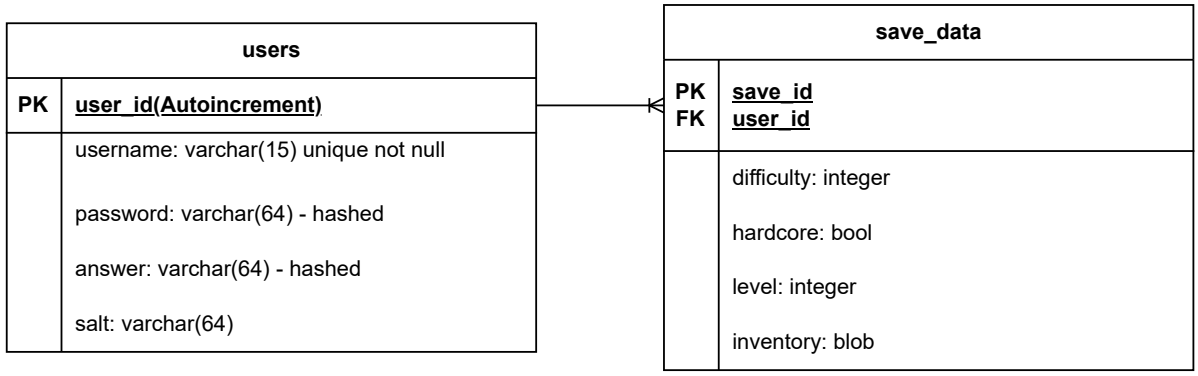


Figure 5: Database Design

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 system 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_id (the path to the item resources location in the game files) 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. This will help keep the inventory more accessible and prevent the need for a BLOB decoder to decode the inventory components allowing ease of access to all the stored and equipped items.

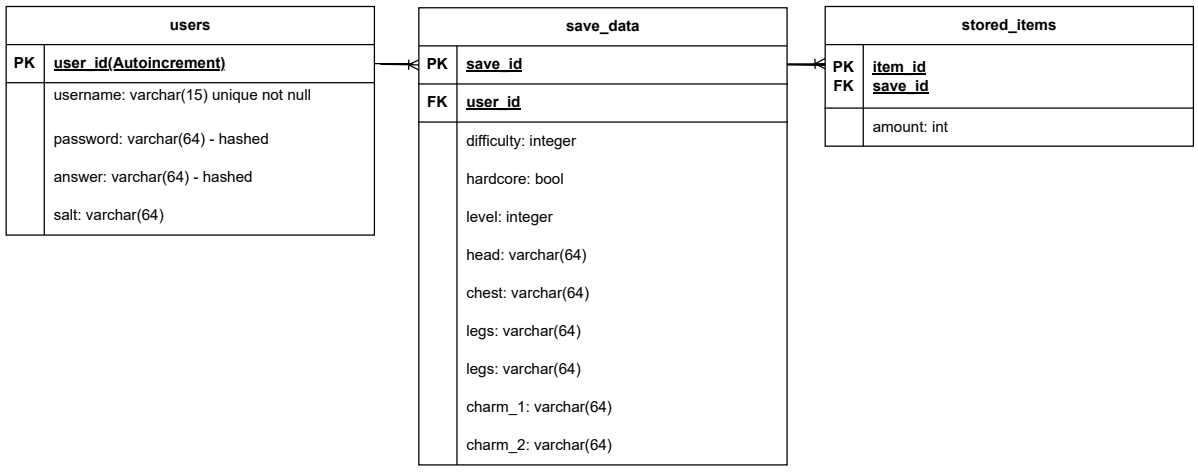


Figure 6: Database Design

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 can't tell the difference between undervalue and underValue thereofre I use the underscores to distinguish words.
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. These queries are necesary for the login system criteria aswell as the Inventory system as they will allow me to read, write and store user data.

Name	Description/Justification	SQL

create_table_users	Create's a table for users if it does not exist. Necessary for creating the table for criteria 6.2.	`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. Necessary for getting the hashed password to check login credentials for criteria 6.6 aswell as getting the hashed challenge answer for criteria 6.7.	`SELECT * FROM users` `WHERE username = ?;`

add_new_user	Inserts a new user into users with username, password, challenge question answer and salt. Necessary for creating a new user for criteria 6.3	`--Assume hashed password and answer`
		`INSERT INTO`
		`users(username,password,answer,salt)`
		`VALUES (?,?,?,?);`

reset_password	Changes a users password. Necessary for criteria 6.7.	`--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. Necessary for creating the table for criteria 14.4 and part one of the two tables for criteria 12.6 (storing equipped items).	`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),`
		`  weapon VARCHAR(32),`
		`  charm_1 VARCHAR(32),`
		`  charm_2 VARCHAR(32)`
		`);`

add_new_save_data	Adds new save data for a user. Necessary for criteria 14.4	`INSERT INTO`
		`save_data(user_id,difficulty,hardcore,level)`
		`VALUES (?,?,?,?);`

Name	Description/Justification	SQL

get_save_data	Get's the save data with a specific user_id and save_id. Necessary for fetching difficulty for criteria 14.2.	`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. Necessary for getting the data of all the saves to display in the save selection/creation menu.	`SELECT level, hardcore FROM save_data` `WHERE user_id = ?;`

update_save_data	Update's the save data with the new data. Necessary to change save difficulty if needed.	`UPDATE save_data`
		`SET`
		`head = ?,`
		`chest = ?,`
		`legs = ?,`
		`weapon = ?,`
		`charm_1 = ?,`
		`charm_2 = ?,`
		`level = ?`
		`WHERE`
		`user_id = ?`
		`AND save_id = ?;`

create_table_stored_items	Create's a table for stored_items if it does not exist. Necessary as the second table for criteria 12.6.	`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). Necessary for adding items if the player already has some of that item for criteria 12.5.	`UPDATE stored_items`
		`SET amount = amount + ?`
		`WHERE item_id = ?`
		`AND save_id = ?;`

Name	Description/Justification	SQL
get_stored_item_amount	Get's the amount of an item being stored. Necessary to know wether you can remove an item from the inventory for criteria 12.5.	`SELECT amount FROM stored_items WHERE save_id = ? AND item_id = ?;`
add_stored_item	Adds an item to the stored_items. Necessary for adding an item the player doesnt already have in their inventory for criteria 12.5.	`INSERT INTO stored_items(save_id,item_id,amount) VALUES (?,?,?);`
count_stored_items	Count's the number of items stored for a save_id. Necessary if I implement a maximum item limit for the inventory.	`SELECT COUNT(*) FROM stored_items WHERE save_id = ?;`
remove_stored_item	Removes an item from stored_items. Necessary for if you remove all of an item for criteria 12.5.	`DELETE * FROM stored_items WHERE save_id = ? AND item_id = ?;`
get_slot_value	Gets the file path of the item equipped in the slot. Necessary for 12.4 and for making sure the player knows what weapon to attack with.	`SELECT ? FROM save_data WHERE save_id = ?;`
set_slot_value	Sets the value of the slot to the file path given. Necessary for equipping stuff for criteria 12.4	`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 database isn't vulnerable to SQL injection that could lose user data preventing proper storage and criteria 6.2, 12.6 and 14.4.

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. This is necessary as it will help fulfill criteria 6.6 along with the login form.

```
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. This is necessary as it will help fulfill criteria 6.3 along with the create account form.

```
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. This is necessary as it will help fulfill criteria 6.7 along with the reset password form.

```
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
```

2.2.5 Testing Plan

Test #	Function	Parameters	Expected Outcome
6.3.1	add_user()	”Hyrule”, ”Password”, ”Answer”	True
6.3.2	add_user()	”Hyrule”, ”Password”, ”Answer”	”InvalidUsernameError” as a user already exists with that username
6.6.1	login()	”Hyru1e”, ”Password”	”InvalidUsernameError”
6.6.2	login()	”Hyrule”, ”Password”	True
6.7.1	reset_password()	”Hyru1e”, ”Answer”, ”password	”InvalidUsernameError”
6.7.2	reset_password()	”Hyrule”, ”answer”, ”password”	”IncorrectAnswerError”
6.7.3	reset_password()	”Hyrule”, ”Answer”, ”password”	True
6.6.3	login()	”Hyrule”, ”Password”	”IncorrectPasswordError”

These tests will be used to evaluate to what extent I have met the criteria to which they are indexed and allow me to specifically identify where any issues in my code are and fix them. The test data is used such that

users can already exist in the database with the username and such that we test logging in with the correct and incorrect password and ressetting the password with the correct and incorrect challenge question answer.

2.3 Login System

2.3.1 Activity Diagram

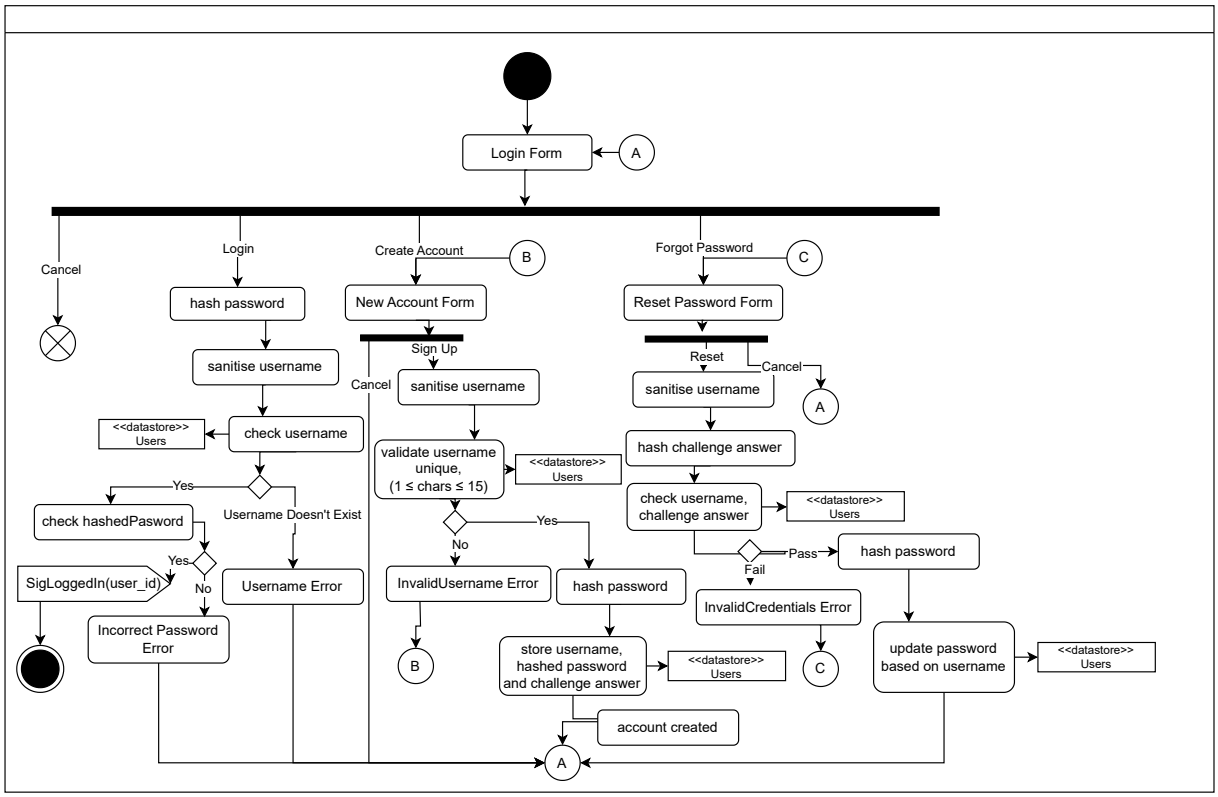


Figure 7: Activity Diagram for login forms

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. This login form along with the database functions provides all the necessary UI and algorithms to fulfill criteria 6.

2.3.2 Algorithms

hash():
This function alternates between two different hashing functions a consistent number of times while making sure the final hash is a consistent length. It also adds in a unique salt for every user which is necessary to prevent rainbow table lookups and keep the passwords secure even if it is generic. This and the salt function fulfills criteria 6.1.

```
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. This needs to be random for every user to prevent the potential to create a table of common password hashes to loop up user's passwords in.

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

2.3.3 Testing Plan

Test #	Function	Parameters	Expected Outcome
6.1.1	gen_salt()		random 256 bit hex string
6.1.2	hash()	"password", "salt"	random 256 bit hex string
6.1.3	hash()	"password", "salt"	the same random 256 bit hex string
6.1.2	hash()	"Password", "salt"	random 256 bit hex string different from before

2.3.4 Mockup Forms

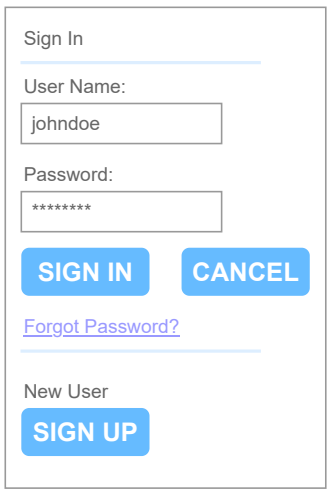


Figure 8: Login Form

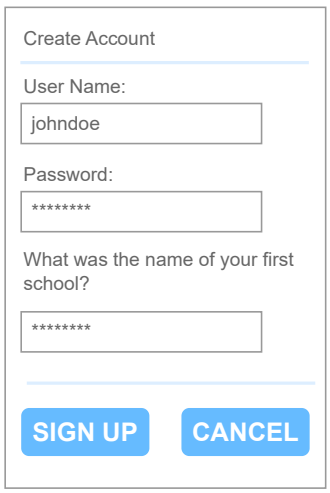


Figure 9: New Account Form

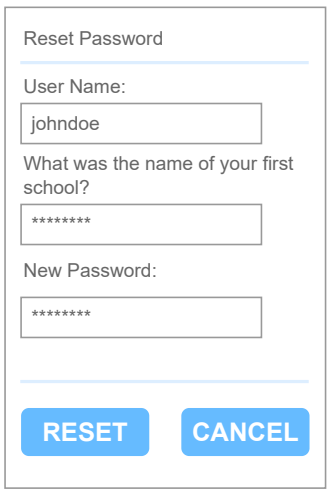


Figure 10: Reset Password Form

These forms would be used in order to create an account, reset your password and login, Helping fulfill criteria 6.3, 6.6, 6.7 and 6.8 alongside the algorithms.
The Password and Challenge question entries would be hidden/starred for privacy.

2.4 Save Select System

I will implement the save select system as an additional menu for the login system in order to improve replayability to help using a ScrollContainer with a VBoxContainer inside to create a list of scrollable items. I will get the list of saves for a given user_id and then display them and when the button for that save is pressed the current_save_id will be set and the scene will be switched to the current level. This will be used to fulfill criteria 14.1 aswell as creating and storing saves for criteria 14.4.
In order to add new save's I will add a button for hardcore and a slider for difficulty as well as an add save button which will add the save and update the list.

2.4.1 Algorithms

_ready(): This function is designed to add buttons for each save the user has to the scene in order to allow the player to select and play that save. This is necessary for the fulfillment of criteria 14.1.

```
def _ready():
    save_data_list = get_user_save_data(current_user_id)
    for save_data in save_data_list:
        button = Button.new
        button.text = f"Level: {save_data['level']} \t Difficulty: {save_data['difficulty']}\n
                                                    Hardcore: {save_data['hardcore']}"
        button.connect("pressed", self, "_on_save_selected", [save_data])
        add_child(button)
```

_on_save_selected():

```
def _on_save_selected(save_data):
    current_save_id = save_data["save_id"]
    get_tree().change_scene_to_file(world)
```

2.5 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. This is necessary to fulfill criteria 2.3, 2.4 and 2.5 for the weapons, aswell as providing the elements of a more complex combat system and the keys will help making doors and chests that only open when you have the keys helping with criteria 3.3 and 3.5.

2.5.1 Class Diagram

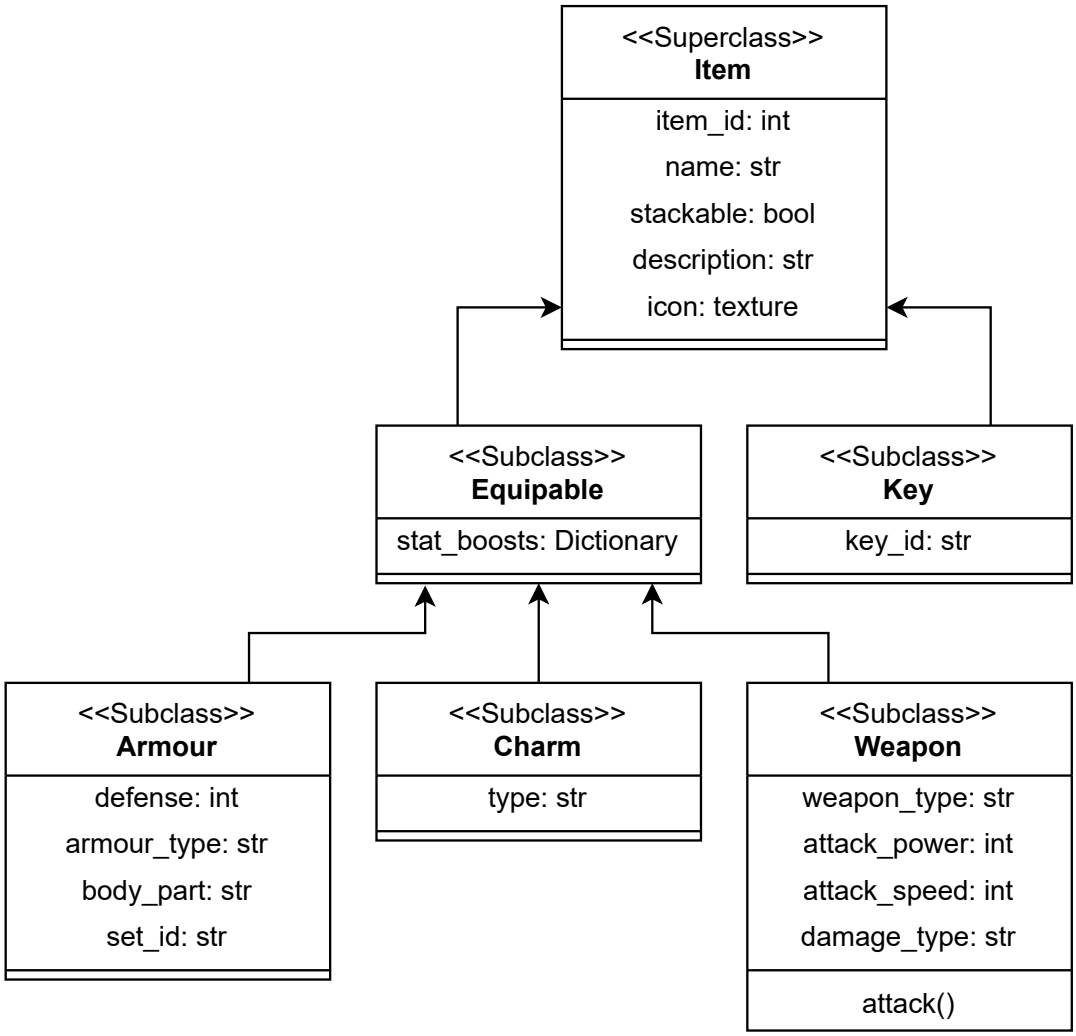


Figure 11: Player Class Diagram

2.5.2 Algorithms

attack():
I will have a number of different weapon types that will have different attacks. These attack functions are necesarry for fulfilling the weapon criteria (2.3,2.4 and 2.5) aswell as criteria 2.1.
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, direction):
    #Load the hitbox scene
    hitbox_scene = load(hitbox_scene_path)
    hitbox_instance = hitbox_scene.instantiate()
    hitbox_instance.range = range
    hitbox_instance.damage = damage
    hitbox_instance.rotation = direction.angle()

    #Add child
    owner.add_child(hitbox_instance)

    #Hitbox lasts for a tenth of a second
    sleep(0.1)

    hitbox_instance.queue_free()
```

Ranged Projectile:
My 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) #add as a child of the parent so that it doesnt move
                                                with the enemy/playe
```

Area Of Effect:
My AOE magic weapons will spawn in area's which affect enemies and players that walk in, damaging them.

```
def attack():
    #Load the area scene
    area_scene = load(area_scene_path)\
    area_instance = area_scene.instantiate()
    area_instance.damage = damage

    #Add Child
    owner.add_child(projectile_instance) #add as a child of the parent so that it doesnt move
                                                with the enemy/player
```

2.6 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.

2.6.1 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.

2.6.2 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.

2.6.3 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.

2.6.4 Algorithms

add_item():
This uses the SQL query to add items to the users stored items in the stored_items SQL table. This is necessary for criteria 12.5.

```
def add_item(item_id: file_path, amount):
    if get_stored_item_amount(save_id, item_id) and item.stackable: #Checks if you can stack
                                                the item
        update_stored_item_amount(amount, item_id, save_id)
    elif count_stored_items(save_id) >= max_inventory_size: #Full Inventory
        return "FullInventoryError"
```

```
    else:
        add_stored_item(save_id, item_id, amount)
    return True
```

remove_item():
This uses the SQL query to remove items from the users stored items in the stored_items SQL table. This is necessary for criteria 12.5.

```
def remove_item(item_id: 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) < amount: #Not enough items
            return "ItemQuantityError"
        if get_stored_item_amount(save_id, item_id) == amount: #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():
This uses the SQL query to unequip items from the users equipped item slots in the save_data SQL table. This is necessary for criteria 12.4.

```
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)
        if (add_item(item, 1) == "FullInventoryError"): #Adds the item back to the stored_items
                                                          and checks if the inventory is full
            return "FullInventoryError"
        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():
This uses the SQL query to equip items to the users equipped item slots in the save_data SQL table. This is necessary for criteria 12.4.

```
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
        unequip_item(slot)
        set_slot_value(slot, item, save_id) #Equips item
        remove_item(item) #Removes from stored_items
        return True
```

2.6.5 Testing Plan

Test #	Function	Parameters	Expected Outcome
12.5.1	add_item()	"test_item.tres", 2	Adds test item to the stored_items table.
12.5.2	add_item()	"test_item.tres", 3	As the item already exists it should add 3 to the amount.
12.5.3	add_item()	"test_weapon.tres", 1	As in the testing environment the max inventory size will be 1 and this should return "FullInventoryError"
12.5.4	remove_item()	"test_item.tres", 2	As more than the amount of the item is in the inventory it should subtract 2.
12.5.5	remove_item()	"test_item.tres", 10	"ItemQuantityError" as there isnt enough of the item in the database
12.5.6	remove_item()	"test_item.tres", 3	As exactly the amount is in the database the item entry should get removed.
12.5.7	remove_item()	"test_item.tres", 2	"ItemQuantityError" as there isnt any of the item in the database
12.4.1	unequip_item()	"head"	True and the head slot should remain as NULL
12.4.2	equip_item()	"test_helmet.tres"	True and the head slot should become "test_helmet.tres"
12.4.3	equip_item()	"test_helmet_2.tres"	True and the head slot should become "test_helmet.tres"
12.4.4	unequip_item()	"head"	"FullInventoryError"
12.4.5	unequip_item()	"head"	True as I will empty the inventory and the head should be NULL and the inventory should contain the helmet.

These tests will be used to evaluate to what extent I have met the criteria to which they are indexed and allow me to specifically identify where any issues in my code are and fix them. The test data is chosen such that the inventory will end up full at certain points due to the testing environments custom max inventory size aswell as testing wether removing all of an item correctly removes its entry from the database. It also is chosen to test the equipping and unequipping of items in the correct slots.

2.7 Player Character

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

2.7.1 Composition

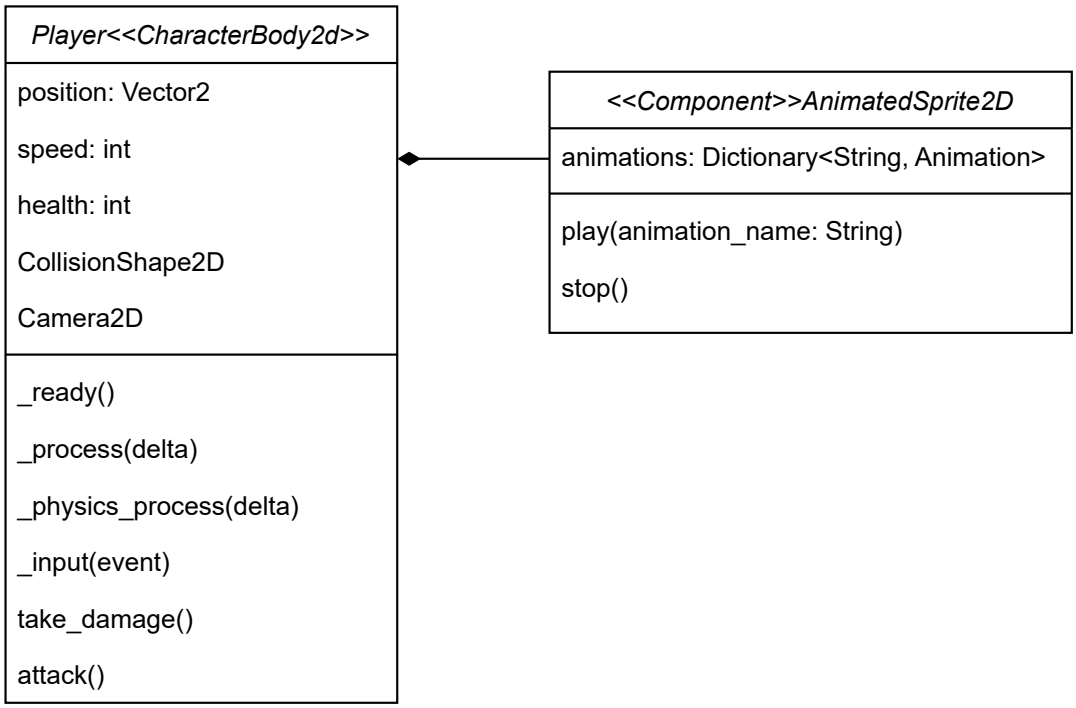


Figure 12: Player Class Diagram

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.

2.7.2 Animations

I have an animation set for the player that includes 8 Directional top down animations for all player actions and so how I will decide which directional animation will be based of the last direction the player walked.
I will use a get_animation function in order to get the directional animation to play.

2.7.3 Help Screen

I have decided to add a simple Help Screen in order to allow the user to check the controls when they want a reminder. I will implement this by creating a seperate scene with a label and then when the help button is pressed I will pause the tree and instance the scene before waiting for the help button to be pressed again to unpause the scene tree and queue_free the label.

2.7.4 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. It is necesarry for setting up key parts of the combat system and movement system.

```
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 frame and is usually used to deal with movement and physics processes. This is necesarry for handling criteria 5.2 aswell as movement of the player.

```
def _physics_process(delta):
   direction = Input.get_vector("left", "right", "up", "down")
   velocity = direction * speed
   if direction:
      last_direction = direction
      anim.play(get_animation("run"))
   else:
      anim.play(get_animation("idle"))
   move_and_slide()
```

_input(event):
This function handles all interrupts from player inputs I use it to call the attack function if the attack action is pressed aswell as loading the help scene and waiting till the key is pressed to deload it.

```
def _input(event):
   if event.is_action_pressed('attack'):
      attack()
   if event.is_action_pressed('help'):
      help_screen = preload("path/to/help/scene").instantiate()
      add_child(help_screen)
      get_tree().paused = True
      while not Input.is_action_just_pressed("help"):
         sleep(0.01)
      help_scene.queue_free()
      get_tree().paused = false
```

attack():
This handles the attack animation for criteria 5.4 aswell as calling the weapons attack function.

```
def attack():
   anim_player.play(get_animation("melee"))
   Inventory.get_weapon().attack(last_direction)
```

get_animation():
This is used to get the animation fo the type corresponding to the direction the player is facing. This is necesarry for criteria 5.3

```
def get_animation(animation_type):
   anim = animation_type + "_"
   if direction.x:
      if direction.x == 1:
         anim += "r"
      else:
         anim += "l"
   if direction.y:
      if direction.y == 1:
         anim += "d"
      else:
         anim += "u"
   return anim
```

2.8 Weapon Hurtbox

I will use an Area2D node with a capsule shape CollisionShape2D node attached oriented in order to encompass the sword swing area and then I will rotate it around the player dependent on the direction of attack. It will have variables for the damage and damage_type

2.8.1 Algorithms

_ready():
This script is used to get the bodies overlapping and if they are enemies call their take_damage function using polymorphism to decide the effect this will have on the specific enemy. This is necesarry for handling attack collisions and enemies being able to track their health for criteria 4.5.

```
def _ready():
    bodies = self.get_overlapping_bodies() #Get bodies in the area
    for body in bodies:
        if body.is_in_group("enemies"): #Damages bodies if they are an enemy.
            body.take_damage(damage, damage_type)
```

2.9 Dungeon Environment Design

2.9.1 Generic Tiles

I will use a TileMap node as that will allow me to import a spritesheet for a tilemap and define the properties of the tiles such as hitboxes and place them down easily. This will be used for criteria 3.1 and 3.2.

2.9.2 Chests

I will implement Chests, the basic structure of the chest will be a StaticBody2D with a circular area2D node to tell if the player is within range to open the chest aswell as the chest sprite. Each chest will have a key path for the key needed to unlock it. The chests will have a loot pool given by a dictionary where the key is the item path and the value relates to the ratio of getting it. There will also be a variable for the amount of items given by the chest and upon opening the chest will give that many items. This is necesarry as it comes together with the key item to fulfull all of criteria 3.3.

2.9.3 Doors

I will implement doors, the basic structure of the door will be using a StaticBody2D with an Area2D Node to detect if the player is infront of the door. Each door will have a key path for the key needed to unlock it. If the player is infront of the door and if the I key is pressed the door will either dissappear revealing another room or switch the scene to another room depending on the purpose. I will export the variables for the scene to go to and wether it just dissappears or changes scene. This is all necesarry as it comes together with the key item to fulfill all of criteria 3.5

2.9.4 Algorithms

Chest Script:
This will consist of the exported variables aswell as an input function that checks if the requirements to unlock the chest are met before generating the random items based on their probabilities which will be set up using an array and after the chest is opened then it will dissappear.

```
@export item_pool: dict
@export item_number: int
@export key_path: string

#Constructing the item_list
def _ready():
    item_list = []
    for item in item_pool: #For all items
```

```
    item_list += [item for x in range(item_pool[item])] #Adding on the amount relevant to
                                                        the ratio of the item

def _input(event):
   if event.is_action_pressed("enter") and Area2D.overlaps_body("player"): #Check if player
                                                       within range and pressed button
      if Inventory.remove_item(key_path,1) is bool: #Check if player has the key to unlock it
         for x in range(item_number):
            Inventory.add_item(item_list[randint(0,len(item_list)-1)])
        queue_free()
```

Door Script:
This consists of the variables and input function that checks if they right key is pressed then it checks if the player is in the Area2D Node and if so will check if the player has the correct key and then either dissappear or load another scene. This is the algorithm to allow criteria 3.5 to be fulfilled.

```
@export change_scene: bool
@export scene_path: string
@export key_path: string

def _input(event):
   if event.is_action_pressed("enter") and Area2D.overlaps_body("player"): #Check if player
                                                       within range and pressed button
      if Inventory.remove_item(key_path,1) is bool: #Check if player has the key to unlock it
         if change_scene:
            get_tree().change_scene_to_file(scene_path) #Change Scene
         else:
            queue_free() #Dissappear
```

2.10 Projectile Design

2.10.1 Overview

I will create two types of projectiles, one ranged type which will deal damage on impact and dissappear and one area of effect which will deal damage on an enemy or the player entering it and spawn next to the player/enemy that casts it. These projectiles will allow me to fulfill criteria 2.4, 2.5, 2.6, 4.7 and be used for the creation of boss enemies in 4.8.

2.10.2 Ranged

This will have the damage and damage_type variables to be used to call other entities take_damage functions, a variable so it only impacts once called attacking and a speed to determine how fast it goes. It will be comprised of a CharacterBody2D node and a AnimatedSprite2D node which will play the default animation until collision.
Its physics process will handle movement in the direction of rotation (as projectiles can be fired in any direction) and handle collisions.

2.10.3 Area Of Effect

This will have the damage and damage_type variables to be used to call other entities take_damage functions and a time variable for how long it lasts. It will be comprised of an Area2D node and an AnimatedSprite2D node which will play the default animation.
The ready function will be used to await the timer timeout before queue_freeing the area.
The _on_body_entered function will be linked from the Area2D and will deal damage if the body is an enemy or the player.

2.10.4 Algorithms

_physics_process(delta):
For the ranged projectile this will handle movement in the direction of rotation aswell as collisions. This is necessary for the ranged bow and magic staff projectiles for criteria 2.4 and 2.5.

```
def _physics_process(delta):
    #Rotated Movement
    velocity.x = speed * cos(rotation)
    velocity.y = speed * sin(rotation)
    if not attacking:
        move_and_slide()
    #Collisions
    for i in range(get_slide_collision_count()): #Loops through collisions
        if not attacking:
            attacking = true
            collision = get_slide_collision(i)
            collider = collision.get_collider()
            if collider.is_in_group("player") or collider.is_in_group("enemies"): #Damages what
                                                    it hits if its an enemy or the player
                collider.take_damage(damage, damage_type)
            CollisionShape2D.disabled = true #Disable collision shape so cant block other
                                                    projectiles
            AnimatedSprite2D.play("impact")
            await AnimatedSprite2D.animation_finished #Wait for animation to finish
            queue_free()
```

_ready():
Will set the area of effect to dissappear after a certain time which is necessary for the area of affect attacks for criteria 2.6.

```
def _ready():
    await get_tree().create_timer(time).timeout
    queue_free()
```

_on_body_entered(body):
This is called by the Area2D node upon a body entering it with that body passed as a parameter so we can make it take damage if it needs to. This is necesarry so that the area of effect attacks properly allow the player and enemies to track their health for fulfilling criteria 2.6, 4.5 and 2.8.

```
def _on_body_entered(body):
    if body.is_in_group("enemies") or body.is_in_group("player"): # If its an enemy or player
                                                    it will damage It
        body.take_damage(damage, damage_type)
```

2.11 Enemy Design

2.11.1 Overview

For my basic enemies I have decided to go with different types of slimes representing different elements. The slimes will have animations and collisions aswell as a radius that they will detect the player and navigate towards them dealing damage upon impact. Each unique slime will deal a different damage type and have a different look.

2.11.2 Composition

The root node of the slimes will be a CharacterBody2D node to allow for a physics body that can be easily moved via the script. I will then have a CollisionShape2D (criteria 4.4) for collision detection aswell as an Area2D node for detecting the player. It will contain an AnimatedSprite2D for the animations.

Identifier	Data Type	Justification
speed	Exported Integer	This allows slimes to have variable speeds to allow for difficulty increase.
health	Exported Integer	This allows for slimes to have variable healths to allow for difficulty increase.
damage	Exported Integer	This allows for slimes to deal variable damage to allow for difficulty increase.
damage_type	Exported String	This allows for slimes to deal different damage types to pose a different challenge.
direction	Vector2	This is used for direction to move in and the type of animation to play.
weaknesses	Exported List	Shows which damage_types to take more damage from.

2.11.3 Navigation

Upon researching I found the godot documentation on 2D navigation(6) and decided to use the NavigationAgent2D node to utilise the A* algorithm for criteria 4.2, this means that I would need to add a navigation

layer to my tileset to show the areas that the navigation agent can use. I decided to use a Timer node that autostarts and repeats and update the navigation path on the timeout so that the player can evade the slimes to a certain extent.
Ontop of this I used a circular CollisionShape2D with and Area2D node to detect when the player is within a certain range and pathfind towards them.

2.11.4 Animation

I decided to add idle animations for all of the 4 cardinal directions aswell as idle and hurt animations. I used a get_animation() function to get the relevant animation based on the direction. These animations are necesarry for the player to be able to tell what the enemy is doing aswell as seing them as part of criteria 4.1.

2.11.5 Projectile Enemies

There will be some enemies that will shoot a projectile instead of navigating in the direction given by the NavigationAgent2D so that the targetting would only update every couple seconds. These enemies will be non moving enemies and are necesarry for criteria 4.7.

2.11.6 Algorithms

get_animation(animation_type):
This gets the animation based on the type and direction.

```
def get_animation(animation_type: String):
  if abs(direction.x) > abs(direction.y):
    if direction.x > 0:
      return animation_type + "_r"
    else:
      return animation_type + "_l"
  else:
    if direction.y > 0:
      return animation_type + "_d"
    else:
      return animation_type + "_u"
```

_ready():
This is ran on the enemies addition to the scene tree to add the slime to the enemies group. This will allow the enemies to take damage from the players attack helping fulfill criteria 4.4 and 4.6.

```
def _ready():
   add_to_group("enemies")
```

_physics_process(delta):
Melee Enemies:
This checks if the player is within the detection range and if so moves towards them aswell as handling animations and detecting if the slime collides with the player so that the player will be damaged. This script is necessary for the movement, detection and melee attack of the enemies helping fulfill criteria 4.2, 4.3 and 4.6.

```
def _physics_process(delta):
   move = false
   for body in Area2D.get_overlapping_bodies(): #Checking if player in the detection area
      if body.name == "Player":
         move = true
   if not animating: #If not playing a different animation
      if move: #If the slime can move
         direction = NavigationAgent2D.get_next_path_position().normalized() #Getting the
                                                          direction of the next point on the
                                                          path
         velocity = direction * speed
         AnimatedSprite2D.play(get_animation("walk"))
      else:
         AnimatedSprite2D.play(get_animation("idle"))
   move_and_slide() #moving
```

```
    if can_attack: #If the attack cooldown is done
        for i in range(get_slide_collision_count()): #Loops through collisions
            collision = get_slide_collision(i)
            collider = collision.get_collider()
                if collider.is_in_group("player"): #Checks if collision is with the player
                    collider.take_damage(damage, damage_type) #Deals damage
                    can_attack=false
                    AttackTimer.start() #Starts attack cooldown
```

Ranged Enemies:
This checks if the player is within detection range and if so will fire a projectile towards them. This is necesarry for Enemy sight range and ranged attacks aswell as the pathfinding for aiming at the player helping fulfill criteria 4.2, 4.3 and 4.7.

```
def _physics_process(delta):
    var player_detected = false
    for body in Area2D.get_overlapping_bodies: #Checking if the player is in the detection
                                                Area
        if body.name == "Player":
            player_detected = true
    if not animating: #If not playing a different animation
        AnimatedSprite2D.play(get_animation("idle"))
        if player_detected and can_attack: # Will instantiate a projectile scene aimed at the
                                             player if it can attack and detects the
                                             player
            direction = NavigationAgent2D.get_next_path_position.normalized()
            var projectile = load(projectile_scene_path).instantiate()
            projectile.rotation_degrees = direction.angle()
            projectile.position = position + 20*direction
            projectile.damage = damage
            get_parent.add_child(projectile)
            can_attack = false
            AttackTimer.start() #Starts the attack cooldown
```

take_damage(damage, damage_type):
The take damage function will allow the enemies to take more or less damage based on weaknesses and update their health aswell as applying knockback and checking to see if the enemy should die. This is necesarry for criteria 4.5.

```
def take_damage(damage, damage_type):
    player = get_tree().get_first_node_in_group("player")
    animating = true
    if damage_type in weaknesses: # If the enemy is weak to a specific damage type then they
                                    will take more damage
        health -= 2*damage
    else:
        health -= damage
    velocity = - 25 * player.global_position.normalized() # Move away from the player for
                                                            knockback
    if health <= 0:
        queue_free()
    else:
        AnimatedSprite2D.play(get_animation("hurt"))
        await AnimatedSprite2D.animation_finished
        animating = false
```

_on_navigation_timer_timeout():
This will run when the navigation timer runs out and only update the navigation then to allow the enemies to not have perfect tracking so the player can avoid them easier. This is necesarry for updating the pathfinding for criteria 4.2.

```
def _on_navigation_timer_timeout() -> void:
    player = get_tree().get_first_node_in_group("player")
    NavigationAgent2D.set_target_position(player.global_position)
```

_on_attack_timer_timeout():
Thus function will run when the attack timer ends (1 second after an attack) to allow the enemy to attack again. This is necesarry to not allow the enemies to continously attack the player which would make the game

unplayable.

```
def _on_attack_timer_timeout():
    can_attack = True
```

2.12 UI

2.12.1 Display Strings

In order to allow for better displaying of items I have decided to add a display string function to all items that will return nicely formatted key information about the item that can be displayed. This will help with displaying items for criteria 7.3 and 12.1.
Weapon:

```
def display_string():
    return f"Weapon\nName: {resource_name}\nType: {weapon_type}\nDamage: {attack_power}\
                                                nDamage Type: {damage_type}\n Description:
                                                {description}"
```

Armour:

```
def display_string():
    return f"Armour\nName: {resource_name}\nType: {armour_type}\nDefense: {defense}\n
                                                Description: {description}"
```

Charm:

```
def display_string():
    return f"Charm\nName: {rescource_name}\nType: {charm_type}\n Description: {description}"
```

Key:

```
def display_string():
    return f"Key\nName: {resource_name}\n Description: {description}"
```

2.12.2 Inventory UI

The Inventory UI will be displayed upon pressing the E key (detected in the global script).
For the Inventory UI the important things to display are the stored items and the equipped items
For equipped items I decided to use labels displaying the items display strings.
For stored items I decided to use a ScrollContainer with a VBoxContainer and then append HBoxContainers with 4 buttons per container to display item display strings to create a 4 wide vertically scrolling system. I would have a selected script variable to be changed when a button is pressed and keep track of the last one pressed aswell as buttons to bin or equip the selected item.

2.12.3 Game UI

For the Game UI the important things to display are the player health and magic points aswell as the equipped weapon.
For the player health and magic points I have decided to use Progress Bars with custom textures to get the correct colours that have their maximum value set on going into the level and update each frame. These will aloow the players to see the percentage of their total health they have left.
For the weapon I have decided to display the display string in the bottom right corner. I will exclude the description of the weapon as this will take up too much screen space when displayed.

2.12.4 Algorithms

Inventory UI refresh():
This function will be called to setup the ui and refresh the ui every time an item is equipped or binned. It will set up the scrolling inventory aswell as the labells for equipped items. This is necesarry to see the stored items to remove for criteria 12.5 aswell as criteria 12.1.

```
def refresh():
    inventory = Database.get_stored_items()
    item_count = 0
    for child in ScrollContainer.VBoxContainer.get_children(): #Get rid of existing
                                                            HBoxContainers
        child.queue_free()
    for item in inventory: # Creates a button for each item in inventory with the display
                                                            string
        var button = Button.new()
        if item_count % 4 == 0:
            hbox = HBoxContainer.new()
            ScrollContainer.VBoxContainer.add_child(hbox)
        button.text = load(item["item_id"]).display_string() + f"\nAmount: {item["amount"]}"
        button.connect("pressed",_select.bind(item["item_id"]))
        hbox.add_child(button)
        item_count += 1

    # Loads the equipped items display strings and displays them
    if Database.get_slot_value("weapon"):
        WeaponLabel.text = load(Database.get_slot_value("weapon")).display_string()
    if Database.get_slot_value("head"):
        HeadLabel.text = load(Database.get_slot_value("head")).display_string()
    if Database.get_slot_value("chest"):
        ChestLabel.text = load(Database.get_slot_value("chest")).display_string()
    if Database.get_slot_value("legs"):
        LegsLabel.text = load(Database.get_slot_value("legs")).display_string()
    if Database.get_slot_value("charm_1"):
        Charm1Label.text = load(Database.get_slot_value("charm_1")).display_string()
    if Database.get_slot_value("charm_2"):
        Charm2Label.text = load(Database.get_slot_value("charm_2")).display_string()
```

Inventory UI _select():, equip button and bin button
The select and button functions make use of script variables or exisiting inventory functions to perform their purposes. These are necesarry for equipping and removing items for criteria 7.5, 12.4 and 12.5.

```
def _select(item_id):
    selected = item_id

def _on_equip_button_pressed():
    if selected != "":
        Inventory.equip_item(selected)
        refresh()

def _on_bin_button_pressed():
    if selected != "":
        Database.remove_stored_item(selected)
```

Game UI script
The Game UI script will consist of a _ready function that sets the maximum player health and magic points, an update_ui function that is called in the _ready function and a _process function that is called every frame to update the current player health and magic points. This is necesarry for displaying and updating the healthbar, magic points bar and display of the weapon being used for criteria 7.1, 7.2 and 7.3.

```
def _ready():
    player = get_tree().get_first_node_in_group("player")
    HealthBar.max_value = player.health
    MagicBar.max_value = player.mana
    update_ui()

def _process(delta):
    player = get_tree().get_first_node_in_group("player")
    HealthBar.value = player.health
    MagicBar.value = player.mana
```

```
def update_ui():
    if Database.get_slot_value("weapon"):
        weapon = load(Database.get_slot_value("weapon"))
        WeaponLabel.text = weapon.display_string()
```

2.13 Procedural Generation Design

2.13.1 Overview

The concept I most liked the idea of for procedural generation came from my research into existing solutions in which I found a dev log on procedural generation in Dead Cells$_{(3)}$. This dev log outlines how they use this idea of smaller handmade 'tiles' that can be peiced together using a skeletal structure of the level represented by a graph where each node is a room or corridor that can be selected randomly from a group of this type. I decided to take a similar approach as this allowed for a more handmade feel with similar lengths from the start to the finish of a level but still having a different level each time.

2.13.2 Graph Design

Firstly I needed to design the graph data structure I would use for my level design, I decided that each instance of the DungeonGraphNode class would have entrances/exits on the north, south, east and west of the room and that there would be the oppurtunity for another room to connect to these entrances. Because of this my DungeonGraphNodes will have specific variables to point to the rooms in each of these directions using aliasing. The only other thing needed to be stored in each room/node was the room type which I will store as a string.
The nodes themselves will not have any methods and I will use a DungeonGraph class to manage the graph for each level and all of the DungeonGraphNode objects associated with it.
In the DungeonGraph I will have an Array storing all the current nodes aswell as a Dictionary to store a list of rooms for each room type for when generating. I will also have a Dictionary to get the opposite direction of north, south, east and west.
The setup of the dungeon graph will have a new root node declared with room type "start" and appended to the node list, every node after will have to specify which node in the list of nodes it wants to be put next to, what direction and the room type of that node before being added on.
The generate dungeon function will perform a pre-order depth first search of the graph generating the rooms when they are processed.

2.13.3 Room Design

The Rooms will have Node2D's at each of the entrances to allow the positions to be got easily in order to align them with all the entrances being the same size. Their will be a get_pos() and set_pos() function for each of the directions to move the positions of those entrances so that they can line up with each other.
The rooms will also have a cap function to cap the entrance in a certain direction if there is no room attached there.
Finally the rooms will also use the _ready() function to set all the slimes health and damage according to the level.

2.13.4 Algorithms

DungeonGraphNode Script:
This sets up the necesarry variables.

```
class_name DungeonGraphNode
north: DungeonGraphNode
south: DungeonGraphNode
east: DungeonGraphNode
west: DungeonGraphNode
room_type: str
```

DungeonGraph _init():
Used to setup the root node. Necessary so that the Dungeon has a defined start.

```
def _init():
    root = DungeonGraphNode.new()
    root.room_type = "start"
    nodes.append(root)
```

DungeonGraph add_node(onto_index, direction, room_type):
Used to add a new DungeonGraphNode to the DungeonGraph in the direction specified onto the DungeonGraphNode at the index specified with the room type specified. Is necesarry for building level structure and requirements for criteria 10.1 and 10.4.1.

```
def add_node(onto_index, direction, room_type):
    onto = nodes[onto_index]
    if onto[direction]:
        return false
    new_node = DungeonGraphNode.new()
    new_node.room_type = room_type
    onto[direction] = new_node
    new_node[direction] = onto
    nodes.append(new_node)
    return true
```

DungeonGraph gen_node(node, previous_direction, previous):
Generates the room associated with the node using the direction it came from to line up its entrance with the previous room and then return the room generated. This is necesarry for the generation of each room of the dungeon from the graph for criteria 10.3.

```
def gen_room(node,previous_direction = null, previous = null):
    room = load(rooms[node.room_type].pick_random()).instantiate() #Picks the random room from
                                                    the list of that type and instantiates it
    add_child(room)
    if previous_direction: #Sets the position to align the entrances if the previous room
                                                   exists
        room.set_pos(previous_direction,previous.get_pos(opposite_direction[previous_direction]
                                                     ))
    for i in ['north', 'south', 'east', 'west']: #Caps the entrances with no rooms attached
        if node[i] == null:
            room.cap(i)
    return room #returns the generated room
```

DungeonGraph gen_dungeon(node, previous_direction, previous, generated):
This is a recursive implementation of a depth first search passing the previous_direction and previous room generated to be used for the gen_room function and using the generated list to keep track of which nodes have been processed. This is necesarry to traverse the graph structure getting each room to generate into the dungeon and where to generate it for criteria 10.3.

```
def gen_dungeon(node=root, previous_direction = null, previous = null, generated = []):
    room = await gen_room(node, previous_direction, previous) #Generates current room and
                                                   stores it
    generated.append(node) #Adds the node to the generated list
    for i in ['north', 'south', 'east', 'west']: #Checks each direction from the generated
                                                   node for ungenerated nodes
        if node[i] and node[i] not in generated:
            generated = await gen_dungeon(node[i], opposite_direction[i], room, generated) #
                                                           Recursive call on ungenerated nodes
                                                           passing in the list of nodes that
                                                           have been generated
    return generated #Returning generated to make sure it stays updated
```

Room get_pos(direction):
Used to get the global position of the directional Node2D markers. This is necesarry for the generation of the rooms and peicing together for criteria 10.3 and 10.2.

```
def get_pos(direction):
    match direction:
```

```
        'north':
            return North.global_position
        'south':
            return South.global_position
        'east':
            return East.global_position
        'west':
            return West.global_position
```

Room set_pos(direction, global_pos):
Used to set the position of the room such that the Node2D corresponding to the direction is at the global_pos provided. This is necesarry for the generation of the rooms and peicing together for criteria 10.3 and 10.2.

```
def set_pos(direction, global_pos):
    match direction:
        'north':
            global_position += global_pos - North.global_position
        'south':
    global_position += global_pos - South.global_position
        'east':
    global_position += global_pos - East.global_position
        'west':
    global_position += global_pos - West.global_position
```

Room cap(direction):
Loads the cap scene for the direction and then adds it to the entrance in that direction, blocking it off. This is necesarry when generating the level to make sure there is no entrances leading nowhere for criteria 10.3.

```
def cap(direction):
    match direction:
        'north':
    north_cap = load(north_cap_scene_path).instantiate()
    North.add_child(north_cap)
        'south':
    south_cap = load(south_cap_scene_path).instantiate()
    South.add_child(south_cap)
        'east':
    east_cap = load(east_cap_scene_path).instantiate()
    East.add_child(east_cap)
        'west':
    west_cap = load(west_cap_scene_path).instantiate()
    West.add_child(west_cap)
```

Room _ready():
Used to set slime health/damage with a logarithmic function of the difficulty so the rate of health./damage increase decreases as difficulty goes up. This is necesarry for increasing difficulty for criteria 14.2.1.

```
def _ready():
    if Slimes:
        for slime in Slimes.get_children():
    slime.health = floor(2*Global.current_level*log(3*Global.difficulty))
    slime.damage = floor(Global.current_level*log(3*Global.difficulty))
```

3 Development & Testing

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

    db.query("PRAGMA foreign_keys = ON;")
```

Figure 13: _ready

In the database script db is declared using *SQLite.new*(). 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 using godot's _exit_tree() function.

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 14: 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 15: 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 as before there was not enough different salts to properly prevent rainbow tables.

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 16: 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 17: 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 18: reset_password

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

3.1.4 Testing

```
func test_database():
    #Criteria 6.
    print("6.1:")
    print((len(db.gen_salt()) == 64) and (db.gen_salt() != db.gen_salt())) # Test 6.1.1
    print(len(db.j_hash("password", "salt")) == 64) # Test 6.1.2
    print(db.j_hash("password", "salt") == db.j_hash("password", "salt")) # Test 6.1.3
    print(db.j_hash("password", "salt") != db.j_hash("Password", "salt")) # Test 6.1.4

    print("6.3")
    print(db.add_user("Hyrule", "Password", "Answer")) # Test 6.3.1
    print(db.add_user("Hyrule", "Password", "Answer") == "InvalidUsernameError") # Test 6.3.2

    print("6.7,6.8,6.10")
    print(db.login("Hyru1e", "Password") == "InvalidUsernameError") # Test 6.7.1
    print(db.login("Hyrule", "Password")) # Test 6.7.2
    print(db.reset_password("Hyru1e", "Answer", "password") == "InvalidUsernameError") # Test 6.8.1
    print(db.reset_password("Hyrule", "answer", "password") == "IncorrectAnswerError") # Test 6.8.2
    print(db.reset_password("Hyrule", "Answer", "password")) # Test 6.8.3
    print(db.login("Hyrule", "Password") == "IncorrectPaswordError") # Test 6.7.3
```

Figure 19: test_database.gd

This testing function was implemented as detailed in my testing plan that I designed.

Test #	Function	Parameters	Expected Outcome	Actual Outcome
6.1.1	gen_salt()		random 256 bit hex string	Success
6.1.2	hash()	”password”, ”salt”	random 256 bit hex string	Success
6.1.3	hash()	”password”, ”salt”	the same random 256 bit hex string	Success
6.1.2	hash()	”Password”, ”salt”	random 256 bit hex string different from before	Success
6.3.1	add_user()	”Hyrule”, ”Password”, ”Answer”	True	Success
6.3.2	add_user()	”Hyrule”, ”Password”, ”Answer”	”InvalidUsernameError” as a user already exists with that username	Success
6.6.1	login()	”Hyru1e”, ”Password”	”InvalidUsernameError”	Success
6.6.2	login()	”Hyrule”, ”Password”	True	Success
6.7.1	reset_password()	”Hyru1e”, ”Answer”, ”password	”InvalidUsernameError”	Success
6.7.2	reset_password()	”Hyrule”, ”answer”, ”password”	”IncorrectAnswerError”	Success
6.7.3	reset_password()	”Hyrule”, ”Answer”, ”password”	True	Success
6.6.3	login()	”Hyrule”, ”Password”	”IncorrectPasswordError”	Success

Upon Testing I realised I needed a delete user function so that the user can be deleted.
I designed a simple SQL query and created a function to delete users.

```
var _delete_user = """
DELETE FROM users
WHERE username = ?;
"""
```

Figure 20: _delete_user

```
func delete_user(username, password):
    db.query_with_bindings(_get_user_data,[username])
    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
        db.query_with_bindings(_delete_user,[username]) # Deleting User
        return true
    return "IncorrectPasswordError" # If password doesnt match
```

Figure 21: delete_user

Test #	Function	Parameters	Expected Outcome	Actual Outcome
6.9.1	delete_user()	”Hyrule”, ”Password”	IncorrectPasswordError	Success
6.9.2	delete_user()	”Hyrule”, ”password”	True	Success
6.6.4	login()	”Hyrule”, ”password”	InvalidUsernameError	Success

```
print(db.delete_user("Hyrule", "Password") == "IncorrectPaswordError") # Test 6.10.1
print(db.delete_user("Hyrule", "password")) # Test 6.10.2
print(db.reset_password("Hyrule", "Answer", "password") == "InvalidUsernameError") # Test 6.7.4
```

Figure 22: Test Remove User

Upon Testing the remove functions in the database I have updated the table queries to add ON DELETE CASCADE so that if the user gets deleted all their saves get deleted.

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 23: Layout

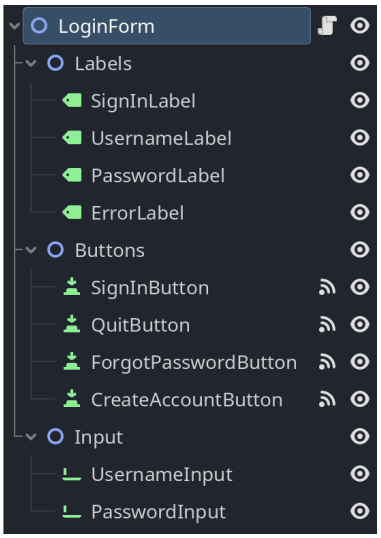


Figure 24: Structure

The layout and structure are as designed but I decided to group all the Labels, Buttons and Input Boxes using Node2D to have a neater 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 25: 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 26: _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 27: Layout

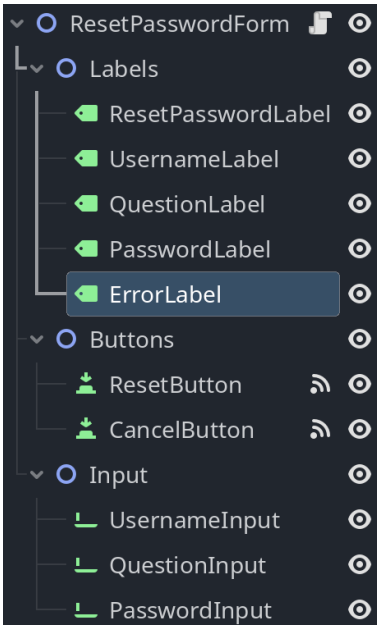


Figure 28: Structure

The layout and structure are as designed but I decided to group all the Labels, Buttons and Input Boxes using Node2D to have a neater structure.

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

Figure 29: _on_cancel_button_pressed

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

```
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 30: _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

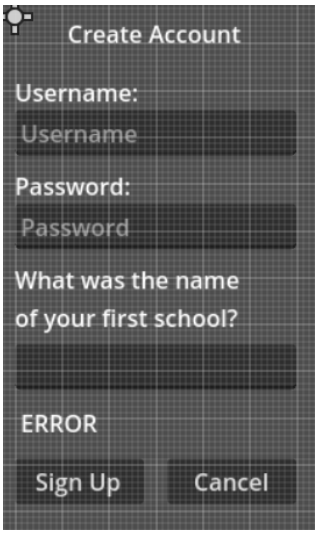


Figure 31: Layout

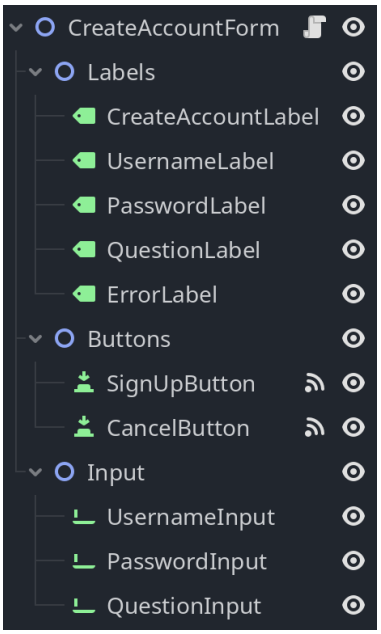


Figure 32: Structure

The layout and structure are as designed but I decided to group all the Labels, Buttons and Input Boxes using Node2D to have a neater structure.

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

Figure 33: _on_cancel_button_pressed

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

```
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 34: _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 Save Select System

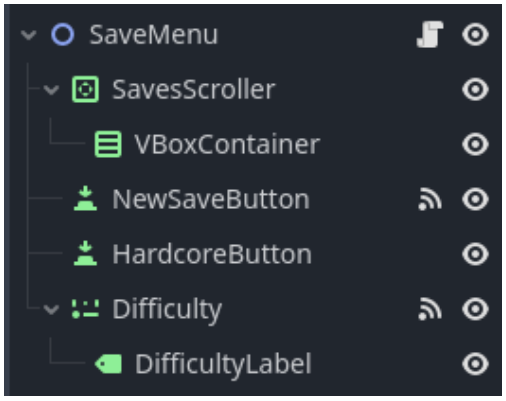


Figure 35: Structure

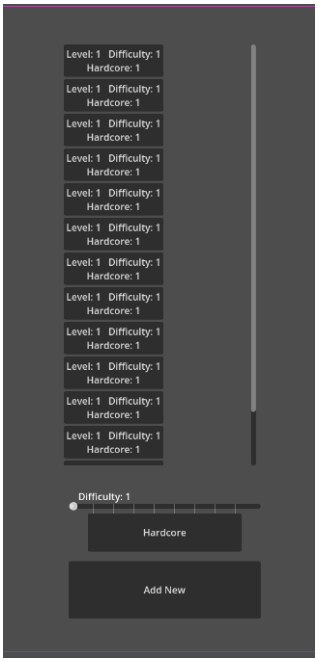


Figure 36: Layout

The structure is as was described with the VBoxContainer in the ScrollContainer to make sure the save buttons are only stacked vertically. There is also a label atatched to the Difficulty slider.
The layout is simple with the save list in the center and scrollbar next to it and the add new save elements underneath.

```
extends Node2D

#Function to load the list
func _load_list():
    var save_data_list = Database.get_user_save_data()
    for save_data in save_data_list:
        var button = Button.new()
        button.text = "Level: %s \t Difficulty: %s\nHardcore: %s" % [save_data["level"], save_data["difficulty"], save_data["hardcore"]]
        button.connect("pressed",_on_save_selected.bind(save_data))
        $SavesScroller/VBoxContainer.add_child(button)

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
    _load_list()

func _on_save_selected(save_data):
    Database.current_save_id = save_data["save_id"]
    #Link to current level when made


func _on_difficulty_drag_ended(value_changed: bool) -> void:
    $Difficulty/DifficultyLabel.text = "Difficulty: " + str($Difficulty.value)

func _on_new_save_button_pressed() -> void:
    Database.add_new_save_data($Difficulty.value,$HardcoreButton.button_pressed)
    _load_list()
```

Figure 37: Script

This script is similar to pseudocode with the string formatting changed to work in godot and the code to load the save list moved to another function so the save list can be reloaded every time a new save is added aswell as a function linked to the Difficulty slider to update the Difficulty label.

3.4 Video Testing (Login Forms)

I initially tested the login forms and save menu in the video here[2]. The Login Forms did their function perfectly however the save menu kept the previous save data in the list when calling the _load_list function for the second time. To fix this I added a loop at the start of the function to get rid of all the children of the VBoxContainer.

```
for child in $SavesScroller/VBoxContainer.get_children():
    child.queue_free()
```

Figure 38: Script Fix

I then tested the save menu again here[3] and it worked just fine.

3.5 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.

3.5.1 Folder Structure

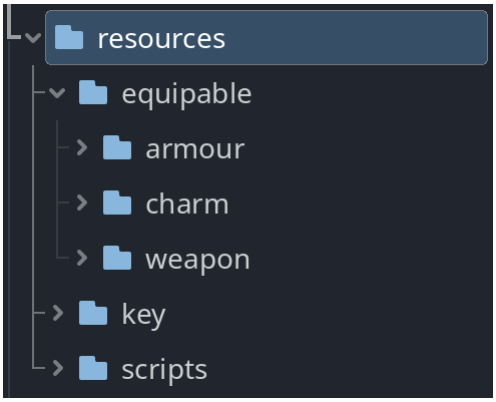


Figure 39: Folder Structure

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

3.5.2 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 40: Item Script

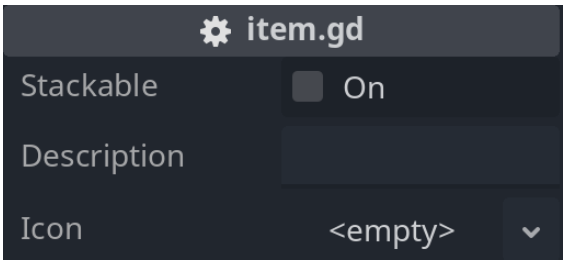


Figure 41: Item Exports

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.

3.5.3 Equipable

```gdscript
extends Item

class_name Equipable

@export var stat_boosts: Dictionary = {}

func _init():
    stackable = false
```

Figure 42: Equipable Script

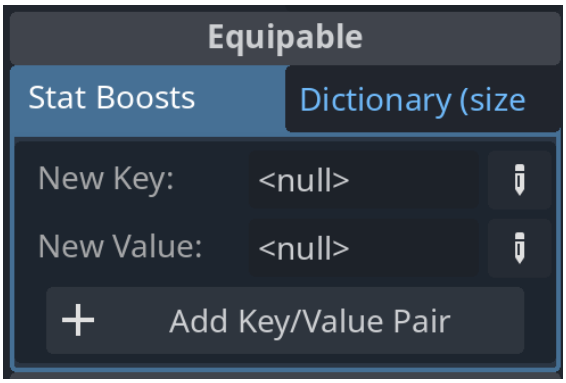


Figure 43: Equipable Exports

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.

3.5.4 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 44: Armour Script

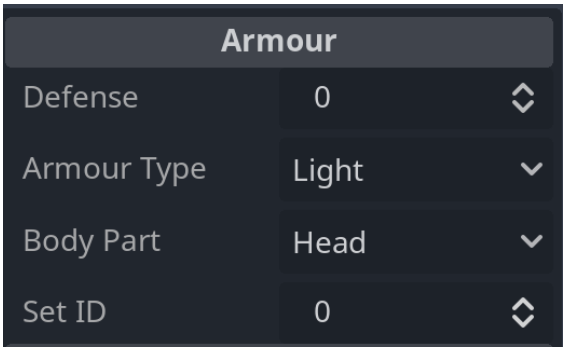


Figure 45: Armour Exports

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.

3.5.5 Charm

```gdscript
extends Equipable

class_name Charm

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

Figure 46: Charm Script

Figure 47: 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.

3.5.6 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 48: Weapon Script

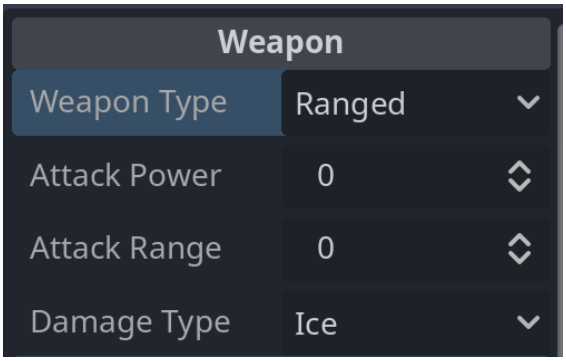


Figure 49: Weapon Exports

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

```gdscript
func attack(owner, direction: Vector2):
    #Load the hurtbox scene
    var hurtbox_scene = load(hurtbox_scene_path)
    var hurtbox_instance = hurtbox_scene.instantiate()
    hurtbox_instance.range = attack_range
    hurtbox_instance.damage = attack_power
    hurtbox_instance.damage_type = damage_type
    hurtbox_instance.rotation = direction.angle()
    #Add child
    owner.add_child(hurtbox_instance)
    #Hitbox lasts for a tenth of a second
    await owner.get_tree().create_timer(0.1).timeout
    hurtbox_instance.queue_free()
```

Figure 50: Weapon attack() function

This shows the weapon attack script that I implemented I changed the name from hitbox to hurtbox as that is more accurate and I had to use a timer in order to sleep for an amount of time that the hitbox will last for.

3.5.7 Key

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

class_name Key

@export var key_id: String
```

Figure 51: Key Script

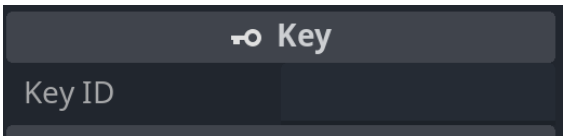


Figure 52: Key Exports

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.

3.5.8 Revision

Upon starting the inventory development I have decided to switch from enums to strings as the enums make it more complicated to access the string associated with the number.

3.6 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 execute the query passing in the current_save_id and return the query result.

3.6.1 Add Item

```
func add_item (item_id, amount):
    var item = load(item_id) # Loads the item
    if Database.count_stored_items() > max_inventory_size and not(item.stackable): # Full Inventory
        return "FullInventoryError"
    else :
        # Checks if you can stack the item and either adds a new entry or stacks it
        if len(Database.get_stored_item_amount(item_id)) != 0 and item.stackable: # If the item is in the database
            Database.update_stored_item_amount(amount, item_id)
        else :
            Database.add_stored_item(item_id, amount)
        return true
```

Figure 53: add_item()

The add_item function stayed mostly faithful to the pseudocode except I moved stuff arround for clarity.

3.6.2 Remove Item

```
func remove_item(item_id, amount):
    var amount_stored = Database.get_stored_item_amount(item_id)
    if len(amount_stored) != 0: # If the item is in the database
        amount_stored = amount_stored[0]["amount"]
        if amount_stored < amount: # Not enough items
            return "ItemQuantityError"
        elif amount_stored == amount: # Exactly enough items
            Database.remove_stored_item(item_id)
        else :
            Database.update_stored_item_amount(-amount, item_id) # Removes the ammount of that item
        return true # Indicates that it was successful
    return "ItemQuantityError"
```

Figure 54: remove_item()

The remove_item function is mostly the same as the pseudocode except I extracted some of the query data so I don't end up running repeat queries.

3.6.3 Unequip Item

```
func unequip_item(slot):
    # Gets the slot value
    var value = Database.get_slot_value(slot)
    # Checks if there is an item to unequip
    if value != null:
        if (add_item(value, 1) == "FullInventoryError"): # Adds the item back to the stored_items/
            return "FullInventoryError"
        Database.set_slot_value(slot, null) # Sets the slot back to null
        return true
    return true # If there is no item to unequip we have succeeded
```

Figure 55: unequip_item()

The unequip_item function is again close to the pseudocode except I extracted repeat queries to a variable.

3.6.4 Equip Item

```
func equip_item(item_id):
    var slot: String
    var item = load(item_id) # Loads the item
    # Checks if the item is Equipable
    if not(item is Equipable):
        return false
    # Gets the slot to equip it into
    if item is Armour:
        slot = item.body_part
    elif item is Weapon:
        slot = "weapon"
    else:
        if Database.get_slot_value("charm_1") == null:
            slot = "charm_1"
        else:
            slot = "charm_2"

    remove_item(item_id,1)
    var success = unequip_item(slot) # Checks the success of unequipping the item
    if not(success is bool) and success == "FullInventoryError":
        add_item(item_id,1)
        return "FullInventoryError"
    Database.set_slot_value(slot, item_id) # Equips the item
    return true
```

Figure 56: equip_item()

In the equip item function I decided to simplify the proccess of deciding which charm slot as it was unnecesarily complex and I also added a check for if the inventory is full making sure to still accept it if equipping the item leaves just enough room in the inventory.

3.6.5 Testing

```gdscript
func test_inventory():
    Database.add_user("test", "password", "answer")
    Database.login("test", "password")
    Database.add_new_save_data(1,true)
    Database.current_save_id = 1
    #Criteria 12
    print(Inventory.add_item("res://utils/test_item.tres", 2)) # Test 12.5.1
    print(Inventory.item_amount("res://utils/test_item.tres") == 2)
    print(Inventory.add_item("res://utils/test_item.tres", 3)) # Test 12.5.2
    Inventory.max_inventory_size = 1
    print(Inventory.add_item("res://utils/test_weapon.tres", 1) == "FullInventoryError") # Test 12.5.3
    print(Inventory.item_amount("res://utils/test_item.tres") == 5)
    print(Inventory.remove_item("res://utils/test_item.tres",2)) # Test 12.5.4
    print(Inventory.item_amount("res://utils/test_item.tres") == 3)
    print(Inventory.remove_item("res://utils/test_item.tres",10) == "ItemQuantityError") # Test 12.5.5
    print(Inventory.remove_item("res://utils/test_item.tres",3)) # Test 12.5.6
    print(Inventory.item_amount("res://utils/test_item.tres") == 0)
    print(Inventory.remove_item("res://utils/test_item.tres",2) == "ItemQuantityError") # Test 12.5.7
    print(Inventory.unequip_item("head")) # Test 12.4.1
    Inventory.add_item("res://utils/test_helmet.tres",1) #Test 12.4.2
    print(Inventory.equip_item("res://utils/test_helmet.tres"))
    print(Database.get_slot_value("head") == "res://utils/test_helmet.tres")
    print(Inventory.item_amount("res://utils/test_helmet.tres") == 0)
    Inventory.add_item("res://utils/test_helmet2.tres",1) # Test 12.4.3
    print(Inventory.equip_item("res://utils/test_helmet2.tres"))
    print(Inventory.item_amount("res://utils/test_helmet.tres") == 1)
    print(Inventory.unequip_item("head") == "FullInventoryError") # Test 12.4.4
    Inventory.remove_item("res://utils/test_helmet.tres",1) # Test 12.4.5
    print(Inventory.unequip_item("head"))
```

Figure 57: test_inventory()

I added the relevant test items in order to test the inventory functions with this script which I ran.

Test #	Function	Parameters	Expected Outcome	Actual Outcome
12.5.1	add_item()	"test_item.tres", 2	Adds test item to the stored_items table.	Success
12.5.2	add_item()	"test_item.tres", 3	As the item already exists it should add 3 to the amount.	Success
12.5.3	add_item()	"test_weapon.tres", 1	As in the testing environment the max inventory size will be 1 and this should return "FullInventoryError"	Fail
12.5.4	remove_item()	"test_item.tres", 2	As more than the amount of the item is in the inventory it should subtract 2.	Success
12.5.5	remove_item()	"test_item.tres", 10	"ItemQuantityError" as there isnt enough of the item in the database	Success
12.5.6	remove_item()	"test_item.tres", 3	As exactly the amount is in the database the item entry should get removed.	Success
12.5.7	remove_item()	"test_item.tres", 2	"ItemQuantityError" as there isnt any of the item in the database	Success
12.4.1	unequip_item()	"head"	True and the head slot should remain as NULL	Fail
12.4.2	equip_item()	"test_helmet.tres"	True and the head slot should become "test_helmet.tres"	Success
12.4.3	equip_item()	"test_helmet_2.tres"	True and the head slot should become "test_helmet.tres"	Success
12.4.4	unequip_item()	"head"	"FullInventoryError"	Success
12.4.5	unequip_item()	"head"	True as I will empty the inventory and the head should be NULL and the inventory should contain the helmet.	Success

The add_item function failed to return "FullInventoryError" as it only checked if the item wasnt stackable not if it wasnt stackabled and already stored so I updated the script.

```
func add_item (item_id, amount):
    var item = load(item_id) # Loads the item
    if Database.count_stored_items() >= max_inventory_size and not(item.stackable and item_amount(item_id) != 0):
        return "FullInventoryError"
    else :
        print("hi")
        # Checks if you can stack the item and either adds a new entry or stacks it
        if item_amount(item_id) != 0 and item.stackable: # If the item is in the database
            Database.update_stored_item_amount(amount, item_id)
        else :
            Database.add_stored_item(item_id, amount)
        return true
```

Figure 58: add_item() fixed

The remove_item failed due to a syntax error with the SQL statement so I edited the get_slot_value function to query for all values and then look it up I also updated the set_slot_value function to query the slot using a match case rather than bindings.

```
func get_slot_value(slot):
    db.query_with_bindings(_get_slot_values,[current_user_id, current_save_id])
    print(db.query_result)
    if len(db.query_result) == 0:
        return null
    return db.query_result[0][slot]
```

Figure 59: get_slot_value() fixed

```
func set_slot_value(slot, item_id):
    match slot:
        "head":
            db.query_with_bindings(_set_head_value, [item_id, current_save_id])
        "chest":
            db.query_with_bindings(_set_chest_value, [item_id, current_save_id])
        "legs":
            db.query_with_bindings(_set_legs_value, [item_id, current_save_id])
        "weapon":
            db.query_with_bindings(_set_weapon_value, [item_id, current_save_id])
        "charm_1":
            db.query_with_bindings(_set_charm_1_value, [item_id, current_save_id])
        "charm_2":
            db.query_with_bindings(_set_charm_2_value, [item_id, current_save_id])
    return db.query_result
```

Figure 60: set_slot_value() fixed

The unequip_item function failed due to not checking if success is a boolean before comparing it so I amended it.

```
func unequip_item(slot):
    # Gets the slot value
    var value = Database.get_slot_value(slot)
    # Checks if there is an item to unequip
    if value != null:
        var success = add_item(value, 1)
        if (not(success is bool) and success == "FullInventoryError"): #
            return "FullInventoryError"
        Database.set_slot_value(slot, null) # Sets the slot back to null
        return true
    return true # If there is no item to unequip we have succeeded
```

Figure 61: unequip_item() fixed

Retesting:

Test #	Function	Parameters	Expected Outcome	Actual Outcome
12.5.1	add_item()	"test_item.tres", 2	Adds test item to the stored_items table.	Success
12.5.2	add_item()	"test_item.tres", 3	As the item already exists it should add 3 to the amount.	Success
12.5.3	add_item()	"test_weapon.tres", 1	As in the testing environment the max inventory size will be 1 and this should return "FullInventoryError"	Success
12.5.4	remove_item()	"test_item.tres", 2	As more than the amount of the item is in the inventory it should subtract 2.	Success
12.5.5	remove_item()	"test_item.tres", 10	"ItemQuantityError" as there isnt enough of the item in the database	Success
12.5.6	remove_item()	"test_item.tres", 3	As exactly the amount is in the database the item entry should get removed.	Success
12.5.7	remove_item()	"test_item.tres", 2	"ItemQuantityError" as there isnt any of the item in the database	Success
12.4.1	unequip_item()	"head"	True and the head slot should remain as NULL	Success
12.4.2	equip_item()	"test_helmet.tres"	True and the head slot should become "test_helmet.tres"	Success
12.4.3	equip_item()	"test_helmet_2.tres"	True and the head slot should become "test_helmet.tres"	Success
12.4.4	unequip_item()	"head"	"FullInventoryError"	Success
12.4.5	unequip_item()	"head"	True as I will empty the inventory and the head should be NULL and the inventory should contain the helmet.	Success

All the tests came out as a success so that concludes my testing.

3.7 Hurtbox Development

3.7.1 Layout

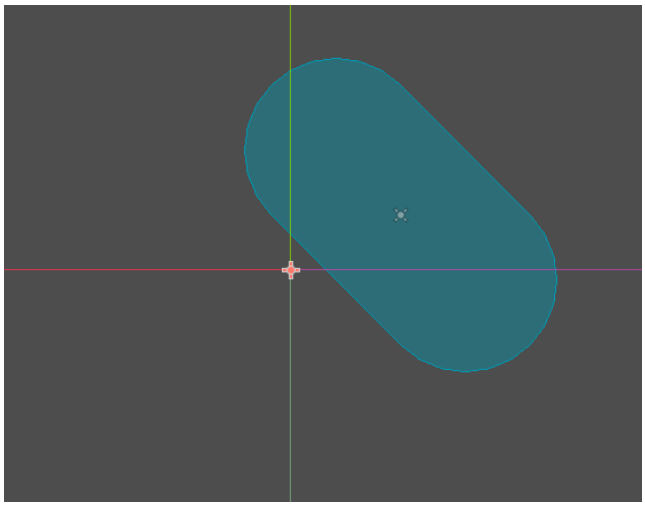


Figure 62: HurtBox Layout

I made the hurtbox like this as the center of the area2d node (the cross) can rotate allowing the actual collisionshape to rotate around the player depending on the direction of attack.

3.7.2 Script

```
extends Area2D

var damage: int
var damage_type: String
var range: int



func _on_body_entered(body: Node2D) -> void:
    if body.is_in_group("enemies"): # If its an enemy damages it
        body.take_damage(damage, damage_type)
```

Figure 63: Hurtbox Script

I changed the script from using the ready function to running every time a body enters the hurtbox as it allows the hurtbox to work for the duration of its existence rather than just at the start. Other than that the script is the same.

3.8 Player Development

3.8.1 Layout and Structure

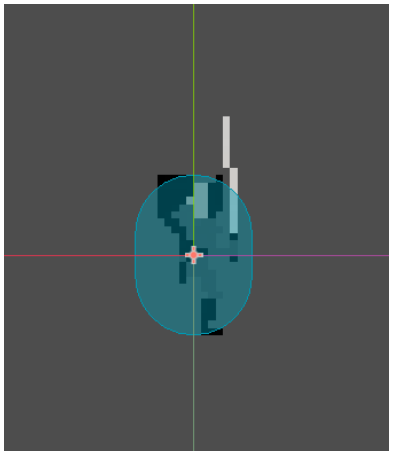


Figure 64: Layout

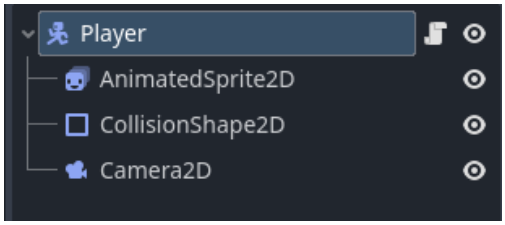


Figure 65: Structure

The structure is the same as was described in the class diagram and the layout is as such so that the player can interact with walls and enemy hits can register it.

3.8.2 _physics_proccess(delta):

```gdscript
func _physics_process(delta: float) -> void:
    # Get the direction
    var direction = Vector2(Input.get_axis("left", "right"),Input.get_axis("up","down"))
    # Velocity
    velocity = direction.normalized() * SPEED
    #Mocing/Idling if its not already animating
    if not animating:
        if direction:
            last_direction = direction
            $AnimatedSprite2D.play(get_animation("walk"))
        else:
            $AnimatedSprite2D.play(get_animation("idle"))
        move_and_slide()
```

Figure 66: _physics_process():

I decided to add an animating flag for use in this script that will be flagged when animations that cannnot be interrupted are playing (e.g. attacks) so that the player will not move or switch animations during that. Other than that the function is the same as the designed function with a slightly different method of getting direction.

3.8.3 Player Animation

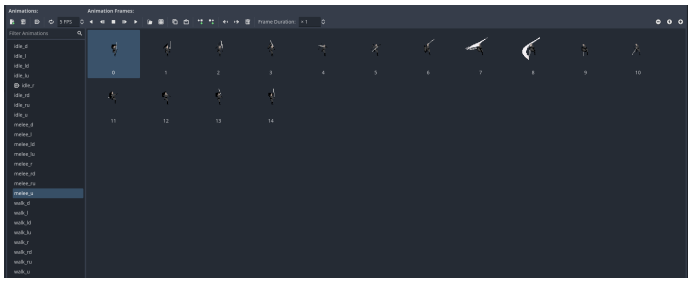


Figure 67: Animations

```gdscript
func get_animation(animation_type: String):
    var anim = animation_type + "_"
    if last_direction.x:
        if last_direction.x == 1:
            anim += 'r'
        else:
            anim += 'l'
    if last_direction.y:
        if last_direction.y == 1:
            anim += 'd'
        else:
            anim += 'u'
    return anim
```

Figure 68: get_animation():

I added the animation frames into seperate animations in the Player's animationPlayer naming them such that the different directional animation names can be got using the get_animation() function.

3.8.4 Player Attack

```gdscript
func _input(event: InputEvent) -> void:
    if event.is_action_pressed("attack"):
        attack()

func attack():
    if not animating:
        animating = true
        $AnimatedSprite2D.play(get_animation("melee"))
        load(Database.get_slot_value("weapon")).attack(self, last_direction)
        await $AnimatedSprite2D.animation_finished
        animating = false
```

Figure 69: Player Attack

I added code to the input function to call the player's attack function if the attack action is pressed which mostly does the same as the planned script except for setting the animating flag to true until it finishes animating to prevent the player from being able to attack twice or walk while attacking.

3.9 TileMap Development

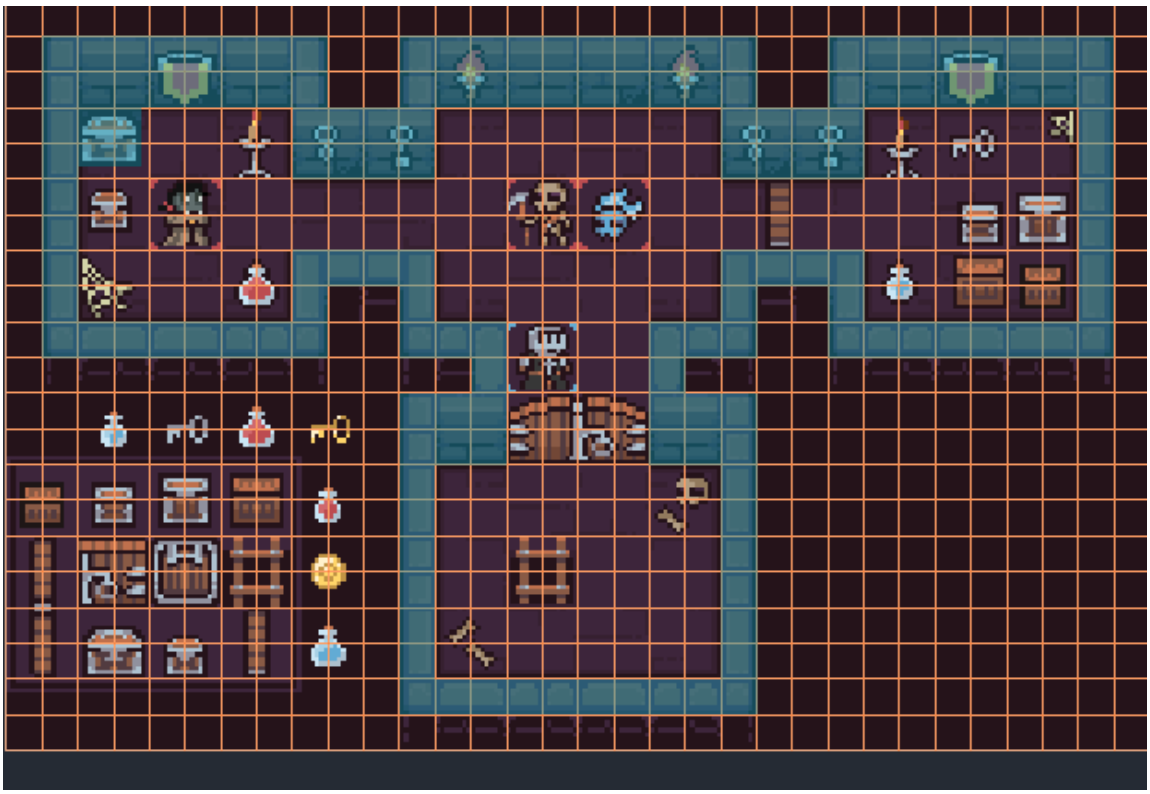


Figure 70: Physics Layer on TileMap

I added a TileMap using Godot's TileMap node and a free dungeon tileset painting a physics layer on the walls so that the player and enemies will collide with them. I can add this as a child to any scene and use it to place individual tiles and build level/room structures. I only added physics layers to the wall and floor tiles as I will only use them.

3.10 Video Testing (Movement and TileMap)

In the video here[1] I have documented my testing of the player movement, moving and idle animations aswell as the TileMap within a square test level I made using the tilemap.
The Player movement was a consistent speed, smooth and worked in all 8 directions the movement direction corresponded to the keybinds accurately (1.1-1.4).
The Player animation was updated correctly depending on wether I was walking or idle and the direction I was facing.
The TileMap tiles displayed correctly and the walls correctly blocked the player movement through them even in the corners.

3.11 Dummy Development

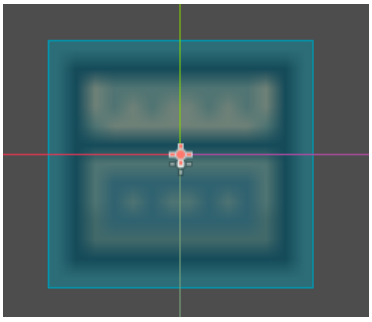


Figure 71: Layout

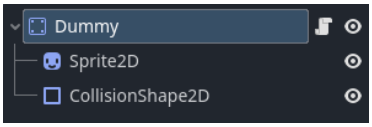


Figure 72: Structure

```
extends StaticBody2D

func _ready() -> void:
    add_to_group("enemies")

func take_damage(damage, damage_type):
    print(damage)
    print(damage_type)
```

Figure 73: Script

As part of my developmental testing I decided to create a simple dummy to test weapons on. The dummy consists of a StaticBody2D with a CollisionShape2D and a Sprite2D it also has a simple script for taking damage which I have set to print damage type and damage to the console.

3.12 Stakeholder Feedback 1

I got my first stakeholder feedback on 02/03/25 from Stakeholders Daniel and Samuel. I showed them my Login System and Save Select System aswell as player movement and got their feedback. The video can be found here[10].

3.12.1 Login System and Save Menu

During the Demo of the Login System and Save Menu Samuel got stuck on the Save Menu and had to restart to log out, to fix this I added a simple Logout Button to the Save Menu which switches the scene back to the login page to allow login as another account.

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

Figure 74: Logout Button

Stakeholders mentioned how they would like the addition of a confirm password box to make sure that no mistypes in the password would affect logging in and so I added this with a simple check to see if the entered fields are equal.

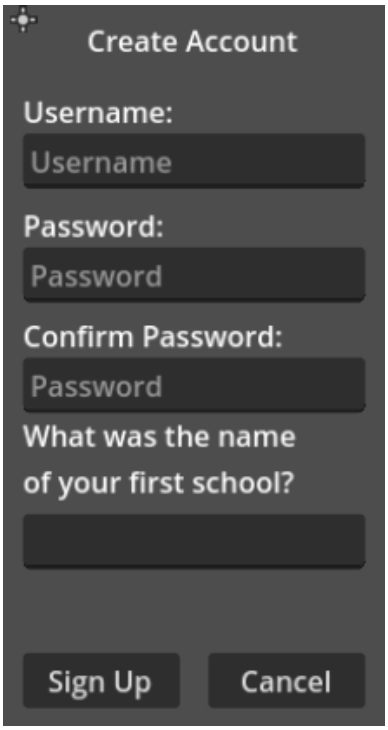


Figure 75: Confirm Password Field

```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 $Input/PasswordInput.text == $Input/ConfirmPasswordInput.text:
        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")
    else:
        $Labels/ErrorLabel.text = "Passwords Don't Match"
```

Figure 76: Updated Script

3.12.2 Test Scene

During the demo the stakeholders ran into an issue with the attacks where the attack function in the weapon scripts wasnt able to properly use await and timers due to them not being in the scene tree and so crashed the program when the player tried to attack, to fix this I moved the attack functions out of the weapon script and directly into the player's attack function sacrificing the polymorphism for the sake of time.

```gdscript
func attack():
    if not animating:
        animating = true
        $AnimatedSprite2D.play(get_animation("melee"))
        weapon = load(Database.get_slot_value("weapon"))
        if weapon.weapon_type == "melee":
            #Load the hurtbox scene
            var hurtbox_scene = load(weapon.hurtbox_scene_path)
            var hurtbox_instance = hurtbox_scene.instantiate()
            hurtbox_instance.range = weapon.attack_range
            hurtbox_instance.damage = weapon.attack_power
            hurtbox_instance.damage_type = weapon.damage_type
            hurtbox_instance.rotation = last_direction.angle()
            #Add child
            add_child(hurtbox_instance)
            #Hitbox lasts for a tenth of a second
            await get_tree().create_timer(0.1).timeout
            print("yes")
            hurtbox_instance.queue_free()
        await $AnimatedSprite2D.animation_finished
        animating = false
```

Figure 77: Updated Player Attack Function

3.13 Dash Development

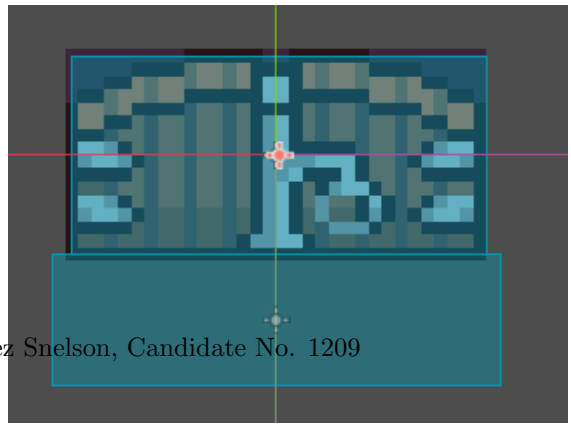
```gdscript
elif event.is_action_pressed("dash"):
    if can_dash: # Increases speed momentarily if you can dash
        speed = 8*speed
        can_dash = false
        await get_tree().create_timer(0.05).timeout #dash duration
        speed = speed/8
        await get_tree().create_timer(0.5).timeout #dash cooldown
        can_dash = true
```

Figure 78: Updated Player Attack Function

For the dash development I added a clause to the input function to check for the dash input (Q Key) and then if the player could dash I would increase the player speed aswell as setting can_dash to false momentarily before decreasing the speed it and waiting half a second for the dash cooldown before setting can_dash to true.

3.14 Dungeon Environment Development

3.14.1 Door



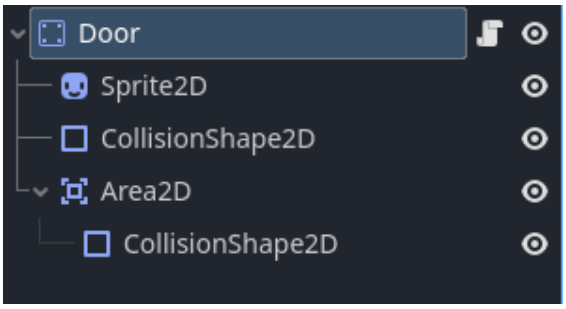


Figure 80: Structure

The layout and structure of the door is as shown above, mostly the same as was designed making sure to get the area2d infront of the door for opening.

```
extends StaticBody2D

@export var change_scene: bool
@export var scene_path: String
@export var key_path: String
@export var locked: bool

func _input(event: InputEvent) -> void:
    if event.is_action_pressed("interact"): # Check pressed button
        for x in $Area2D.get_overlapping_bodies():
            if x.name == "Player": # Check the player is within range
                if not(locked) or Inventory.remove_item(key_path,1) is bool: # Check if the player has the key and removes it if the door isnt locked
                    if change_scene:
                        get_tree().change_scene_to_file(scene_path) # Change Scene
                    else:
                        queue_free()
```

Figure 81: Script

I added a variable that detemines wether or not the door is locked aswell as looping through the overlapping bodies instead of checking if it overlaps a certain body as that was easier to implement.

3.14.2 Chest

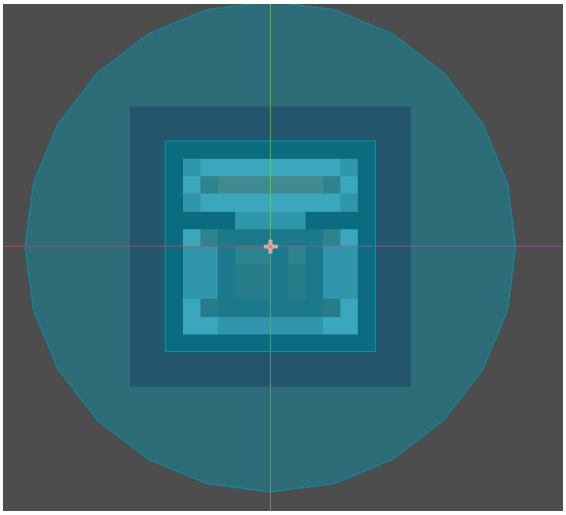


Figure 82: Layout

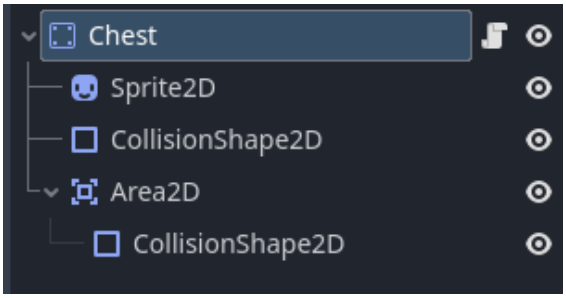


Figure 83: Structure

The layout and structure of the chest are mostly the same as was designed with the detection area being a circle around the chest.

```
extends StaticBody2D

@export var item_pool: Dictionary
@export var item_number: int
@export var key_path: String
@export var locked: bool
var item_list: Array



func _ready() -> void:
    for item in item_pool:
        var add_list = []
        for x in item_pool[item]:
            add_list.append(item)
        item_list.append_array(add_list)

func _input(event):
    if event.is_action_pressed("interact"):
        for i in $Area2D.get_overlapping_bodies():
            if i.name == "Player": # Check the player is within range
                if not(locked) or Inventory.remove_item(key_path,1) is bool: # Check if the player has the key and removes it if the door isnt locked
                    for j in range(item_number):
                        Inventory.add_item(item_list[randi_range(0,len(item_list)-1)],1)
                    print(item_list)
                    queue_free()
```

Figure 84: Script

The chest script is has a couple changes from the design, the addition of a locked flag aswell as making the item list a script variable so that it can be accessed from the input function and constructing the list without the use of pythonic list comprehension. I also switched from directly seeing if the player body overlaps to looping through overlapping bodies again as this wasy easier to implement and works the same.

3.15 Video Testing (Chests and Doors)

I set up my testing environment with two doors, a unlocked door and a locked one aswell as a chest that should give me the necessary key in order to unlock the door, the video can be found here[5].
The testing ended up as expected with the unlocked door able to unlock on the interact action (I key) the locked door would not open before I opened the chest using the interact action and then it would open.

3.16 Projectile Development

3.16.1 Ranged

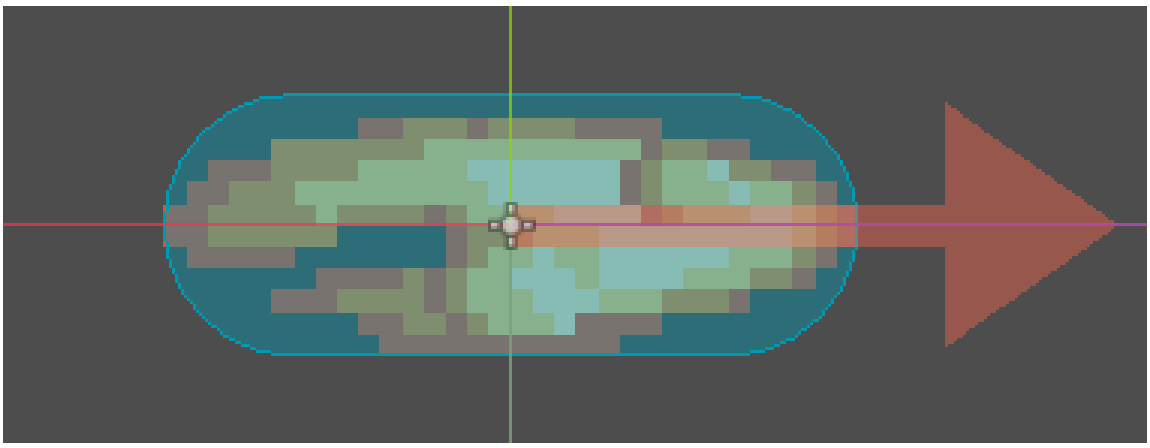


Figure 85: Layout (fire projectile)

This is the layout of the fire ranged projectile the other ranged projectiles followed the same layout with different sprites but the key part is I made the hitbox one way so that the player can't get blocked by the projectile and so projectiles can't get blocked by other projectiles.

```gdscript
func _physics_process(delta: float) -> void:
    velocity.x = speed * cos(rotation)
    velocity.y = speed * sin(rotation)
    if not attacking:
        move_and_slide()
    #Collisions
    for i in range(get_slide_collision_count()):
        if not attacking:
            attacking = true
            var collision = get_slide_collision(i)
            print(collision)
            var collider = collision.get_collider()
            print(collider)
            if collider.is_in_group("player") or collider.is_in_group("enemies"):
                collider.take_damage(damage, damage_type)
            $CollisionShape2D.disabled = true
            $AnimatedSprite2D.play("impact")
            await $AnimatedSprite2D.animation_finished
            queue_free()
```

Figure 86: Scipt

This is the script of the ranged projectiles the damage_type is different depending on wether it is ice, fire or thunder and the damage is set by the script that launches the projectile.

3.16.2 Area

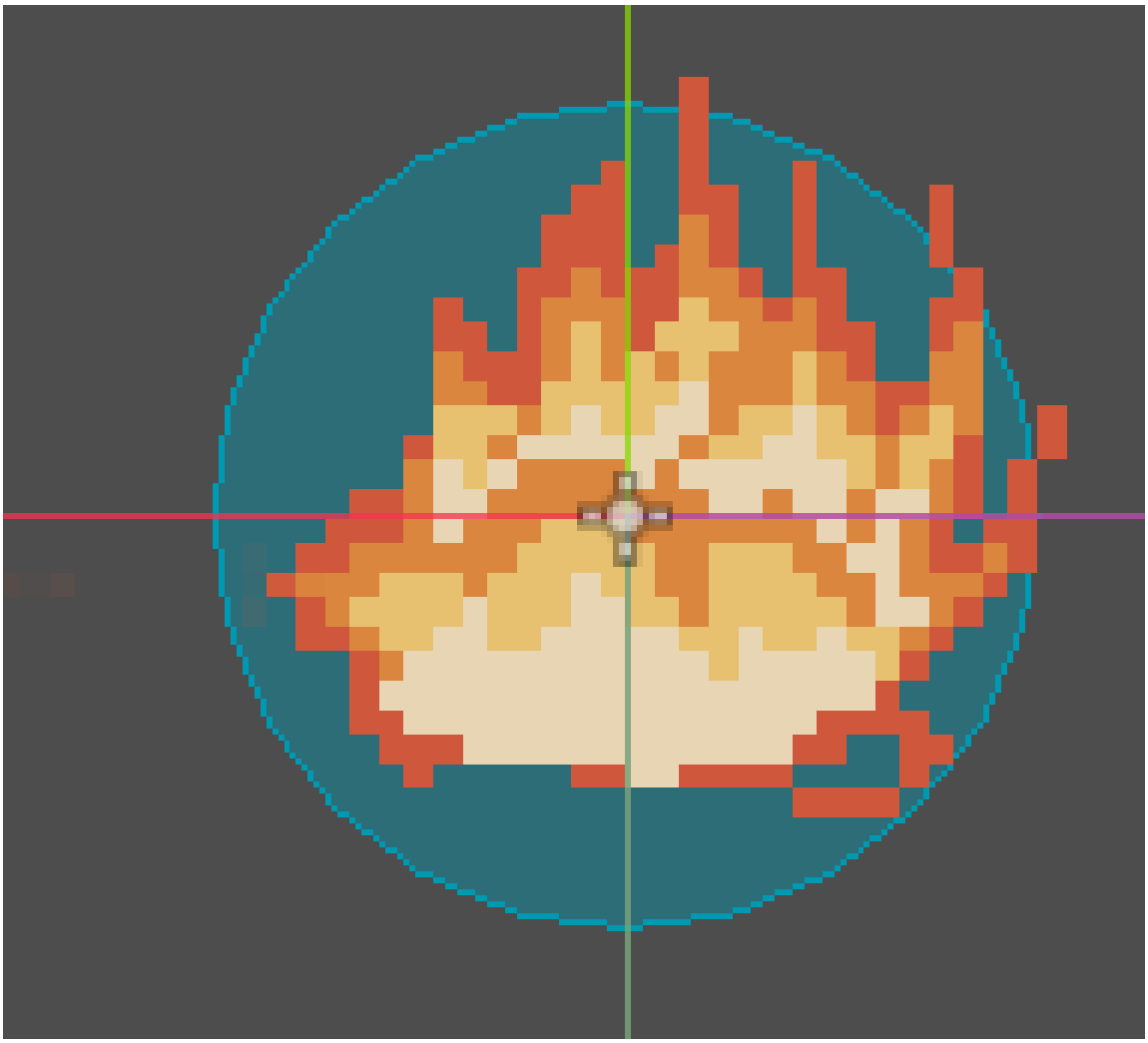


Figure 87: Layout (fire projectile)

This is the layout of the fire area of effect projectiles the other area of effect projectiles followed the same layout with different sprites.

```
func _ready():
    $AnimatedSprite2D.play("default")
    await get_tree().create_timer(time).timeout
    queue_free()

func _on_body_entered(body: Node2D) -> void:
    if body.is_in_group("enemies") or body.is_in_group("player"): # If its an enemy or player damages it
        body.take_damage(damage, damage_type)
```

Figure 88: Script

This is the script of the area of effect projectiles the damage_type is different depending on wether it is ice, fire or thunder and the damage is set by the script that launches the projectile.

3.17 Magic Weapon Development

Due to lack of time for the animations when using magic weapons I have still used the same animation as the melee animation.

I added if statements in the attack function in the player script to handle if the weapon is either a Ranged Magic Weapon or an Area of Effect Magic Weapon aswell as a statement at the start of the attack function to decrease the mana based on the mana cost of the weapon which I added as a new variable in the weapon script.
I also set the projectiles to fire in the same direction as the mouse is after feedback from stakeholder Samuel.

3.17.1 Ranged

```
if weapon.weapon_type == "ranged_magic":
  $AnimatedSprite2D.play(get_animation("melee")) #Change later
  await get_tree().create_timer(0.5).timeout # Account for animation delay
  #Load the projectile scene
  var projectile = load("res://scenes/game/projectiles/%s_projectile.tscn"%weapon.damage_type).instantiate()
  var projectile_direction = (get_viewport().get_mouse_position()-Vector2(1920/2,1080/2)).normalized()
  projectile.rotation_degrees = rad_to_deg(projectile_direction.angle())
  projectile.position = position + 20 * projectile_direction
  projectile.damage = weapon.attack_power
  get_parent().add_child(projectile)
```

Figure 89: Script

This script follows the same principles of the planned script with a couple changes, first I added a part to spawn the projectile away from the player and make it face the same way as the mouse direction, unfortunately I was not able to add support for controllers with this due to lack of time.
Aswell as thus I added a timer at the start to account for the time the animation takes to run so it looks more fluid. And there is a part that finds the relevant projectiled file path from the damage type to load it.

3.17.2 Area Of Effect

```
elif weapon.weapon_type == "area_magic":
    $AnimatedSprite2D.play(get_animation("melee")) #Change later
    await get_tree().create_timer(0.5).timeout # Account for animation delay
    #Load the projectile scene
    var area_direction = (get_viewport().get_mouse_position()-Vector2(1920/2,1080/2)).normalized()
    var area = load("res://scenes/game/projectiles/%s_area.tscn"%weapon.damage_type).instantiate()
    area.position = position + 25 * area_direction
    area.damage = weapon.attack_power
    get_parent().add_child(area)
await $AnimatedSprite2D.animation_finished
```

Figure 90: Script

This script follows the same principles of the planned script with a couple changes, first I added a part to spawn the projectile away from the player (further than the ranged to make sure the player cannot accidentally walk into it) and make it face the same way as the mouse direction, unfortunately I was not able to add support for controllers with this due to lack of time.
Aswell as thus I added a timer at the start to account for the time the animation takes to run so it looks more fluid. And there is a part that finds the relevant projectiled file path from the damage type to load it.

3.18 Enemy Development

I implemented three types of enemies a default and poison slime with melee attacks aswell as a fire slime with ranged attacks.

3.18.1 Layout and Structure

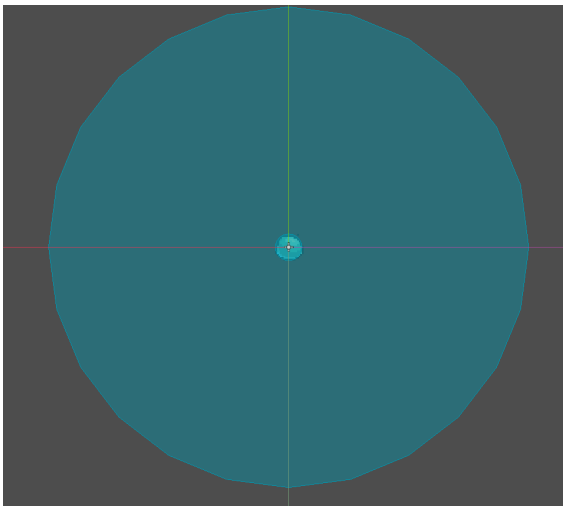


Figure 91: Layout

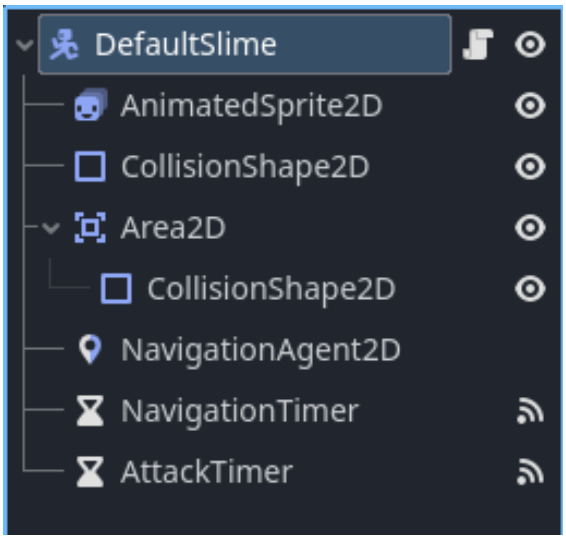


Figure 92: Structure

The layout and structure are as shown above with all the relevant timer nodes, the larger collision shape is the area2D for detecting the player whereas the smaller one handles physics collisons. The NavigationAgent2D handles the navigation and the AnimatedSprite2D handles animation.

3.18.2 General Script

```
extends CharacterBody2D



var direction = Vector2(0,1)
@export var speed:int = 20
@export var health: int = 10
@export var damage: int = 1
@export var damage_type: String = "normal"
@export var weaknesses: Array
var animating: bool
var can_attack = true

func get_animation(animation_type: String):

func _ready():

func _physics_process(delta: float) -> void:

func take_damage(damage, damage_type):

func _on_navigation_timer_timeout() -> void:

func _on_attack_timer_timeout() -> void:
```

Figure 93: General Script and Variables

```
func _ready():
    add_to_group("enemies")
```

Figure 94: _ready():

```
func _on_navigation_timer_timeout() -> void:
    var player = get_tree().get_first_node_in_group("player")
    $NavigationAgent2D.set_target_position(player.global_position)

func _on_attack_timer_timeout() -> void:
    can_attack = true
```

Figure 95: timer_timeouts

This shows the general script structure and variables as was described with the animating and can_attack flags declared as script variables so they can be used throughout the functions.
The ready function adds it to the enemies group so that the player's attacks will damage it and the navigation timer is set to autorun and when it times out the pathfinding towards the player (using NavigationAgent2D) is updated. When the attack timer runs out then the enemy can attack again.

3.18.3 _physics_process():

```gdscript
func _physics_process(delta: float) -> void:
	var move = false
	for body in $Area2D.get_overlapping_bodies():
		if body.name == "Player":
			move = true
	if not animating:
		if move:
			direction = to_local($NavigationAgent2D.get_next_path_position()).normalized()
			velocity = direction * speed
			$AnimatedSprite2D.play(get_animation("walk"))
			move_and_slide()
		else:
			$AnimatedSprite2D.play(get_animation("idle"))
	else:
		move_and_slide()

	if can_attack:
		for i in range(get_slide_collision_count()):
			var collision = get_slide_collision(i)
			var collider = collision.get_collider()
			if collider.is_in_group("player"):
				collider.take_damage(damage, damage_type)
				can_attack=false
				$AttackTimer.start()
```

Figure 96: melee _physics_process(delta)

This script is the same as the designed script except with the move_and_slide function moved so that the enemy will not move if the idle animation is playing.

```gdscript
func _physics_process(delta: float) -> void:
	var player_detected = false
	for body in $Area2D.get_overlapping_bodies():
		if body.name == "Player":
			player_detected = true
	if not animating:
		$AnimatedSprite2D.play(get_animation("idle"))
		if player_detected and can_attack:
			direction = to_local($NavigationAgent2D.get_next_path_position()).normalized()
			var projectile = load("res://scenes/game/projectiles/fire_projectile.tscn").instantiate()
			projectile.rotation_degrees = rad_to_deg(direction.angle())
			projectile.position = position + 20*direction
			projectile.damage = damage
			get_parent().add_child(projectile)
			can_attack=false
			$AttackTimer.start()
```

Figure 97: ranged _physics_process(delta)

This is the same as the designed script using godot syntax and with the fire projectile scene added in as this is fire ranged slime.

3.18.4 take_damage(damage, damage_type):

```
func take_damage(damage, damage_type):
    var player = get_tree().get_first_node_in_group("player")
    animating = true
    if damage_type in weaknesses:
        health -= 2*damage
    else:
        health -= damage
    velocity = - 25 * to_local(player.global_position).normalized()
    if health == 0:
        queue_free()
    else:
        $AnimatedSprite2D.play(get_animation("hurt"))
    await $AnimatedSprite2D.animation_finished
    animating = false
```

Figure 98: take_damage()

This is the same as the designed script but changing the player global position to a localised position relative to the enemy to properly get the direction.

3.18.5 Animation

For animation I used three different spritesheets for the different types of slimes.

```
func get_animation(animation_type: String):
    if abs(direction.x) > abs(direction.y):
        if direction.x > 0:
            return animation_type + "_r"
        else:
            return animation_type + "_l"
    else:
        if direction.y > 0:
            return animation_type + "_d"
        else:
            return animation_type + "_u"
```

Figure 99: get_animation()

This is exactly the same as the designed script.

3.19 Video Testing (Player Attacks and Enemies)

I created a test scene with a couple enemies aswell as the player and made it so the player has automatically got an item equipped to test the player attacks and enemies in the evidence here[4].
The Testing was a success, the player directional attacks, different weapons all functioned as intended and the enemies attacked and tracked the player firing projectiles and moving as expected.

3.20 UI Development

3.20.1 Inventory UI

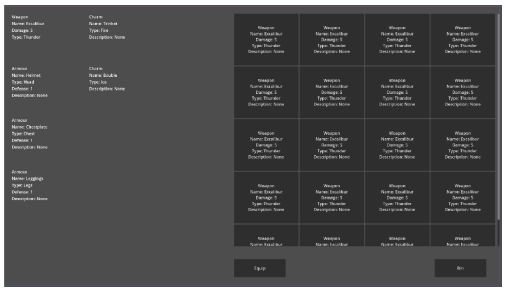


Figure 100: Layout

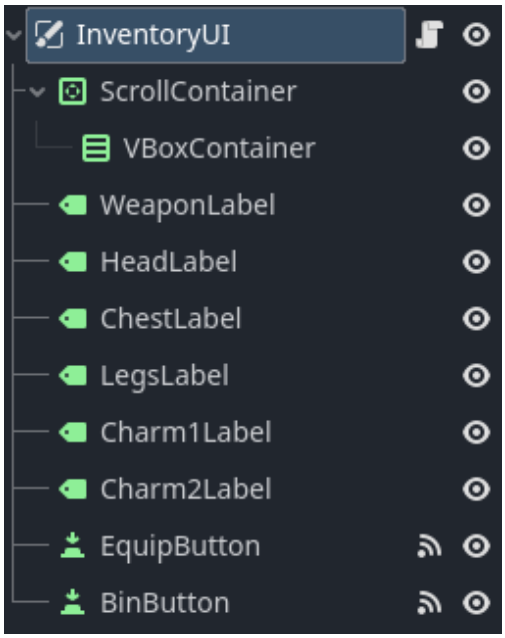


Figure 101: Structure

The Layout and Structure are as was designed.

```
func _ready() -> void:
    process_mode = Node.PROCESS_MODE_ALWAYS
    refresh()
```

Figure 102: _ready

I added the ready function as when we pause the tree the buttons wont work unless we set the process_mode to PROCESS_MODE_ALWAYS. It also calls the redresh function to setup the UI.

```gdscript
func refresh():
    var inventory = Database.get_stored_items()
    var item_count = 0
    for child in $ScrollContainer/VBoxContainer.get_children(): #Get rid of existing HBoxContainers
        child.queue_free()
    for item in inventory: # Creates a button for each item in inventory with the display string
        var button = Button.new()
        if item_count % 4 == 0:
            hbox = HBoxContainer.new()
            $ScrollContainer/VBoxContainer.add_child(hbox)
            print(hbox)
        button.text = load(item["item_id"]).display_string() + "\nAmount: %d" % item["amount"]
        button.connect("pressed",_select.bind(item["item_id"]))
        button.custom_minimum_size = Vector2(250,200)
        hbox.add_child(button)
        item_count += 1

    # Loads the equipped items display strings and displays them
    if Database.get_slot_value("weapon"):
        $WeaponLabel.text = load(Database.get_slot_value("weapon")).display_string()
    if Database.get_slot_value("head"):
        $HeadLabel.text = load(Database.get_slot_value("head")).display_string()
    if Database.get_slot_value("chest"):
        $ChestLabel.text = load(Database.get_slot_value("chest")).display_string()
    if Database.get_slot_value("legs"):
        $LegsLabel.text = load(Database.get_slot_value("legs")).display_string()
    if Database.get_slot_value("charm_1"):
        $Charm1Label.text = load(Database.get_slot_value("charm_1")).display_string()
    if Database.get_slot_value("charm_2"):
        $Charm2Label.text = load(Database.get_slot_value("charm_2")).display_string()
```

Figure 103: refresh()

The refresh function is as was designed with the added custom minimum size set so that all buttons are the same size for neatness and readability.

```
func _select(item_id):
    selected = item_id


func _on_equip_button_pressed() -> void:
    if selected != "":
        Inventory.equip_item(selected)
        refresh()


func _on_bin_button_pressed() -> void:
    if selected != "":
        Database.remove_stored_item(selected)
```

Figure 104: select() and _on_buttons_pressed()

These functions are as was designed using pre-existing functions to perform their tasks.

3.20.2 Game UI

Figure 105: Layout

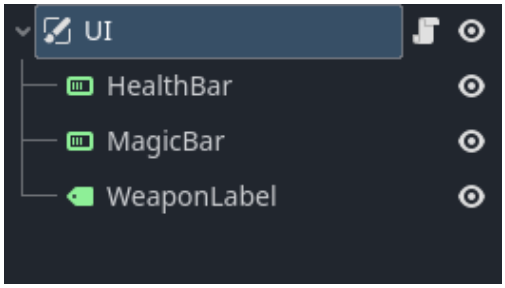


Figure 106: Structure

This layout and structure is as was described with the custom texture colours for the progressbars being set in the editor to colours similar to ones that a majority of other games use so that it is intuitive. The Health Bar is on top of the Magic Bar.

```gdscript
var player

func _ready():
    var player = get_tree().get_first_node_in_group("player")
    $HealthBar.max_value = player.health
    $MagicBar.max_value = player.mana
    update_ui()


func _process(delta: float) -> void:
    var player = get_tree().get_first_node_in_group("player")
    $HealthBar.value = player.health
    $MagicBar.value = player.mana


func update_ui():
    if Database.get_slot_value("weapon"):
        var weapon = load(Database.get_slot_value("weapon"))
        $WeaponLabel.text = weapon.display_string()
```

Figure 107: Script

The GameUI script is as was designed.

3.20.3 Enemy UI addition

I used the same concept of a healthbar to add a ProgressBar to the enemies which only shows up when they take damage and displays their health until their take damage animation has finished playing.

Figure 108: Layout

```gdscript
func take_damage(damage, damage_type):
    var player = get_tree().get_first_node_in_group("player")
    animating = true
    if damage_type in weaknesses: #Takes more damage from weaknesses
        health -= 2*damage
    else:
        health -= damage
    velocity = - 25 * to_local(player.global_position).normalized() #takes knockback by changing velocity
    $HealthBar.value=health
    $HealthBar.visible = true
    if health <= 0: #dies if health is low
        queue_free()
    else:
        $AnimatedSprite2D.play(get_animation("hurt"))
    await $AnimatedSprite2D.animation_finished
    animating = false
    velocity = Vector2(0,0) #Removes knockback velocity
    $HealthBar.visible = false
```

Figure 109: Script

This shows the new layout with the healthbar shown and the changes made to the take_damage() function to update and show the healthbar till the animation is finished.

3.20.4 Help Menu

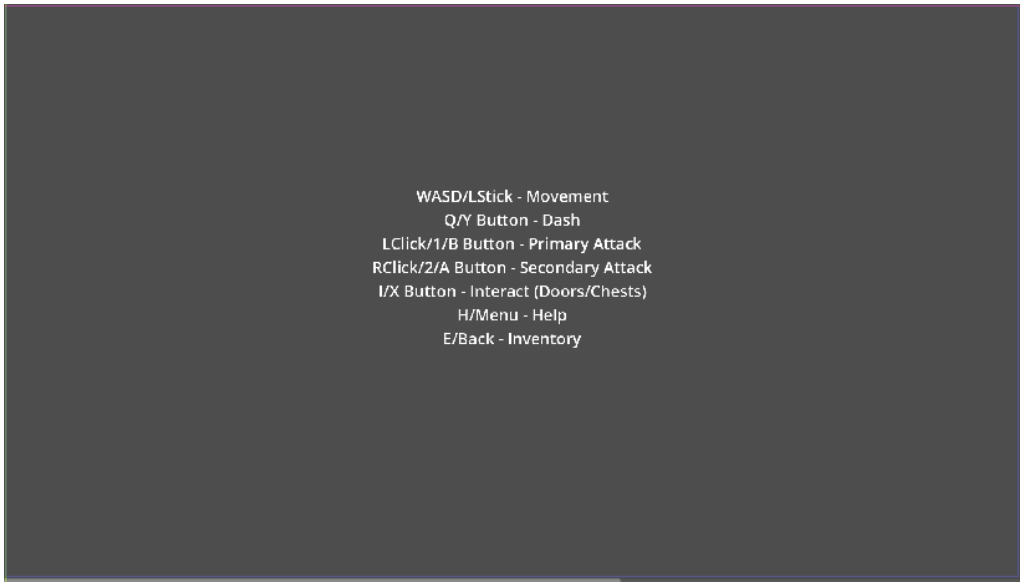


Figure 110: Layout

This is the layout explaining all the controls.

```
elif event.is_action_pressed("help"):
    var help_menu = load("res://scenes/menu/help_menu.tscn").instantiate()
    add_child(help_menu)
    get_tree().paused = true
    await get_tree().create_timer(0.2).timeout
    while not Input.is_action_just_pressed("help"):
        await get_tree().process_frame
    help_menu.queue_free()
    get_tree().paused = false
```

Figure 111: Script

This is in the input script if the help button is pressed it displays the help menu, pauses the tree and waits till the help button is pressed again before getting rid of the help menu and unpausing the tree.

3.21 Video Testing (UI)

For the UI testing I set up an environment where the player can gain items to test all the functions of the Inventory UI I made it so I was able to equip magic weapons to use up magic points and so I could take damage and deal damage to an enemy to view both HealthBars.
The UI testing can be found here[6] and overall went well with all the individual parts of the UI functioning as expected.

3.22 Tutorial Development

For the tutorial I tried to introduce all the mechanics I had made in the game to the player in a series of small challenges, guiding them along with labels introducing controls etc.

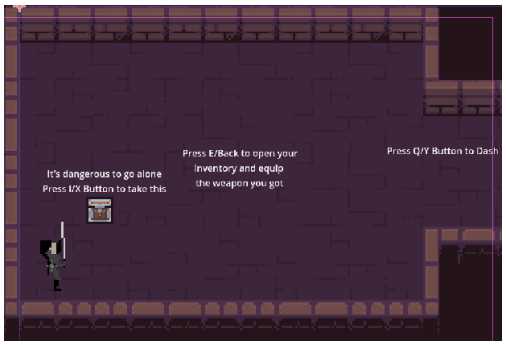


Figure 112: Beggining

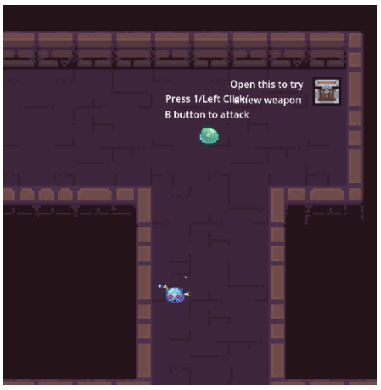


Figure 114: First Enemies and Chest

Figure 113: Main Room

Figure 115: Level End and Reward

The Player progresses getting their first weapon and equipping it.
Fighting their first enemy and getting their first magic ranged weapon (thunder).
Fighting different enemy types.
Unlocking the final door and chest using keys gained from the chest in the main room and then trying out a magic Area of Effect weapon (Fire).

3.23 Stakeholder Feedback 2

In this stakeholder feedback I showed stakeholders my improvements based on their feedback from last time aswell as the tutorial and features implemented there. The video can be found here[11]

3.23.1 Dropped Features

Several Features had to be dropped due to time constraints.
I removed the secondary attack from the help menu due to the fact it wasnt necessary and couldnt be implemented within time constraints aswell as the fact it confused stakeholders.
I dropped support for controller controls due to time constraints and it taking to long to fully implement it for projectile attacks aswell as the fact that it being in all the labels confused stakeholders.

3.23.2 Tutorial

In the tutorial I forgot to place tiles of the background colour all around so I decided to make the default background colour the same as the tilesets, this will help when I do procedural generation as I wont have overlapping background tiles.
I also lowered the health of the enemies in the tutorial level as that was commented on and the enemies need to be easier to defeat so that the new players dont get overwhelmed.
I increased the range of enemy detection as the player was able to defeat the enemies with ranged weapons before the enemies could see them.

3.23.3 Weapons

I decreased the size of the Area of Effect's hitboxes so that the player can't take damage from them by accident as easily as this happened to stakeholders in the test.
I added new magic weapon types (ice projectile and thunder area of effect) to the chest at the end of the tutorial so that the player gets a range of options to choose from as in the tutorial the stakeholders only got to try out 3 weapon types.

3.23.4 Menus

Upon Feedback I changed the Save Menu Layout by adding a name feild to the saves in the database and then allowing the user to set that name when creating a new save for more clear labelling of saves and added more labels to the menu. This involved slightly modifying the SQL queries create_table_save_data, add_new_save_data and get_user_save_data for use in the save menu to add in the name field.

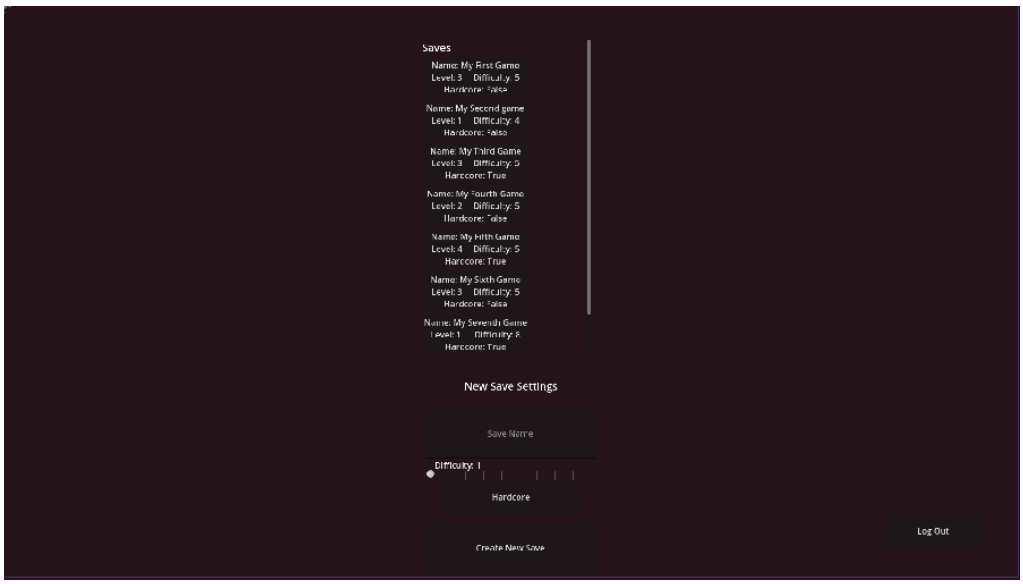


Figure 116: Save Menu Layout

I also added a confirm password box to the reset password menu which needs to be the same as the password box to properly reset the password to prevent users from mistyping the password.

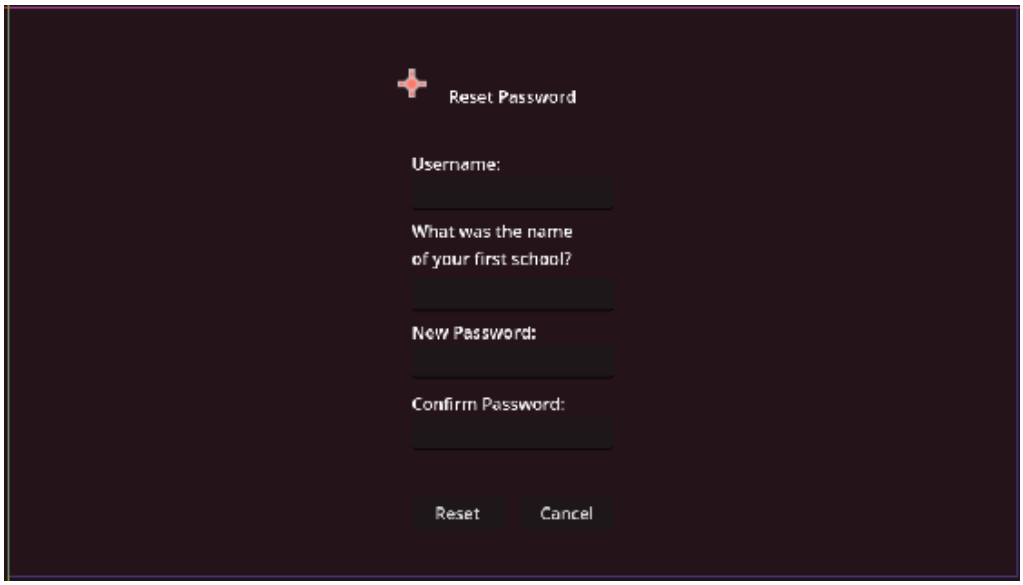


Figure 117: Reset Password Layout

3.23.5 Inventory

One thing I noticed during the stakeholder feedback was that when Daniel selected an item and equipped it twice the item would first dissappear from the inventory then go back into the inventory the second time, this is because of a lack of checking if the item is in the inventory before equipping it which I fixed with a simple SQL query in the equip_item() function returning false if none of the item is in the inventory.

```
# Checks if the item is Equipable and in the inventory
if not(item is Equipable) and item_amount(item_id) >= 1:
    return false
```

Figure 118: Script Change

I also added a logout button in the inventory which takes you to the login form as was requested and removed the placeholder text for the equipped item slots as that isnt needed.

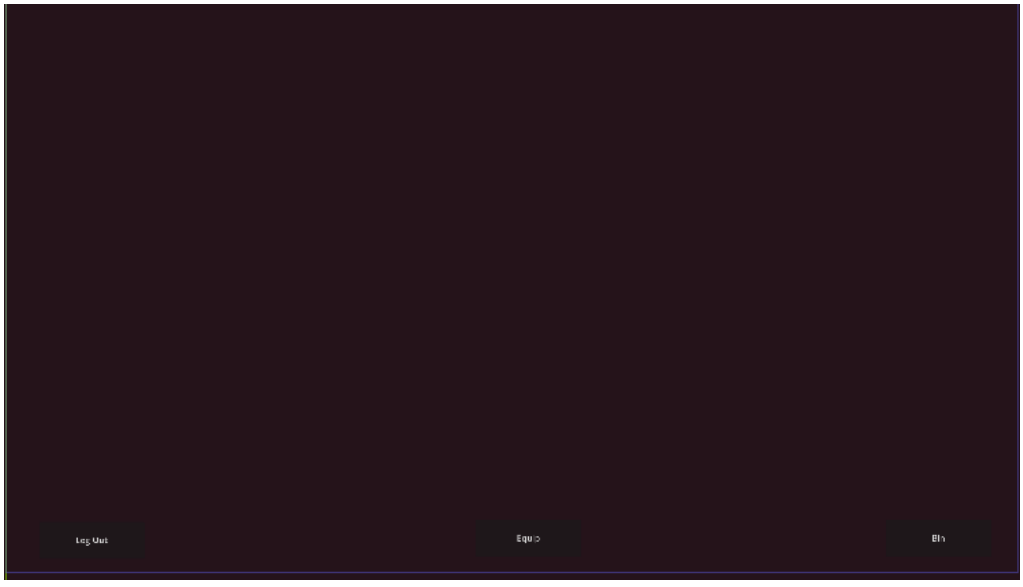


Figure 119: Layout Change

3.24 Procedural Generation Development

3.24.1 Graph

DungeonGraphNode:

```
extends Node


class_name DungeonGraphNode


var north: DungeonGraphNode = null
var south: DungeonGraphNode = null
var east: DungeonGraphNode = null
var west: DungeonGraphNode = null
var room_type: String
```

Figure 120: Script

This script is as described setting default values and outlining class attributes.
DungeonGraph:

```
class_name DungeonGraph

var root: DungeonGraphNode
var nodes: Array
var rooms: Dictionary = {'chest':["res://scenes/game/worlds/rooms/monster/room.tscn"],
'end':["res://scenes/game/worlds/rooms/end/end.tscn"],
'start':["res://scenes/game/worlds/rooms/start/start.tscn"],
'monster':["res://scenes/game/worlds/rooms/monster/m_room_1.tscn","res://scenes/game/worlds/rooms/monster/m_room_2.tscn","res://scenes/game/worlds/rooms/mons
'corridor':["res://scenes/game/worlds/rooms/corridors/corridor_1.tscn","res://scenes/game/worlds/rooms/corridors/corridor_2.tscn","res://scenes/game/worlds/r
'corridor_along':["res://scenes/game/worlds/rooms/corridors/corridor_1.tscn","res://scenes/game/Worlds/rooms/corridors/corridor_7.tscn"],
'corridor_up':["res://scenes/game/worlds/rooms/corridors/corridor_2.tscn","res://scenes/game/worlds/rooms/corridors/corridor_8.tscn"],
'corridor_corner':["res://scenes/game/worlds/rooms/corridors/corridor_3.tscn","res://scenes/game/worlds/rooms/corridors/corridor_4.tscn","res://scenes/game/w
var opposite_direction = {'north':'south', 'south':'north', 'east':'west', 'west':'east'}
```

Figure 121: Variables

This is the variables and class_name of the DungeonGraph you can see the dictionary with the lists of paths to rooms for each different room type.

```
func _init():
    root = DungeonGraphNode.new()
    root.room_type = "start"
    nodes.append(root)

func add_node(onto_index, direction, room_type):
    var onto = nodes[onto_index]
    if onto[direction]:
        return false
    var new_node = DungeonGraphNode.new()
    new_node.room_type = room_type
    onto[direction] = new_node
    new_node[direction] = onto
    nodes.append(new_node)
    return true
```

Figure 122: _init() and add_node()

This is the _init and add_node functions implemented as was designed.

```
func gen_room(node,previous_direction = null, previous = null):
    var room = load(rooms[node.room_type].pick_random()).instantiate()
    add_child(room)
    if previous_direction:
        room.set_pos(previous_direction,previous.get_pos(opposite_direction[previous_direction]))
    else:
        room.position = Vector2(0,0)
    for i in ['north', 'south', 'east', 'west']:
        if node[i] == null:
            room.cap(i)
    return room

func gen_dungeon(node=root, previous_direction = null, previous = null, generated = []):
    var room = await gen_room(node,previous_direction,previous)
    generated.append(node)
    for i in ['north', 'south', 'east', 'west']:
        if node[i] and node[i] not in generated:
            generated = await gen_dungeon(node[i], opposite_direction[i], room, generated)
    return generated
```

Figure 123: gen_room() and gen_dungeon()

This is the gen_room() and gen_dungeon() functions as was described the only extra thing is I had to add the DungeonGraph to a Node2D in order for the script to be able to add the room as a child.

3.24.2 Room

```
func get_pos(direction):
    match direction:
        'north':
            return $North.global_position
        'south':
            return $South.global_position
        'east':
            return $East.global_position
        'west':
            return $West.global_position

func set_pos(direction, global_pos):
    match direction:
        'north':
            global_position += global_pos - $North.global_position
        'south':
            global_position += global_pos - $South.global_position
        'east':
            global_position += global_pos - $East.global_position
        'west':
            global_position += global_pos - $West.global_position
```

Figure 124: get_pos() and set_pos()

This is the get_pos() and set_pos() functions implemented as designed with godot's syntax for accessing child nodes.

```
func cap(direction):
    match direction:
        'north':
            var north_cap = load("res://scenes/game/worlds/rooms/north_cap.tscn").instantiate()
            $North.add_child(north_cap)
        'south':
            var south_cap = load("res://scenes/game/worlds/rooms/south_cap.tscn").instantiate()
            $South.add_child(south_cap)
        'east':
            var east_cap = load("res://scenes/game/worlds/rooms/east_cap.tscn").instantiate()
            $East.add_child(east_cap)
        'west':
            var west_cap = load("res://scenes/game/worlds/rooms/west_cap.tscn").instantiate()
            $West.add_child(west_cap)
```

Figure 125: cap()

This is the cap() function as was designed using godot's syntax for referencing child nodes.

```
func _ready():
    if $Slimes:
        for slime in $Slimes.get_children():
            slime.health = floor(2*Global.current_level*log(3*Global.difficulty))
            slime.damage = floor(Global.current_level*log(3*Global.difficulty))
    if $Labels:
        for label in $Labels.get_children():
            label.text = label.text.replace("0", str(Global.current_level))
```

Figure 126: _ready()

This is the rooms _ready() function as was designed with and added bit to change the text in all labels replacing the number 0 with the current level, this helps for the start and end levels where I have added text sayig about the levels.

Room Types:

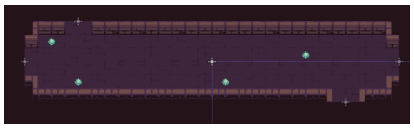


Figure 127: corridor_along

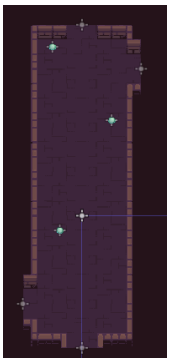


Figure 128: corridor_up

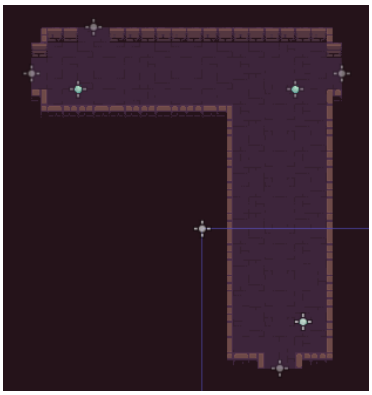


Figure 129: corridor_corner

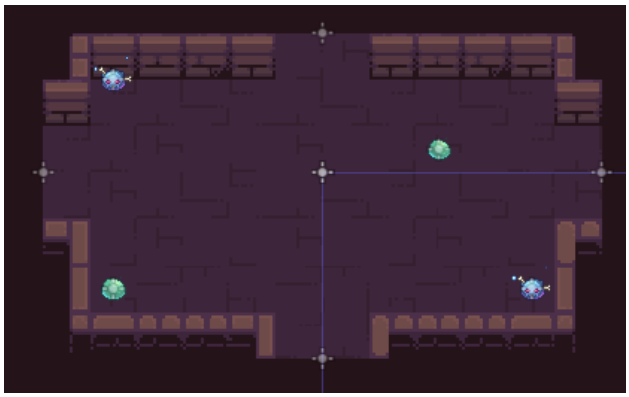


Figure 130: monster_1

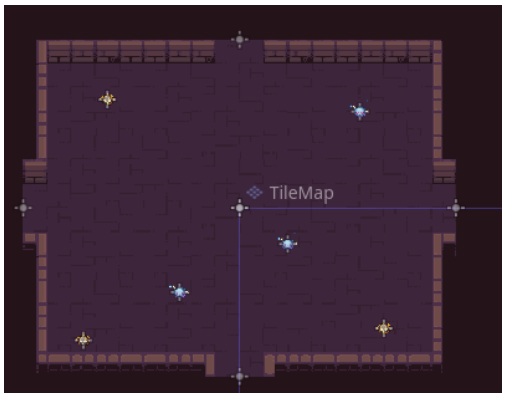


Figure 131: monster_2

For the room types I decided on monster, corridor, corridor_along, corridor_up and corridor_corner aswell as special room types for the start and end. With the corridors I decided to have a convention of always having the north entrance on the west of the north side, the south entrance on the east of the south side, the east entrance on the north of the east side and the west entrance on the south of the west side. This pattern helped solve the problem of generating levels that dont have overlapping rooms which was a concern. Ontop of that I had to find level structure's such that it wouldnt overlap when generating ever.

3.24.3 Testing

For Testing I used a room_test scene in which I tested various graph layouts for generation patterns aswell as testing loading the player sprite in after.

```
func _ready():
    await dungeon_3()
    #Used for shrinking dungeon to view generation
    #self.scale = Vector2(0.5,0.5)
    #self.position = Vector2(1920/4,1080/4)
    #Used for placing player in
    var player = load("res://scenes/game/player/player.tscn").instantiate()
    player.position = Vector2(1920/2,1080/2)
    add_child(player)
```

Figure 132: _ready

This function generated the dungeon and either instantiated the player or shrunk it to check the generation.

```gdscript
func dungeon_1():
    var graph = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate()
    add_child(graph)
    graph.add_node(0, 'north', 'corridor_along')
    graph.add_node(0, 'east', 'corridor_corner')
    graph.add_node(1, 'west', 'monster')
    graph.add_node(2, 'north', 'corridor_up')
    graph.add_node(4, 'north', 'monster')
    graph.add_node(2, 'east', 'monster')
    graph.add_node(0, 'south', 'corridor_corner')
    graph.add_node(7, 'south', 'monster')
    graph.add_node(6, 'south', 'corridor')
    graph.add_node(9, 'south', 'monster')
    await graph.gen_dungeon()
```

Figure 133: dungeon_1()

```gdscript
func dungeon_2():
    var graph  = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate()
    add_child(graph)
    graph.add_node(0, 'north', 'corridor_corner')
    graph.add_node(1, 'east', 'corridor_corner')
    graph.add_node(2, 'east', 'corridor_corner')
    graph.add_node(3, 'east', 'corridor_corner')
    graph.add_node(1, 'north', 'monster')
    graph.add_node(2, 'north', 'monster')
    graph.add_node(2, 'south', 'monster')
    graph.add_node(3, 'north', 'monster')
    graph.add_node(3, 'south', 'monster')
    graph.add_node(4, 'north', 'monster')
    graph.add_node(4, 'south', 'monster')
    graph.add_node(11, 'east', 'corridor_corner')
    graph.add_node(12, 'south', 'monster')
    graph.add_node(12, 'east', 'corridor_up')
    graph.add_node(14, 'north', 'boss')
    await graph.gen_dungeon()
```

Figure 134: dungeon_2()

```gdscript
func dungeon_3():
    var graph  = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate()
    add_child(graph)
    graph.add_node(0, 'north', 'corridor_corner')
    graph.add_node(0, 'east', 'corridor_corner')
    graph.add_node(0, 'south', 'corridor_corner')
    graph.add_node(0, 'west', 'corridor_corner')
    graph.add_node(1, 'north', 'monster')
    graph.add_node(2, 'east', 'monster')
    graph.add_node(3, 'south', 'monster')
    graph.add_node(4, 'west', 'monster')

    await graph.gen_dungeon()

```

Figure 135: dungeon_3()

This is the dungeon generation functions.
There is video evidence of running the test functions to generate here[7] and testing player loading in here[8].

3.24.4 Level 1

For Level 1 I combined a bunch of the different techniques I found with the testing into a level and added an end to the level.

```
func _ready():
    #Set current level
    Global.current_level = 1
    #Level graph setup
    var graph = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate()
    add_child(graph)
    graph.add_node(0, 'east', 'corridor_along')
    graph.add_node(1, 'south', 'monster')
    graph.add_node(2, 'east', 'corridor_corner')
    graph.add_node(3, 'north', 'corridor_up')
    graph.add_node(4, 'north', 'monster')
    graph.add_node(3, 'east', 'monster')
    graph.add_node(2, 'south', 'corridor_corner')
    graph.add_node(7, 'south', 'monster')
    graph.add_node(6, 'south', 'corridor')
    graph.add_node(9, 'south', 'monster')
    graph.add_node(10, "east", 'corridor_along')
    graph.add_node(11, "east", "monster")
    graph.add_node(12, 'east', 'corridor_corner')
    graph.add_node(13, 'east', 'monster')
    graph.add_node(14, 'north', 'corridor_corner')
    graph.add_node(14, 'east', 'corridor_corner')
    graph.add_node(14, 'south', 'corridor_corner')
    graph.add_node(15, 'north', 'monster')
    graph.add_node(16, 'east', 'monster')
    graph.add_node(17, 'south', 'monster')
    graph.add_node(19, 'east', 'corridor_along')
    graph.add_node(21, "east", "monster")
    graph.add_node(22, 'north', 'corridor_corner')
    graph.add_node(23, 'east', 'corridor_corner')
    graph.add_node(24, 'east', 'corridor_corner')
    graph.add_node(25, 'east', 'corridor_corner')
    graph.add_node(23, 'north', 'monster')
    graph.add_node(24, 'north', 'monster')
    graph.add_node(24, 'south', 'monster')
    graph.add_node(25, 'north', 'monster')
    graph.add_node(25, 'south', 'monster')
    graph.add_node(26, 'north', 'monster')
    graph.add_node(26, 'south', 'monster')
    graph.add_node(33, 'east', 'corridor_corner')
    graph.add_node(34, 'south', 'monster')
    graph.add_node(34, 'east', 'corridor_up')
    graph.add_node(36, 'north', 'end')

    await graph.gen_dungeon()
    #Used for shrinking dungeon to view generation
    #self.scale = Vector2(0.25,0.25)
    #self.position = Vector2(0,1080/8)
    #Used for loading player in
    var player = load("res://scenes/game/player/player.tscn").instantiate()
    add_child(player)
```

Figure 136: Script

There is video evidence of generation in Level 1 and player loading in here[9]

4 Evaluation

4.1 Success Criteria

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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.	1.1 Fully Met 1.2 Fully Met 1.3 Fully Met 1.4 Fully Met 1.5 Fully Met 1.6 Fully Met

This criteria is fully met and can be seen through the video evidence here[1]. Both the WASD controls and the controller controls work fully as implemented through the players physics process although the controller controls have been removed from help menu and tutorial tips because not all the controller controls where able to be implemented. The correct keys can be seen to correspond to a movement in the corresponding direction aswell as the player moving diagonally when two keys next to each other are pressed. The Q key corresponding to dash can be seen in stakeholder feedback 2 here[11] stakeholder daniel presses the Q key as instructed bty the tutorial guidance and it correctly triggers the dash.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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 Make sure all attacks go in the direction the player is facing	2.1 Fully Met 2.2 Not Met 2.3 Fully Met 2.4 Not Met 2.5 Fully Met 2.6 Fully Met 2.7 Fully Met 2.8 Fully Met 2.9 Not Met

Successfull:
The attack system and different weapons aswell as enemies and player taking damage can be seen here[4].
2.1 is implemented with the _input() function in the player script handling inputs. This can be seen through in the testing through the succesfull player attacks upon pressing mouse-1 and the 1 key and the X button.
2.3 is implemented through the weapon resource class as instances and the implementation of the sword swing hurtbox detecting collisions enemies hitboxes. This can be seen when the player has the melee sword equipped in the test as the sword correctly damages the enemies within range when the mouse-1/1 key/X button is pressed, successfully killing them.
2.5 is implemented through the weapon resource class as instances and the implementation of the fire, ice and thunder projectiles movement and detecting collisions with enemies or the player. These projectiles are fired in the direction of the player mouse. This can be seen when the magic ranged weapon is equipped in the video testing as the magic weapon successfully shoots the projectile in the correct direction upon the mouse-1/1key/X button being pressed, the projectile then goes on to explode when it hits an enemy/the wall and damages any enemy it hits.
2.6 is implemented through the weapon resource class as instances and the implementation of the fire, ice and thunder area of effects including them lasting a specific time and detecting when enemies or the player are within the harm radius. These areas are spawned in the direction of the player mouse. This can be seen when the magic area of effect weapon is equipped within the video testing, the are correctly spawns infront of the player and damages any enemy that is inside of it, lasting for a set time.
2.7 is met through the usage of a CollisionShape2D node in the player structure and the take_damage function that can get triggered by projectiles or enemies when they collide with the player. This can be seen in the video

testing when the melee enemies or projectiles come into contact with the player hitbox the player successfully registers the hit and takes damage.

Partially/Unsuccessfull:

2.2 is not met due to lack of time to add extra attacks. If I were to have more time I would implement 2.2 as a secondary charged attack using a ProgressBar to measure the charge of the attack and then when released for melee attacks and magic area of effect attacks the attack would do damage proportional to the charge of the bar for magic ranges it would shoot out multi directional projectiles and for magic area of effect attacks the area would be scaled up. I would add a charging animation to be activated when charging the attack.

2.4 is not met due to lack of sprites and animations aswell as lack of extra time.

If I were to have the sprites I would make the primary attack a rapid fire that rapidly charges the bow firing arrow projectiles at the enemies in the direction of the player mouse. For the secondary attack I would have a slower charging animation to release an arrow dealing greater damage.

2.9 Is not met as with projectiles I make them fire the direction of the mouse from the player (can be seen for the magic weapons in the video testing) as this allows better aiming accuracy however with controller controls this does not work as there is no way to move the mouse so I ended up not implementing it due to overcomplicatedness and lack of time.

If I had the extra time I would try to make it so you can use the right control stick in order to aim the projectiles directions either by moving the mouse or storing the last direction of it and then using that direction the same way I used the mouse direction.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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 and Monster 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	3.1 Fully Met 3.2 Fully Met 3.3 Fully Met 3.4 Partially Met 3.5 Fully Met 3.6 Fully Met

Successfull:

The Dungeon Environment and different rooms can be seen through the procedural generation tests here[7], here[8] and here[9]. The Doors and Chests can be seen through their tests here[13]. The tilemap can be seen through the tests here[1]

3.1 and 3.2 are implemented through the use of a dungeon tileset that I have used to make a tilemap with physics layers on the walls so that you cannot walk through them and floor tiles for the floor. This can be seen in the video testing as even when the player can be seen to be walking in the wall and I held the key down the player could not walk through the wall even in the corner.

3.3 is implemented through the chest scene and script, with a dictionary to represent the item pool of the chest along with the ratios of chances of getting each item being used to construct an array from which a random item is chosed for each item to be given to the player when they player is within range of the chest and presses interact (I Key, X Button) from the chest and then the items are added to the player inventory. This can be seen in the video testing as before the player opens the chest they have not recieved the key item necessary to open the door whereas after they open the chest they recieve the key item and are able to open the door it can also be seen in the UI test here[6] where the player only has one item in their inventory before they open the chest giving them a selection of other items.

3.5 is implemented through the door scene and script, the door has an area2d node to detect when the player is close enough to open it, then when the player presses interact (I Key, X Button) the door will check if it is locked and the player inventory has the correct key or it is unlocked in both cases it will dissappear or optionally can change the scene. If the player does not have the key and the door is locked then the door will not react. This can be seen in the video testing as their is both locked and unlocked doors, the unlocked door opens first on the interact key and the locked door does not open until the condition of having the key is fulfilled by getting it from the chest.

3.6 is implemented both in the tutorial through the handmade environment and also through the procedural generation where individual rooms and corridors are peiced together with the graph script in order to make the level environments. This can be seen through the procedural generation tests where the dungeon environment gets generated from the different rooms and corridors and displayed to see.

Partially/Unsuccessfull:

3.4 is only partially successful as I have implemented seperate monster rooms and corridors as can be seen in the procedural generation tests. I have however not implemented chest or boss rooms due to lack of time to

create these rooms.
For the chest rooms I would make it so some chest rooms would contain keys in the chests that are needed to unlock the chests in other chest rooms throughout the dungeon I would add the chest scene to existing monster rooms to create them and have default item pools aswell as special chest rooms with more rare item pools. For the boss rooms I would add rooms designed around the specific implemented bosses with areas to hide from the bosses attacks as it chases you aswell as placing the boss scenes in different places within the rooms.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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	4.1 Fully Met 4.2 Fully Met 4.3 Fully Met 4.4 Fully Met 4.5 Fully Met 4.6 Fully Met 4.7 Fully Met 4.8 Not Met

Successfull:
The enemies and player fighting them can be seen here[4]. 4.1 is implemented through the use of an AnimatedSprite2D node on the different enemy scenes which displays the sprite aswell as different animations relevant to the action the enemy is doing. The animations are triggered in the enemy script. This can be seen throughout the video testing as the visual element of the enemies aswell as their animations.
4.2 is implemented through the use of Godot's NavigationAgent2D which uses a Mesh A* algorithm on a Navigation Layer I added to all the floor tiles in the TileMap to help the enemies pathfind towards the player, giving the next corner of the journey that I have used to move the moving enemies towards the player in the script, I have also used the same direction for the projectile non-moving enemies so that they can detect which way to fire their projectiles. The delay timer godot offers allows for less lock on pathfinding only updating every few seconds to make it easier to avoid. This can be seen in the script as the enemies pathfind towards the player but if the player is moving quickly they do not update their movement direction straight away. The same applies to the pathfinding used for the projectile enemies projectile directions.
4.3 is implemented through the use of an Area2D node and a circular CollisionShape2D this acts as the detection area for detecting the player and their is conditions in the script that only lets the melee enemies move or the projectile enemies attack when the player is within the range of their Area2D. This can be seen through the test aswell as in the UI test here[6] as the when the player is out of the enemies range at the start of the test the enemies does not attempt to pathfind or shoot towards the player.
4.4 is implemented through the use of a CollisionShape2D, this allows both the player hurtbox for swords and any projectiles detect wether their is collisions with the enemy or the enemy is within their hurtbox allowing them to call the enemies take damage function which will decrease their health based on the damage and decrease it more if the enemy is weak to the damage type. This can be seen in the video testing as the enemies correctly register hits from the players attacks and die after losing all their health.
4.5 is implemented through the use of a ProgressBar, using a custom texture to mimic health bars in other games, this HealthBar is set to show up whenever the take damage function is called to remind the player of the enemies remaining health the enemies also will queue_free upon losing all health. The healthbar can be seen through the UI testing here[6] and it displays the enemies health accurately, updating when they take damage aswell as the enemies dieing when they lose all their health.
4.6 melee enemies have been implemented using all of the above, they pathfind and move towards the player when in range as controlled by the script and they will attack the player upon colliding with them calling the players take damage function and starting a cooldown in which they cant attack again. These can be seen in the video testing, their is two different types with different skins that follow the enemy and deal damage upon contact accurately.
4.7 projectile enemies have been implemented using the same range magic projectiles used for the players magic weapons, the projectile enemies dont pathfind towards the player instead using the data to aim their projectiles. They fire at a constant rate while the player is within the range however their tracking data from the NavigationAgent2D only updates evry couple seconds and so they dont have full accuracy to reduce difficulty. This can be seen through the fire slime in the video testing that fires the fire projectiles, successfully dealing damage to the player.

Partially/Unsuccessfull:
4.8 the implementation of Boss enemies was unsuccessfull due to lack of time and lack of sprites and animations in order to properly implement it.
If I had more time and the correct animations I would implement the boss using the pathfinding abilities, hitbox, sight range and health tracking mechanics I already use for other enemies. Aswell as that boss enemies would have different attacks and have sequences of attacks they executed when the player was within a closer range (using another Area2D and CollisionShape2D to detect the closer range) the attack pattersn would start off consistent and predictable but when the health tracking reached a certain level the boss enemies would switch to different attack combinations. For the health tracking aswell I would change the healthbar to be a ui element using a CanvasLayer to display it at the top of the screen aswell as the name of the boss, this would put the focus on the boss enemy as the main event. In the bosses take damage function I would add extra code for when it dies to give the player the key to unlock the end of the level, as well as this I would add boss drops similar to the way I implemented chests with the item pool and choosing randomly from the array for each item to give to the player.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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 Dash Animation	5.1 Fully Met 5.2 Fully Met 5.3 Fully Met 5.4 Fully Met 5.5 Not Met 5.6 Not Met

Successfull:
The Appearance and animations of the player can be seen in the movement test here[1] and the combat test here[4].
5.1, 5.2 and 5.4 are all met through the use of the AnimatedSprite2D node on the player, this node allows the spriteframes to be input for each animation aswell as the frames per second of the animation, it is then activated at the relevant times using the code in the players physics process and the attack function. This can be seen through the player movement test where the animations displayed corresponded correctly to the walking direction aswell as the sprite being visible and recognisable. The combat animation can be seen in the combat video testing and activates in the correct direction when the attack key is pressed.
5.3 is implemented through the use of 8 directional animations, I used a naming convention in order to simplify the names of the animations and then I wrote a get animation function to get the name of the directional animation for the direction the player is facing of the specified type so that the animation can be played using the script. This can be seen in the video testing where the player sprite always faces the direction it last moved.
Partially/Unsuccessfull:
5.5 is not implemented due to lack of magic animation spritesheets for the animation.
If I had more time I would have been able to find or commision the relevant spritesheets for the animation and get all the directional animations set up using the AnimatedSprite2D and then set the animation to play when magic attacks happen instead of playing the melee animation.
5.5 is not implemented due to lack of dash animation spritesheets for the animation.
If I had more time I would have been able to find or commision the relevant spritesheets for the animation and get all the directional animations set up using the AnimatedSprite2D and get the animating to play when the dash key is pressed.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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≤chars<15) 6.5 Input Sanitisation (Removing any escape chars for SQL before sending the command) 6.6 Ability to log in with an exisiting account and correct password 6.7 Ability to reset password (With challenge question) 6.8 A general login form which links the other forms. 6.9 Ability to delete an account.	6.1 Fully Met 6.2 Fully Met 6.3 Fully Met 6.4 Not Met 6.5 Fully Met 6.6 Fully Met 6.7 Fully Met 6.8 Fully Met 6.9 Partially Met

Successfull:

The Database and functions part of the Login System is evidenced through my testing plan for the Database and login system and the forms is evidenced through the video testing here[2].

6.1 is implemented using the j_hash function and the gen_salt function to store unique salt for every user and use that to hash the passwords, the j_hash uses a combination of sha256 and md5 to hash the password while sprinkling the salt throughout by adding it onto the end making sure the hash will always end up the same length. This can be seen through the testing plan where the hash function is seen to be consistent and the salt function is seen to be random.

6.2 is implemented with the users table in the SQL database, this table stores the usernames, passwords, salt and challenge question answer, there is also several SQL queries to store and fetch data from the users table the SQL table is automatically created if it does not exist on the game starting up. This can be seen throughout the testing plan as all the SQL queries to operate on the table work correctly with the table.

6.3 is implemented with the create_account function which validates unique usernames, this function is used with the create account form which provides a user input form to input the username, password, challenge question answer and confirm the password to create the account while displaying errors. The challenge question was chosen to be "what was name of your first school?" as this will most likely have an answer for a majority of people playing the game. This can be seen through the testing plan as the create account function makes sure the username is unique and successfully creates a new account.

6.5 is implemented through the use of godot's query_with_bindings this takes in bindings for the SQL query and sanitises them first to stop escape chars from getting through. This can be seen through the stakeholder feedback 1 and 2 here[10] and here[11] as the stakeholders where not able to cause any issues with the login functions or database.

6.6 is implemented through the login function in the database script which checks the username and if it is valid will hash the password with the salt and check if it matches the stored password hash before logging the user in if it is correct or returning a relevant error if it fails any of the checks. This login function is used within a login form which provides inputs for all the fields necessary to login aswell as a button to attempt it. This can be seen through the testing plan succesfully logging in after creating an account.

6.7 is implemented through the use of a reset_password function in the database script which checks the username and if it is valid will hash the challenge answer with the salt and check if it matches the stored password hash before checking if the passwords match and changing the password if they do and returning relevant errors if it fails any of the checks. This reset_password function is used within a reset password form which provides the fields necessary to reset the password aswell as a button to attempt it. This can be seen through the testing plan as the tests show that the password was succesfully reset with the correct challenge question answer.

6.8 is implemented through buttons on the login form and the other forms in order to navigate to the other forms and back to the login form. This can be seen through the video testing as the login forms work succesfully to navigate the login system and link together with no hitches or errors.

Partially/Unsuccessfull:

6.4 is not met as I realised there is no need for this as the usernames will just be truncated due to the SQL library I am using and even if you input the shortened username it wont allow you to create a new account with that so it doesnt matter.

6.9 is partially met as I implemented the delete_user function which checks the username, hashes the password

and checks it matches and if so would delete the user from the database, deleting all their save data with cascading deletes, however due to a lack of time I wasnt able to include this in its own form or in the login form or save menu scene.
If I had more time I probably would have added a form you can go to from the login form in order to delete the user where you would need to input the username and password of the account before you could delete it.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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	7.1 Fully Met 7.2 Fully Met 7.3 Fully Met 7.4 Fully Met 7.5 Fully Met

All the UI elements where implemented using godot's CanvasLayer so that they dont move when the camera moves. The UI is evidenced through the video testing here[6]
7.1 is implemented through the use of a ProgressBar with a custom texture (red background green fill) to show the percentage of health left. The maximum value of the bar is set to the player health in the _ready() function of the UI scene and then the current value is updated in the _process function every frame to match the players current health. This can be seen in the video testing for the UI where the healthbar accurately displays the player health, updating when they take damage.
7.2 is implemented through the use of a ProgressBar with a custom texture (navy background purple fill) to show the percentage of magic left. The maximum value of the bar is set to the player magic points in the _ready() function of the UI scene and then the current value is updated in the _process function every frame to match the players current magic points. This can be seen in the video testing for the UI where the magic points bar accurately displays the player magic points, updating when they use it to use a weapon.
7.3 is implemented through creating a display string function for every item that displays the key information about it (damage, type, etc) and then taking this display string and displaying it in a label in the UI scene. This can be seen in the video testing of the UI as the weapon is accurately displayed even when the equipped weapon is switched.
7.4 is implemented through the use of a ProgressBar with a custom texture (red background green fill) to show the percentage of enemy health left, this is hidden by default. The maximum value of the bar is set to the enemy health in the _ready() function of the UI scene and then the current value is updated in the take_damage function aswell as showing the healthbar above the enemies head till their damage taking animation finishes. This can be seen in the video testing of the UI as the enemy healthbar pops up above their head when they take damage and accurately displays their health.
7.5 is implemented through the success criteria for the inventory system (12)

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
8	Weapons And a More Advanced Combat System	8.1 Different Styles of melee, magic and ranged weapons 8.2 Boss Drops 8.3 Shop System that appears throughout levels 8.4 Charged Attacks (based on how long you hold down) 8.5 Special attacks	8.1 Partially Met 8.2 Not Met 8.3 Not Met 8.4 Not Met 8.5 Not Met

8.1 was partially met in that I had many different styles of magic and ranged weapons (with the different projectile and area of effect types). This can be seen through the video testing of the combat system and enemies here[4].
If I had extra time I would have added bows aswell as different shorter and longer ranged melee weapons by creating different sized hurtboxes and through the methods I detailed in my evaluation of criteria 2.4.
8.2 was not met due to bosses not being implemented however I already detailed how I would implement boss drops in a similar style to the chests item pool in my evaluation of criteria 4.8.
8.3 was not met due to a lack of time and due to it being only a desireable feature so I implemented other stuff first.
If I had the extra time I would have implemented it in two parts, first part is in creating an npc that wonders the dungeons they npc would not get attacked by monsters unless a projectile fired hits them, the npc would have an Area2D node to detect when the player is close and then the player could open up the shop menu. The shop menu is the second part that I would implement it would allow the player to sell their gathered items for

gold and use that gold to buy from a randomized selection of goods, I would use a CanvasLayer and similar techniques to the Inventory UI in pausing the game. I would use a ScrollContainer and a VBoxContainer to contain the buttons corresponding to the items the player can buy linking them to a function to purchase them. I would make it so the player can buy back the last item they sold to the shop npc (for a higher price of course) this will be added as a button at the bottom of the other items.
8.4 was not met due to a lack of time and due to it being only a desireable feature so it was not the priority. I have detailed how I might implement charged attacks for the different weapon types in my evaluation of criteria 2.2.
8.5 was not met due to a lack of time and due to it being only a desireable feature so I didnt get onto it.
If I were to get more time to get onto implementing 8.5 I would implement special attacks as a more powerful version of the main attack with more damage and a larger hitbox/projectile than the standard version, this would be useful for clearing large hordes of enemies but would have a longer cooldown timer so that it cannot just be spammed I would have to find another keybind in order to allow players to use this attack.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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	9.1 Not Met 9.2 Not Met 9.3 Not Met 9.4 Not Met 9.5 Not Met

The whole of 9 was not met sadly even though it was highly requested by stakeholders, this is because it would not add as much to the finished product for the workload and I ran out of time although it would have been the next set of criteria I would work on if I had more time.
9.1 If I had more time I would implement this UI menu by researching and using a tree visualisation algorithm on the skill tree to display the tree in a nice neat format with all the skills as buttons that are linked to a function to try and unlock them the button texture would change to show the locked skills, alternatively I could manually make the tree structure if the implementation of the tree visualisation algorithm proved to be too hard.
9.2 If I had more time I would implement this by seperating the skill tree into 4 different trees for displaying it using a graph traversal to build them, one for each skill type, however if skills might need skills from multiple trees as prerequesites then the skills from the tree it doesnt belong too will be shown in the button.
9.3.1 If I had more time I would implement this by changing the enemies take damage function to add experience to the global variable to keep track of it when they die, the amount of experience would depend on the difficulty, level and type of enemy with bosses giving considerably more experience.
9.3.2 If I had more time I would implement this through my implementation of the graph data structure for the skill tree, this data structure would allow each node to have a list of children (skills it is needed for) and a list of parents (prerequesites), the skills purchase function would check if the experience meets the requirements before deducting that amount of experience from the player.
9.4 If I had more time I would implement this by making it so that you can unlock skills through the skill tree UI menu by pressing on a locked skills button, this will call the unlock skill function which will check if all the prerequesites are unlocked and the player has enough experience before deducting the experience and unlocking the skill.
9.5 If I had more time I would implement this through adding a button to the UI menu which will reset the graph structure for the skill tree and update the UI aswell as give the player back the experience they spent on unlocking skills so that they can spend it again.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
10	Procedurally Generated Dungeons	10.1 Creating requirements for each level to satisfy 10.2 Creating different 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.)	10.1 Fully Met 10.2 Partially Met 10.3 Fully Met 10.4.1 Fully Met 10.4.2 Partially Met

Successfull:

The procedural generation of the dungeons is evidenced through the testing here[7] 10.1 I implemented a custom graph structure where each room/corridor is a node, these nodes have variables to store the rooms in each of the cardinal directions through aliasing, they also have a variable to store the room type, this allows me to create level requirements by creating a graph structure with all the rooms I want to be a requirement as they will have to be included every time. This could be seen in the procedural generation testing as the same level always generated with the required length and rooms.

10.3 I implemented the algorithm in two parts, firstly in each room I designed I added nodes to mark the entrances in the cardinal directions I created functions to cap the entrances aswell as ones to get the global position and set the position of the room such that the position of an entrance lines up with a certain position, this allows me to line up entrances of rooms through getting ones position and setting anothers to line up with that position. The second part was in traversing the graph and generating the rooms, I used a recursive implementation of a depth first search to iterate through each node in the graph picking a random room of the specified type and instantiating it into the scene lining it up with the previous room in the tree. This can be seen in the video testing as the dungeon rooms are succesfully peiced together to create the final environment.

10.4.1 I implemented this by making each of the rooms of the same type be roughly the same length, this means that whenever a level is generated the length will only be between a maximum and minimum length. This can be seen in the video testing as all the levels I generated where within a consistent range of lengths.

Partially/Unsuccessfull:

10.2 I partially met this requirement as I didnt make chest rooms or many different shaped rooms due to a lack of time. The other rooms can be seen through the video testing making up the generated level.

If I had extra time I would remedy this by making chest and boss rooms as detailed in my evaluation of criteria 3.4. I would also make more interesting shaped rooms by taking the standard rectangle shape and keeping the footprint adding obstacles or corridors winding through them which would allow for more variation in levels generated.

10.4.2 I partially met this requirement as I didnt make all of the rooms I would have liked including secret rooms. However out of the rooms I made the levels will always contain the rooms required which can be seen within the video testing as they always have the same amount of the same room types.

If I had more time after I made more rooms I would add them in at different points in the level to create more incentive to explore different areas.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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.	11.1 Not Met 11.2 Not Met

For 11 I didnt get onto this as it was onyl a desireable feature and I ran out of time to keep implementing them.

11.1.1 To add a hammer I would use my existing item class hierarchy adding in a new class for tools and a child class for hammers, the hammer tool would be a tool that when in your inventory would allow you to

walk up to specially made slightly ruined walls and get detected with an Area2D node, this would allow you to interact with the wall and if you have the hammer to knock it down revealing a part of the level previously hidden to you. I could also implement a hardness feature which means that certain hammers would only be able to knock down weaker walls whereas other ones that are obtained later could knock down stronger ones.
11.1.2 To add climbing gloves I would again use a child class of the tools class for a climbing gloves class, I would need to implement a climbing animation for use when the player tries to traverse up vines which would only activate if the player had the climbing gloves, The vines would change the scene to a hidden level scene. I could again add different levels of climbing gloves or perhaps other climbing gear to allow the player to climb harder walls/vines.
11.2 To implement secret boss and treasure rooms for behind these obstacles I would use seperate scenes from the procedural generated rooms using the TileMap to create the environment, I would not instantiate these scenes into the level until the player conquered the obstacles. The boss and treasure rooms would include bosses and chests with unique loot pools often containing special items that you might not usuaully be able to obtain at that level and they would be structured similarily to how I detailed in my evaluation of criteria 3.4.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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. 12.5 Ability to add or remove items from the inventory. 12.6 SQL table to store inventory contents	12.1 Fully Met 12.2 Fully Met 12.3 Fully Met 12.4 Fully Met 12.5 Fully Met 12.6 Fully Met

Most of the inventory functions are in the inventory.gd and the database.gd script which is tested through my testing plan there and the UI of it is evidenced in the UI testing here[6].
12.1 is implemented similarily to the other UI, there is a canvas layer that pops up and paused the proccess scripts of all the enemies and player upon pressing the inventory key, this layer contains a ScrollContainer for the inventory items aswell as buttons for binning an item, equipping an item and logging out (requested by stakeholders). The ScrollContainer is set up using the list of items from the get_stored_items function it then uses their display strings in order to display them all on buttons 4 across (using a HBoxContainer) and scrolling down (using a VBoxContainer) this creates a scrollable inventory to view all the items. This can be seen in the UI testing as the UI accurately displays information about both the equipped and stored items in a user friendly way.
12.2 is implemented through fulfilling 12.6 and through storing the paths to the item resource file in the SQL table so that you can fetch all the information about the resource from loading it from the path. This can be seen in my testing plan showing the succesfull storage and addition of items to be equipped and removed.
12.3 is implemented through the use of godot's _input function it catches when the inventory action (E key) is pressed and then pauses the other scripts and instantiates the InventoryUI scene. This can be seen in my video testing as the UI opened succesfully upon pressing the E key.
12.4 is implemented through the use of the equip_item function which uses SQL queries to change the equipped item in a slot in the save_data SQL table aswell as the unequip_item function to add the previously equipped item back into the stored_items SQL table. It also removes the item equipped from the stored items table using the remove_stored_item function. These functions are attatched to the equip item button in the InventoryUI scene which will take the last selected item button and equip it using the function. This can be seen succesfully in the video testing as I switched weapons and was able to use the other weapon succesfully. You can also see this in the testing plan results as the functions succesfully equipped and unequipped items.
12.5 is implemented through the database.gd file, specifically the get_stored_items function (Gets an array of the stored items for the save) the get_stored_item_amount function which is used in the remove_stored_item function to see if there is enough of the item to remove. There is also the add_stored_item function. These functions interact with the stored_items SQL table in order to update the items stored there. This is demonstrated succesfully in the testing plan through the tests of the functions which succesfully added and removed items from the inventory.
12.6 is implemented through the stored_items and save_data SQL tables, these tables are created using SQL queries to store all the stored_items and equipped_items respectively and store them through the use of storing the path to the item resource. These tables are automatically created if they do not exist on the game startup. This is demonstrated in the testing as the script succesfully created and used the SQL table for the storage of the inventory contents.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
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.	13.1 Not Met 13.2 Not Met 13.3 Not Met

Criteria 13 was a desireable criteria and so was not implemented due to lack of time and due to the fact that there wasn't any sounds implemented in the game yet to control the volume of.
13.1 If I had the time 13.1 would be implemented through a UI menu similar to the InventoryUI based on a CanvasLayer that would pause the game processes when they key to open it is pressed. I would use godot's sliders for toggling difficulty, add a button for toggling hardcore which would use SQL queries to update the save_data table and a button to quit to the save menu.
13.2 If I had the time and implemented audio I would make sure I implemented the audio by using godot's AudioServer to manage the AudioBuses for the different types of audio (music, sound effects etc) which allows me to control the volumes seperately. I would then add a slide for each audio bus to the settings UI and make it so upon the slider value being set it would change the volume value of the relevant bus on the AudioServer.
13.3 If I had the time I would add a dropdown menu with different settings for the colours of the game, I would add a tint setting through the use of a CanvasItemShader on a CanvasLayer then use godot's shader coding in order to code a shader that shifts the colour of the pixels towards the relevant colour. I could also add a vibrancy slider through another shader that keeps the ratio's of colours RGB values but decreases the values overall to create less vibrant colours.

Criteria #	Abstraction	Success Criteria	Not / Partially / Fully Met
14	Difficulty Levels and Hardcore Mode	14.1 A slider for difficulty in create save 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) 14.4 SQL table to store different saves	14.1 Fully Met 14.2.1 Fully Met 14.2.2 Not Met 14.2.3 Partially Met 14.3 Not Met 14.4 Fully Met

Successfull:
My evidence of setting difficulty levels and hardcore mode can be seen in the login forms testing here[2].
14.1 is implemented through Godot's slider in the save menu scene used to set the difficulty from which the value is saved in the save_data table for use when creating a new save. This can be seen to be succesfull in the menu test as the saves are created with the relevant difficulty.
14.2.1 is implemented through the _ready function in the room scripts which sets all the enemies in a room to have a health that increases as the level and difficulty increase. This can be seen through the stakeholder feedback 3 here[12] where Sam chooses a lower difficulty and enemy health is lower because of it.
14.4 is implemented through the save_data table which stores the save_id, user_id, difficulty, equipped_items and wether the save is hardcore. This table is automatically created in the database if it does not exist on startup.
Partially/Unsuccessfull:
14.2.2 was not met due to lack of time however it could have been implemented through setting the players health to decrease based on the difficulty but never end up at zero with a function like health = $\frac{100}{\sqrt{\text{difficulty}}}$ running on the player being loaded in to set the health.
14.2.3 was not partially met as instead of increasing the number of enemies I increased enemy damage which can have a similar effect and was easier to implement with the lack of time. This can be seen through the stakeholder feedback 3 here[12] where Sam chooses a lower difficulty and enemy damage is lower because of it. If I had more time I could have implemented 14.2.3 by making it so enemies had a chance to duplicate by instantiating another enemy scene upon being loaded in, this chance would be small but would increase as the difficulty increases.
14.3 was not implemented due to it being a fairly large criteria to implement for not as much benefit to

stakeholders aswell as a lack of time.
I could implement several roguelike features if I had more time. Permadeath could be implemented through either deletion of the save when the player dies through an SQL query or by resetting the player back to the first level but letting them keep certain stuff such as the skill tree or certain unlockable items which would not get deleted.

4.2 Stakeholder Feedback 3

In this stakeholder feedback I showed stakeholders the improvements I had made based on their feedback last time aswell as the procedural generation and level 1 implemented. The video can be found here[12].
I considered all of the fixes/ future features they suggested

4.2.1 Save Menu

The stakeholders suggested a change to the Hardcore button such that it would be more obvious when the button is turned on vs turned off in order to fix this I would link the button press to a function in the save_menu script that would change the text in the button to indicate wether or not hardcore was set to true or false.
The stakeholders where slighly annoyed when the save button in the Save Menu did not take them to the next level after having beaten the tutorial, in order to fix this, as the level updates successfully, the button function can be set to take the player to the level they are on through the level stored in the save data table and using string substitution to fetch the right scene to switch to.

4.2.2 Levels

The stakeholders where dissapointed in the lack of chests within the procedural generation. I talked about how I would implement this given more time in my evaluation of criteria 3.4.
The stakeholders wanted different direction of generation for other levels, I can implement this through creating seperate level structures using the DungeonGraph class that lead in different directions or even wrap around the start before finishing.
Samu commenteed on the ease of dodging enemies within the level which can lead to less challenge. I would fix this through making the corridors narrower and the rooms more dynamic with obstacles and walls to avoid rather than being able to walk straight from entrance to exit. This would make it easier for the player to be faced with enemies that they cannot dodge and therefore have to fight.
Sam also commented on the player getting the most powerful weapon straight away within the tutorial. I would fix this by implementing a wider variation of weapons at each damage level and only giving weapons of a certain damage level proportional to the level the player is in.

4.2.3 Inventory and Items

In the tutorial he player was able to pick up the same weapon twice when weapon is equipped which stops them from only having one copy of each weapon momentarily. I would fix this through checking if the item being added to the inventory matches an item that is equipped, if so the player wouldnt pick up the item therefore preventing this bug.
I would implement the player armour through the existing class for armour and by giving it to the player in chests. The armour would allow the player to take less damage by absorbing the same amount of damage as defence each peice has. I could also add set bonuses for when all the peices of a set of armour are equipped, these bonuses could further up the players defense or even boost the attack power through adding to the weapons attack power before attacking in the script.

4.2.4 Maintenance

In order for stakeholders to effectively use the solution in the future there would need to be maintenance including the oppurtunity to report bugs to me for fixing. Another thing that could be an issue for maintenance is the amount of content in the game, with the amount of content currently implemented the stakeholders would not be able to be entertained for long and so this would have to be remedied through content updates in maintenance.
Considered Maintenance issues and limitations

4.3 Final Evaluation

Overall I feel like the NEA project was a partial success, I met most of my basic success criteria and created some amount of playable game content with features that my stakeholders where happy with in the final feedback. I did however fall short of my own expectations both in terms of implementing some of the more problem solving focused elements of the criteria aswell as ending up with a final project that felt like it had enough content and replayability in order to make it a worthwhile investment compared to alternatives my stakeholder's might consider. I feel as though I had most of the core features and building blocks set up in order to, with little extra effort, provide a more content heavy experience that could fully satisfy my stakeholder's requirements for both replayability and general intuitiveness and accessibility to new players and players with special requirements.

I do feel as though I have developed my skills, both in fluency with the concept of a long form coding project with specific requirements to fulfill and in various coding concepts both within and outside of the A level spec. I have solidified my theoretical and practical knowledge in a variety of skills within the OCR specification. Object Oriented Programming through the direct approach with my item classes using inheritance and polymorphism and the indirect ways it came into play with godot's node structure of encapsulation where nodes can contain other nodes to add functionality. Software Development life cycles, although I didnt follow a specific life cycle choosing instead to follow a mostly waterfall approach with some agile components the project helped me learn about the practical application including benifits and drawbacks of the different approaches within a stakeholder driven development process, it also helped me realise that the specific approaches are not as clear cut and many different approaches can be made through adapting elements of the specific approaches. I learn about the uses and practical implementations of alot of the common data structures and algorithms taught in Computer Science, including how the A* algorithm can be adapted to work using a general area through the use of a mesh rather than a manual graph structure which cannot support pathfinding to points inbetween the nodes. I learnt about SQL databases including many practical applications and commands need to make tables viable aswell as the different considerations to make when considering the database structure (how data needs to be accessed, normal forms, etc).

5 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
6	04/03/25	Design/ Enemy Design	Game Engine Documentation, online	Godot Docs (2D navigation)	The documentation for godot's navigation nodes in 2D

6 Code Listings

6.1 resources/scripts

item.gd

```
extends Resource

class_name Item

@export var stackable: bool
@export var description: String
@export var icon: Texture
```

equipable.gd

```
extends Item

class_name Equipable

@export var stat_boosts: Dictionary = {}

func _init():
        stackable = false
```

armour.gd

```
extends Equipable

class_name Armour

@export var defense: int
@export var armour_type: String # "light", "heavy"
@export var body_part: String # "head", "chest", "legs"
@export var set_id: int

func display_string():
        return "Armour\nName: %s\nType: %s\nDefense: %s\n Description: %s" % [resource_name,
            armour_type.capitalize(), defense, description]
```

weapon.gd

```
extends Equipable

class_name Weapon

@export var weapon_type: String # "ranged_magic", "area_magic", "melee"
@export var attack_power: int
@export var attack_range: int
@export var damage_type: String # "normal", "ice", "fire", "thunder", "divine","poison"
@export var mana_cost: int
var hurtbox_scene_path = "res://scenes/game/player/hurt_box.tscn"

func display_string():
        return "Weapon\nName: %s\nType: %s\nDamage: %s\nDamage Type: %s\n Description: %s" % [
            resource_name, weapon_type.replace("_"," ").capitalize(), attack_power, damage_type.
            capitalize(), description]
```

charm.gd

```
extends Equipable

class_name Charm

@export var charm_type: String # "ice", "fire", "cursed", "divine","poison"

func display_string():
        return "Weapon\nName: %s\nType: %s\n Description: %s" % [resource_name, charm_type.
            capitalize(), description]
```

key.gd

```
extends Item

class_name Key

@export var key_id: String


func display_string():
        return "Key\nName: %s\n Description: %s" % [resource_name, description]
```

6.2 scenes

6.2.1 game/enemies

default_slime.gd

```
extends CharacterBody2D



var direction = Vector2(0,1)
@export var speed:int = 20
@export var health: int = 10
@export var damage: int = 1
@export var damage_type: String = "normal"
@export var weaknesses: Array
var animating: bool
var can_attack = true

func get_animation(animation_type: String):
        if abs(direction.x) > abs(direction.y):
                if direction.x > 0:
                        return animation_type + "_r"
                else:
                        return animation_type + "_l"
        else:
                if direction.y > 0:
                        return animation_type + "_d"
                else:
                        return animation_type + "_u"

func _ready():
        add_to_group("enemies")
        $HealthBar.max_value=health
        $HealthBar.value=health

func _physics_process(delta: float) -> void:
        # If player in range sets move to true
        var move = false
        for body in $Area2D.get_overlapping_bodies():
                if body.name == "Player":
                        move = true
        #Moves or idles if not animating
        if not animating:
                if move:
                        direction = to_local($NavigationAgent2D.get_next_path_position()).
                            normalized()
                        velocity = direction * speed
                        $AnimatedSprite2D.play(get_animation("walk"))
                        move_and_slide()
                else:
                        $AnimatedSprite2D.play(get_animation("idle"))
        else:
                move_and_slide()

        #Attacks player if collides with them and can attack
        if can_attack:
                for i in range(get_slide_collision_count()):
                        var collision = get_slide_collision(i)
                        var collider = collision.get_collider()
                        if collider.is_in_group("player"):
                                collider.take_damage(damage, damage_type)
                                can_attack=false
```

```
57                              $AttackTimer.start()
58
59 func take_damage(damage, damage_type):
60         var player = get_tree().get_first_node_in_group("player")
61         animating = true
62         if damage_type in weaknesses: #Takes more damage from weaknesses
63                 health -= 2*damage
64         else:
65                 health -= damage
66         velocity = - 25 * to_local(player.global_position).normalized() #takes knockback by
               changing velocity
67         $HealthBar.value=health
68         $HealthBar.visible = true
69         if health <= 0: #dies if health is low
70                 queue_free()
71         else:
72                 $AnimatedSprite2D.play(get_animation("hurt"))
73         await $AnimatedSprite2D.animation_finished
74         animating = false
75         velocity = Vector2(0,0) #Removes knockback velocity
76         $HealthBar.visible = false
77
78 func _on_navigation_timer_timeout() -> void:
79         var player = get_tree().get_first_node_in_group("player")
80         if player:
81                 $NavigationAgent2D.set_target_position(player.global_position)
82
83 func _on_attack_timer_timeout() -> void:
84         can_attack = true
```

fire_slime.gd

```
1  extends CharacterBody2D
2
3
4
5  var direction = Vector2(0,1)
6  @export var speed:int = 20
7  @export var health: int = 10
8  @export var damage: int = 1
9  @export var damage_type: String = "fire"
10 @export var weaknesses: Array
11 var animating: bool
12 var can_attack = true
13
14 func get_animation(animation_type: String):
15         if abs(direction.x) > abs(direction.y):
16                 if direction.x > 0:
17                         return animation_type + "_r"
18                 else:
19                         return animation_type + "_l"
20         else:
21                 if direction.y > 0:
22                         return animation_type + "_d"
23                 else:
24                         return animation_type + "_u"
25
26
27 func _ready():
28         add_to_group("enemies")
29         $HealthBar.max_value = health
30         $HealthBar.value=health
31
32 func _physics_process(delta: float) -> void:
33         # If player in range sets detected to true
34         var player_detected = false
35         for body in $Area2D.get_overlapping_bodies():
36                 if body.name == "Player":
37                         player_detected = true
38         #Attacks if not animating and can attack
39         if not animating:
40                 $AnimatedSprite2D.play(get_animation("idle"))
41                 if player_detected and can_attack:
42                         direction = to_local($NavigationAgent2D.get_next_path_position()).
                               normalized()
43                         var projectile = load("res://scenes/game/projectiles/fire_projectile.tscn
                               ").instantiate()
44                         projectile.rotation_degrees = rad_to_deg(direction.angle())
```

```
                        projectile.position = position + 20*direction
                        projectile.damage = damage
                        get_parent().add_child(projectile)
                        can_attack=false
                        $AttackTimer.start()


func take_damage(damage, damage_type):
        var player = get_tree().get_first_node_in_group("player")
        animating = true
        if damage_type in weaknesses: #Takes more damage from weaknesses
                health -= 2*damage
        else:
                health -= damage
        velocity = - 25 * to_local(player.global_position).normalized() #takes knockback by
            changing velocity
        $HealthBar.value=health
        $HealthBar.visible = true
        if health <= 0: #dies if health is low
                queue_free()
        else:
                $AnimatedSprite2D.play(get_animation("hurt"))
        await $AnimatedSprite2D.animation_finished
        animating = false
        velocity = Vector2(0,0) #Removes knockback velocity
        $HealthBar.visible = false



func _on_navigation_timer_timeout() -> void:
        var player = get_tree().get_first_node_in_group("player")
        if player:
                $NavigationAgent2D.set_target_position(player.global_position)


func _on_attack_timer_timeout() -> void:
        can_attack = true
```

poison_slime.gd

```
extends CharacterBody2D



var direction = Vector2(0,1)
@export var speed:int = 20
@export var health: int = 10
@export var damage: int = 1
@export var damage_type: String = "poison"
@export var weaknesses: Array
var animating: bool
var can_attack = true

func get_animation(animation_type: String):
        if abs(direction.x) > abs(direction.y):
                if direction.x > 0:
                        return animation_type + "_r"
                else:
                        return animation_type + "_l"
        else:
                if direction.y > 0:
                        return animation_type + "_d"
                else:
                        return animation_type + "_u"


func _ready():
        add_to_group("enemies")
        $HealthBar.max_value = health
        $HealthBar.value=health

func _physics_process(delta: float) -> void:
        # If player in range sets move to true
        var move = false
        for body in $Area2D.get_overlapping_bodies():
                if body.name == "Player":
                        move = true
        #Moves or idles if not animating
```

```
        if not animating:
                if move:
                        direction = to_local($NavigationAgent2D.get_next_path_position()).
                            normalized()
                        velocity = direction * speed
                        $AnimatedSprite2D.play(get_animation("walk"))
                        move_and_slide()
                else:
                        $AnimatedSprite2D.play(get_animation("idle"))
        else:
                move_and_slide()

        #Attacks player if collides with them and can attack
        if can_attack:
                for i in range(get_slide_collision_count()):
                        var collision = get_slide_collision(i)
                        var collider = collision.get_collider()
                        if collider.is_in_group("player"):
                                collider.take_damage(damage, damage_type)
                                can_attack=false
                                $AttackTimer.start()



func take_damage(damage, damage_type):
        var player = get_tree().get_first_node_in_group("player")
        animating = true
        if damage_type in weaknesses: #Takes more damage from weaknesses
                health -= 2*damage
        else:
                health -= damage
        velocity = - 25 * to_local(player.global_position).normalized() #takes knockback by
            changing velocity
        $HealthBar.value=health
        $HealthBar.visible = true
        if health <= 0: #dies if health is low
                queue_free()
        else:
                $AnimatedSprite2D.play(get_animation("hurt"))
        await $AnimatedSprite2D.animation_finished
        animating = false
        velocity = Vector2(0,0) #Removes knockback velocity
        $HealthBar.visible = false



func _on_navigation_timer_timeout() -> void:
        var player = get_tree().get_first_node_in_group("player")
        if player:
                $NavigationAgent2D.set_target_position(player.global_position)


func _on_attack_timer_timeout() -> void:
        can_attack = true
```

6.2.2 game/player

player.gd

```
extends CharacterBody2D


var speed = 100
var health = 100
var mana = 100
var last_direction = Vector2(1,0)
var animating = false
var weapon
var can_dash = true
var ui = preload("res://scenes/game/player/ui.tscn")
func _ready():
        add_to_group("player")
        add_child(ui.instantiate())

```

```
#Function to return the directional name of the animation in the direction player ois facing
func get_animation(animation_type: String):
	var anim = animation_type + "_"
	if last_direction.x:
		if last_direction.x == 1:
			anim += 'r'
		else:
			anim += 'l'
	if last_direction.y:
		if last_direction.y == 1:
			anim += 'd'
		else:
			anim += 'u'
	return anim


func _physics_process(delta: float) -> void:
	# Get the direction
	var direction = Vector2(Input.get_axis("left", "right"),Input.get_axis("up","down"))
	# Velocity
	velocity = direction.normalized() * speed
	#Moving or Idling if its not already animating
	if not animating:
		if direction:
			last_direction = direction
			$AnimatedSprite2D.play(get_animation("walk"))
		else:
			$AnimatedSprite2D.play(get_animation("idle"))
		move_and_slide()

#Handles different input types
func _input(event: InputEvent) -> void:
	if event.is_action_pressed("attack"):
		attack()
	elif event.is_action_pressed("dash"):
		if can_dash: # Increases speed momentarily if you can dash
			speed = 8*speed
			can_dash = false
			await get_tree().create_timer(0.05).timeout
			speed = speed/8
			await get_tree().create_timer(0.5).timeout
			can_dash = true
	if event.is_action_pressed("help"): #Loads help emnu and pauses till key pressed again
		var help_menu = load("res://scenes/menu/help_menu.tscn").instantiate()
		add_child(help_menu)
		get_tree().paused = true
		await get_tree().create_timer(0.2).timeout
		while not Input.is_action_just_pressed("help"):
			await get_tree().process_frame
		help_menu.queue_free()
		get_tree().paused = false
	elif event.is_action_pressed("inventory"): #Loads Inventory and pauses till key pressed
	    again
		var inventory_ui = load("res://scenes/game/player/inventory_ui.tscn").instantiate
		    ()
		$UI.visible = false
		add_child(inventory_ui)
		get_tree().paused = true
		await get_tree().create_timer(0.2).timeout
		while not Input.is_action_just_pressed("inventory"):
			await get_tree().process_frame
		inventory_ui.queue_free()
		$UI.visible = true
		get_tree().paused = false




func attack():
	#Gets weapon and attacks if it isnt animating and you have the mana
	if Database.get_slot_value("weapon"):
		weapon = load(Database.get_slot_value("weapon"))
		if not animating and mana >= weapon.mana_cost:
			animating = true
			mana -= weapon.mana_cost
			if weapon.weapon_type == "melee":
				$AnimatedSprite2D.play(get_animation("melee"))
```

```
                                await get_tree().create_timer(0.5).timeout # Account for
                                    animation delay
                                #Load the hurtbox scene
                                var hurtbox_scene = load(weapon.hurtbox_scene_path)
                                var hurtbox_instance = hurtbox_scene.instantiate()
                                hurtbox_instance.range = weapon.attack_range
                                hurtbox_instance.damage = weapon.attack_power
                                hurtbox_instance.damage_type = weapon.damage_type
                                hurtbox_instance.rotation = last_direction.angle()
                                #Add child
                                add_child(hurtbox_instance)
                                #Hitbox lasts for a tenth of a second
                                await get_tree().create_timer(0.1).timeout
                                print("yes")
                                hurtbox_instance.queue_free()
                        elif weapon.weapon_type == "ranged_magic":
                                $AnimatedSprite2D.play(get_animation("melee")) #Change later
                                await get_tree().create_timer(0.5).timeout # Account for
                                    animation delay
                                #Load the projectile scene
                                var projectile = load("res://scenes/game/projectiles/%
                                    s_projectile.tscn"%weapon.damage_type).instantiate()
                                var projectile_direction = (get_viewport().get_mouse_position()-
                                    Vector2(1920/2,1080/2)).normalized()
                                projectile.rotation_degrees = rad_to_deg(projectile_direction.
                                    angle())
                                projectile.position = position + 20 * projectile_direction
                                projectile.damage = weapon.attack_power
                                get_parent().add_child(projectile)
                        elif weapon.weapon_type == "area_magic":
                                $AnimatedSprite2D.play(get_animation("melee")) #Change later
                                await get_tree().create_timer(0.5).timeout # Account for
                                    animation delay
                                #Load the projectile scene
                                var area_direction = (get_viewport().get_mouse_position()-Vector2
                                    (1920/2,1080/2)).normalized()
                                var area = load("res://scenes/game/projectiles/%s_area.tscn"%
                                    weapon.damage_type).instantiate()
                                area.position = position + 25 * area_direction
                                area.damage = weapon.attack_power
                                get_parent().add_child(area)
                        await $AnimatedSprite2D.animation_finished
                        animating = false


func take_damage(damage, damage_type): #Removes the amount of health for the damage
        health -= damage
        if health <= 0:
                get_tree().change_scene_to_file("res://scenes/menu/save_menu.tscn")
```

hurt_box.gd

```
extends Area2D

var damage: int
var damage_type: String
var range: int


func _on_body_entered(body: Node2D) -> void:
        if body.is_in_group("enemies"): # If an enemy enters damages it
                body.take_damage(damage, damage_type)
```

inventory_ui.gd

```
extends CanvasLayer

var selected: String = ""
var hbox: HBoxContainer
func _ready() -> void:
        process_mode = Node.PROCESS_MODE_ALWAYS
        #var inventory = [{"item_id": "res://resources/equipable/weapon/test_weapon_magic_area.
            tres","amount":1},{"item_id": "res://resources/equipable/weapon/
            test_weapon_magic_ranged.tres","amount":1},{"item_id": "res://resources/equipable/
            weapon/test_weapon_melee.tres","amount":1},{"item_id": "res://resources/key/test_key.
            tres","amount":1},{"item_id": "res://resources/equipable/armour/test_armour.tres","
```

```
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1},{"item_id": "res://resources/key/
            test_key_2.tres","amount":1},{"item_id": "res://resources/key/test_key_2.tres","
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1},{"item_id": "res://resources/key/
            test_key_2.tres","amount":1},{"item_id": "res://resources/key/test_key_2.tres","
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1},{"item_id": "res://resources/key/
            test_key_2.tres","amount":1},{"item_id": "res://resources/key/test_key_2.tres","
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1},{"item_id": "res://resources/key/
            test_key_2.tres","amount":1},{"item_id": "res://resources/key/test_key_2.tres","
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1},{"item_id": "res://resources/key/
            test_key_2.tres","amount":1},{"item_id": "res://resources/key/test_key_2.tres","
            amount":1},{"item_id": "res://resources/key/test_key_2.tres","amount":1},{"item_id":
            "res://resources/key/test_key_2.tres","amount":1}]
        refresh()

func refresh():
        var inventory = Database.get_stored_items()
        var item_count = 0
        #Gets rid of existing children
        for child in $ScrollContainer/VBoxContainer.get_children():
                child.queue_free()
        for item in inventory: #Creates a button to select an item that displays key info about
            the item
                var button = Button.new()
                if item_count % 4 == 0:
                        hbox = HBoxContainer.new()
                        $ScrollContainer/VBoxContainer.add_child(hbox)
                        print(hbox)
                button.text = load(item["item_id"]).display_string() + "\nAmount: %d" % item["
                    amount"]
                button.connect("pressed",_select.bind(item["item_id"]))
                button.custom_minimum_size = Vector2(250,200)
                hbox.add_child(button)
                item_count += 1
        #Creates labels for all the equipped items
        if Database.get_slot_value("weapon"):
                $WeaponLabel.text = load(Database.get_slot_value("weapon")).display_string()
        if Database.get_slot_value("head"):
                $HeadLabel.text = load(Database.get_slot_value("head")).display_string()
        if Database.get_slot_value("chest"):
                $ChestLabel.text = load(Database.get_slot_value("chest")).display_string()
        if Database.get_slot_value("legs"):
                $LegsLabel.text = load(Database.get_slot_value("legs")).display_string()
        if Database.get_slot_value("charm_1"):
                $Charm1Label.text = load(Database.get_slot_value("charm_1")).display_string()
        if Database.get_slot_value("charm_2"):
                $Charm2Label.text = load(Database.get_slot_value("charm_2")).display_string()

#Changes selected item
func _select(item_id):
        selected = item_id

#Equips selected item if selected
func _on_equip_button_pressed() -> void:
        if selected != "":
                Inventory.equip_item(selected)
                refresh()

#Removes selected item if selected
func _on_bin_button_pressed() -> void:
        if selected != "":
                Database.remove_stored_item(selected)
                refresh()


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

ui.gd

```
extends CanvasLayer

var player

```

```
func _ready():
	var player = get_tree().get_first_node_in_group("player")
	$HealthBar.max_value = player.health
	$MagicBar.max_value = player.mana
	update_ui()


func _process(delta: float) -> void:
	var player = get_tree().get_first_node_in_group("player")
	$HealthBar.value = player.health
	$MagicBar.value = player.mana


func update_ui():
	if Database.get_slot_value("weapon"):
		var weapon = load(Database.get_slot_value("weapon"))
		$WeaponLabel.text = weapon.display_string()
```

6.2.3 game/projectiles

ice_projectile.gd

```
extends CharacterBody2D

var damage: int = 1
var damage_type: String = "ice"
var speed = 120
var attacking = false


func _physics_process(delta: float) -> void:
	#Gets movement vector
	velocity.x = speed * cos(rotation)
	velocity.y = speed * sin(rotation)
	if not attacking:
		move_and_slide()
	#Damages player or enemy if collides with them
	for i in range(get_slide_collision_count()):
		if not attacking:
			attacking = true
			var collision = get_slide_collision(i)
			print(collision)
			var collider = collision.get_collider()
			print(collider)
			if collider.is_in_group("player") or collider.is_in_group("enemies"):
				collider.take_damage(damage, damage_type)
			$CollisionShape2D.disabled = true
			$AnimatedSprite2D.play("impact")
			await $AnimatedSprite2D.animation_finished
			queue_free()
```

fire_projectile.gd

```
extends CharacterBody2D

var damage: int = 1
var damage_type: String = "fire"
var speed = 120
var attacking = false


func _physics_process(delta: float) -> void:
	#Gets movement vector
	velocity.x = speed * cos(rotation)
	velocity.y = speed * sin(rotation)
	if not attacking:
		move_and_slide()
	#Damages player or enemy if collides with them
	for i in range(get_slide_collision_count()):
		if not attacking:
			attacking = true
			var collision = get_slide_collision(i)
			print(collision)
			var collider = collision.get_collider()
			print(collider)
```

```
                        if collider.is_in_group("player") or collider.is_in_group("enemies"):
                                collider.take_damage(damage, damage_type)
                        $CollisionShape2D.disabled = true
                        $AnimatedSprite2D.play("impact")
                        await $AnimatedSprite2D.animation_finished
                        queue_free()
```

thunder_projectile.gd

```
extends CharacterBody2D

var damage: int = 1
var damage_type: String = "thunder"
var speed = 120
var attacking = false


func _physics_process(delta: float) -> void:
        #Gets movement vector
        velocity.x = speed * cos(rotation)
        velocity.y = speed * sin(rotation)
        if not attacking:
                move_and_slide()
        #Damages player or enemy if collides with them
        for i in range(get_slide_collision_count()):
                if not attacking:
                        attacking = true
                        var collision = get_slide_collision(i)
                        print(collision)
                        var collider = collision.get_collider()
                        print(collider)
                        if collider.is_in_group("player") or collider.is_in_group("enemies"):
                                collider.take_damage(damage, damage_type)
                        $CollisionShape2D.disabled = true
                        $AnimatedSprite2D.play("impact")
                        await $AnimatedSprite2D.animation_finished
                        queue_free()
```

ice_area.gd

```
extends Area2D

var damage: int = 1
var damage_type: String = "ice"


func _ready():
        await $AnimatedSprite2D.animation_finished
        queue_free()

func _on_body_entered(body: Node2D) -> void:
        if body.is_in_group("enemies") or body.is_in_group("player"): # If its an enemy or player
             damages it
                body.take_damage(damage, damage_type)
```

fire_area.gd

```
extends Area2D

var damage: int = 1
var damage_type: String = "fire"
var time:int = 1


func _ready():
        await get_tree().create_timer(time).timeout
        queue_free()

func _on_body_entered(body: Node2D) -> void:
        if body.is_in_group("enemies") or body.is_in_group("player"): # If its an enemy or player
             damages it
                body.take_damage(damage, damage_type)
```

thunder_area.gd

```
extends Area2D

var damage: int = 1
var damage_type: String = "thunder"


func _ready():
        await $AnimatedSprite2D.animation_finished
        queue_free()

func _on_body_entered(body: Node2D) -> void:
        if body.is_in_group("enemies") or body.is_in_group("player"): # If its an enemy or player damages it
                body.take_damage(damage, damage_type)
```

6.2.4 game/worlds

rooms/graph/dungeon_graph.gd

```
extends Node

class_name DungeonGraph

var root: DungeonGraphNode
var nodes: Array
var rooms: Dictionary = {'chest':["res://scenes/game/worlds/rooms/monster/room.tscn"],
'end':["res://scenes/game/worlds/rooms/end/end.tscn"],
'start':["res://scenes/game/worlds/rooms/start/start.tscn"],
'monster':["res://scenes/game/worlds/rooms/monster/m_room_1.tscn","res://scenes/game/worlds/rooms/monster/m_room_2.tscn","res://scenes/game/worlds/rooms/monster/m_room_3.tscn","res://scenes/game/worlds/rooms/monster/m_room_4.tscn","res://scenes/game/worlds/rooms/monster/m_room_5.tscn"],
'corridor':["res://scenes/game/worlds/rooms/corridors/corridor_1.tscn","res://scenes/game/worlds/rooms/corridors/corridor_2.tscn","res://scenes/game/worlds/rooms/corridors/corridor_3.tscn","res://scenes/game/worlds/rooms/corridors/corridor_4.tscn","res://scenes/game/worlds/rooms/corridors/corridor_5.tscn","res://scenes/game/worlds/rooms/corridors/corridor_6.tscn","res://scenes/game/worlds/rooms/corridors/corridor_7.tscn","res://scenes/game/worlds/rooms/corridors/corridor_8.tscn","res://scenes/game/worlds/rooms/corridors/corridor_9.tscn","res://scenes/game/worlds/rooms/corridors/corridor_10.tscn","res://scenes/game/worlds/rooms/corridors/corridor_11.tscn","res://scenes/game/worlds/rooms/corridors/corridor_12.tscn"],
'corridor_along':["res://scenes/game/worlds/rooms/corridors/corridor_1.tscn","res://scenes/game/worlds/rooms/corridors/corridor_7.tscn"],
'corridor_up':["res://scenes/game/worlds/rooms/corridors/corridor_2.tscn","res://scenes/game/worlds/rooms/corridors/corridor_8.tscn"],
'corridor_corner':["res://scenes/game/worlds/rooms/corridors/corridor_3.tscn","res://scenes/game/worlds/rooms/corridors/corridor_4.tscn","res://scenes/game/worlds/rooms/corridors/corridor_5.tscn","res://scenes/game/worlds/rooms/corridors/corridor_6.tscn","res://scenes/game/worlds/rooms/corridors/corridor_9.tscn","res://scenes/game/worlds/rooms/corridors/corridor_10.tscn","res://scenes/game/worlds/rooms/corridors/corridor_11.tscn","res://scenes/game/worlds/rooms/corridors/corridor_12.tscn"]}
var opposite_direction = {'north':'south', 'south':'north', 'east':'west', 'west':'east'}


func _init():
        root = DungeonGraphNode.new()
        root.room_type = "start"
        nodes.append(root)
#Adds node onto the node at the onto index of the type
func add_node(onto_index, direction, room_type):
        var onto = nodes[onto_index]
        if onto[direction]: # returns false if their is already a node in the place we are trying to place a new one
                return false
        var new_node = DungeonGraphNode.new()
        new_node.room_type = room_type
        #Sets up connections
        onto[direction] = new_node
        match direction:
                'north':
                        new_node.south = onto
                'south':
                        new_node.north = onto
                'east':
                        new_node.west = onto
                'west':
```

```
                        new_node.east = onto
        nodes.append(new_node)
        return true


func gen_room(node,previous_direction = null, previous = null):
        var room = load(rooms[node.room_type].pick_random()).instantiate() #instantiates random room of specified type
        add_child(room)
        if previous_direction: #sets psoition to match entrances
                room.set_pos(previous_direction,previous.get_pos(opposite_direction[previous_direction]))
        for i in ['north', 'south', 'east', 'west']: #Caps entrances with no neighbour
                if node[i] == null:
                        room.cap(i)
        return room

func gen_dungeon(node=root, previous_direction = null, previous = null, generated = []):
        var room = await gen_room(node,previous_direction,previous) #Generates room
        generated.append(node) #Adds to generated list
        for i in ['north', 'south', 'east', 'west']: #recursive call on any neighbour nodes that havent been generated
                if node[i] and node[i] not in generated:
                        generated = await gen_dungeon(node[i], opposite_direction[i], room, generated)
        return generated #returns generated to update list
```

rooms/graph/dungeon_graph_node.gd

```
extends Node

class_name DungeonGraphNode

var north: DungeonGraphNode = null
var south: DungeonGraphNode = null
var east: DungeonGraphNode = null
var west: DungeonGraphNode = null
var room_type: String
```

rooms/room.gd

```
extends Node2D

#Gets position of entrance in direction
func get_pos(direction):
        match direction:
                'north':
                        return $North.global_position
                'south':
                        return $South.global_position
                'east':
                        return $East.global_position
                'west':
                        return $West.global_position

#sets the position so that the entrance in the direction is at the position provided
func set_pos(direction, global_pos):
        match direction:
                'north':
                        global_position += global_pos - $North.global_position
                'south':
                        global_position += global_pos - $South.global_position
                'east':
                        global_position += global_pos - $East.global_position
                'west':
                        global_position += global_pos - $West.global_position

#adds a caps scene onto the direction specified
func cap(direction):
        match direction:
                'north':
                        var north_cap = load("res://scenes/game/worlds/rooms/north_cap.tscn").instantiate()
                        $North.add_child(north_cap)
                'south':
                        var south_cap = load("res://scenes/game/worlds/rooms/south_cap.tscn").instantiate()
```

```
35                      $South.add_child(south_cap)
36              'east':
37                      var east_cap = load("res://scenes/game/worlds/rooms/east_cap.tscn").
                            instantiate()
38                      $East.add_child(east_cap)
39              'west':
40                      var west_cap = load("res://scenes/game/worlds/rooms/west_cap.tscn").
                            instantiate()
41                      $West.add_child(west_cap)
42
43
44 func _ready():
45      if $Slimes: #Sets enemy health
46              for slime in $Slimes.get_children():
47                      slime.health = floor(2*Global.current_level*log(3*Global.difficulty))
48                      slime.damage = floor(Global.current_level*log(3*Global.difficulty))
49      if $Labels: #Updates Label text so levels match
50              for label in $Labels.get_children():
51                      label.text = label.text.replace("0", str(Global.current_level))
```

rooms/room_test.gd

```
1 extends Node2D
2
3
4 func _ready():
5       await dungeon_1()
6       self.scale = Vector2(0.5,0.5)
7       self.position = Vector2(1920/4,1080/4)
8       #var player = load("res://scenes/game/player/player.tscn").instantiate()
9       #player.position = Vector2(1920/2,1080/2)
10      #add_child(player)
11
12
13 func dungeon_1():
14      var graph = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate()
15      add_child(graph)
16      print(len(graph.nodes))
17      print(graph.add_node(0, 'north', 'corridor_along'))
18      print(graph.add_node(0, 'east', 'corridor_corner'))
19      print(graph.add_node(1, 'west', 'monster'))
20      print(graph.add_node(2, 'north', 'corridor_up'))
21      print(graph.add_node(4, 'north', 'monster'))
22      print(graph.add_node(2, 'east', 'monster'))
23      print(graph.add_node(0, 'south', 'corridor_corner'))
24      print(len(graph.nodes))
25      graph.add_node(7, 'south', 'monster')
26      graph.add_node(6, 'south', 'corridor')
27      graph.add_node(9, 'south', 'monster')
28      await graph.gen_dungeon()
29
30 func dungeon_2():
31      var graph  = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate
            ()
32      add_child(graph)
33      graph.add_node(0, 'north', 'corridor_corner')
34      graph.add_node(1, 'east', 'corridor_corner')
35      graph.add_node(2, 'east', 'corridor_corner')
36      graph.add_node(3, 'east', 'corridor_corner')
37      graph.add_node(1, 'north', 'monster')
38      graph.add_node(2, 'north', 'monster')
39      graph.add_node(2, 'south', 'monster')
40      graph.add_node(3, 'north', 'monster')
41      graph.add_node(3, 'south', 'monster')
42      graph.add_node(4, 'north', 'monster')
43      graph.add_node(4, 'south', 'monster')
44      graph.add_node(11, 'east', 'corridor_corner')
45      graph.add_node(12, 'south', 'monster')
46      graph.add_node(12, 'east', 'corridor_up')
47      graph.add_node(14, 'north', 'boss')
48      await graph.gen_dungeon()
49
50 func dungeon_3():
51      var graph  = load("res://scenes/game/worlds/rooms/graph/dungeon_graph.tscn").instantiate
            ()
52      add_child(graph)
53      graph.add_node(0, 'north', 'corridor_corner')
54      graph.add_node(0, 'east', 'corridor_corner')
```

```
        graph.add_node(0, 'south', 'corridor_corner')
        graph.add_node(0, 'west', 'corridor_corner')
        graph.add_node(1, 'north', 'monster')
        graph.add_node(2, 'east', 'monster')
        graph.add_node(3, 'south', 'monster')
        graph.add_node(4, 'west', 'monster')

        await graph.gen_dungeon()
```

chest.gd

```
extends StaticBody2D

@export var item_pool: Dictionary
@export var item_number: int
@export var key_path: String
@export var locked: bool
var item_list: Array



func _ready() -> void:
        for item in item_pool:
                var add_list = []
                for x in item_pool[item]:
                        add_list.append(item)
                item_list.append_array(add_list)

func _input(event):
        if event.is_action_pressed("interact"):
                for i in $Area2D.get_overlapping_bodies():
                        if i.name == "Player": # Check the player is within range
                                if not(locked) or Inventory.remove_item(key_path,1) is bool: # Check if the player has the key and removes it if the door isnt locked
                                        for j in range(item_number):
                                                Inventory.add_item(item_list[randi_range(0,len(item_list)-1)],1)
                                print(item_list)
                                queue_free()
```

door.gd

```
extends StaticBody2D

@export var change_scene: bool
@export var scene_path: String
@export var key_path: String
@export var locked: bool
@export var level: int

func _input(event: InputEvent) -> void:
        if event.is_action_pressed("interact"): # Check pressed button
                for x in $Area2D.get_overlapping_bodies():
                        if x.name == "Player": # Check the player is within range
                                if not(locked) or Inventory.remove_item(key_path,1) is bool: # Check if the player has the key and removes it if the door isnt locked
                                        if change_scene:
                                                if Database.get_save_data()[0]["level"] < level:
                                                        Database.set_level_value(level)
                                                get_tree().change_scene_to_file(scene_path) # Change Scene
                                        else:
                                                queue_free()
```

6.2.5 menu

login_form.gd

```
extends Node2D


```

```
func _on_sign_in_button_pressed() -> void:
	#Gets Inputs
	var username = $Input/UsernameInput.text
	var password = $Input/PasswordInput.text
	#Tries to login and handles errors
	var success = Database.login(username, password)
	if not (success is bool and success == true):
		if success == "InvalidUsernameError":
			$Labels/ErrorLabel.text = "Invalid Username"
		elif success == "IncorrectPasswordError":
			$Labels/ErrorLabel.text = "Incorrect Password"
	else: #Changes to save menu if succesfull
		get_tree().change_scene_to_file("res://scenes/menu/save_menu.tscn")

#Changes scene to reset password form
func _on_forgot_password_button_pressed() -> void:
	get_tree().change_scene_to_file("res://scenes/menu/reset_password_form.tscn")

#Changes scene to create acount form
func _on_create_account_button_pressed() -> void:
	get_tree().change_scene_to_file("res://scenes/menu/create_account_form.tscn")

#Quits
func _on_quit_button_pressed() -> void:
	Global.quit()
```

create_account_form.gd

```
extends Node2D




func _on_sign_up_button_pressed() -> void:
	#Get inputs
	var username = $Input/UsernameInput.text
	var password =  $Input/PasswordInput.text
	var answer = $Input/QuestionInput.text
	#Add's user if the passwords match checking for errors
	if $Input/PasswordInput.text == $Input/ConfirmPasswordInput.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:
			#Changes scene if user added
			get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
	else:
		$Labels/ErrorLabel.text = "Passwords Don't Match"

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

reset_password_form.gd

```
extends Node2D




func _on_reset_button_pressed() -> void:
	var username = $Input/UsernameInput.text
	var answer = $Input/QuestionInput.text
	var password = $Input/PasswordInput.text
	var confirm_password = $Input/ConfirmPasswordInput.text
	if password == confirm_password:
		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")
	else:
```

```
                $Labels/ErrorLabel.text = "Passwords Don't Match"


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

save_meu.gd

```
extends Node2D

#Function to load the list
func _load_list():
        #Deleting existing children
        for child in $SavesScroller/VBoxContainer.get_children():
                child.queue_free()
        var save_data_list = Database.get_user_save_data()
        for save_data in save_data_list: #Iterates through save data for the user creating a
            button for each one
                var button = Button.new()
                var hardcore
                if save_data["hardcore"] == 1:
                        hardcore = "True"
                else:
                        hardcore = "False"
                print(save_data)
                button.text = "Name: %s\nLevel: %s \t Difficulty: %s\nHardcore: %s" % [save_data[
                    "name"], save_data["level"], save_data["difficulty"], hardcore]
                button.connect("pressed",_on_save_selected.bind(save_data))
                $SavesScroller/VBoxContainer.add_child(button)

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
        _load_list()

#Sets the global difficulty and current save id before changing scene
func _on_save_selected(save_data):
        Database.current_save_id = save_data["save_id"]
        Global.difficulty = save_data["difficulty"]
        get_tree().change_scene_to_file("res://scenes/game/worlds/tutorial.tscn")

#Changes the difficulty label
func _on_difficulty_drag_ended(value_changed: bool) -> void:
        $Difficulty/DifficultyLabel.text = "Difficulty: " + str($Difficulty.value)

#Adds a new save with the parameters given
func _on_new_save_button_pressed() -> void:
        Database.add_new_save_data($Name.text,$Difficulty.value,$HardcoreButton.button_pressed)
        _load_list()


#Changes scene to login form
func _on_log_out_button_pressed() -> void:
        get_tree().change_scene_to_file("res://scenes/menu/login_form.tscn")
```

6.3 src

database.gd

```
extends Node

var db = SQLite.new()
var current_user_id: int
var current_save_id: int

#region Prepared Statements

var _create_table_users = """
CREATE TABLE IF NOT EXISTS users (
        user_id INTEGER PRIMARY KEY AUTOINCREMENT,
        username VARCHAR(15) UNIQUE NOT NULL,
        password VARCHAR(64) NOT NULL,
        salt VARCHAR(64) NOT NULL,
        answer VARCHAR(64) NOT NULL
);
```

```
"""

var _get_user_data = """
SELECT * FROM users
WHERE username = ?;
"""

var _add_new_user = """
--Assume hashed password and answer
INSERT INTO
users(username,password,answer,salt)
VALUES (?,?,?,?);
"""

var _delete_user = """
DELETE FROM users
WHERE username = ?;
"""

var _reset_password = """
--Assume hashed password and answer
UPDATE users
SET password = ?
WHERE username = ?
"""

var _create_table_save_data = """
CREATE TABLE IF NOT EXISTS save_data (
save_id INTEGER PRIMARY KEY AUTOINCREMENT,
user_id INTEGER,
name VARCHAR(32),
difficulty INTEGER,
hardcore INTEGER,
level INTEGER,
head VARCHAR(32),
chest VARCHAR(32),
legs VARCHAR(32),
weapon VARCHAR(32),
charm_1 VARCHAR(32),
charm_2 VARCHAR(32),
FOREIGN KEY(user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
"""

var _add_new_save_data = """
INSERT INTO
save_data(user_id,name,difficulty,hardcore,level)
VALUES (?,?,?,?,?);
"""

var _get_save_data = """
SELECT * FROM save_data
WHERE save_id = ?;
"""

var _get_user_save_data = """
SELECT name, level, hardcore, save_id, difficulty FROM save_data
WHERE user_id = ?;
"""

var _update_save_data = """
UPDATE save_data
SET
head = ?,
chest = ?,
legs = ?,
weapon = ?
charm_1 = ?,
charm_2 = ?,
level = ?
WHERE
user_id = ?
AND save_id = ?;
"""

var _create_table_stored_items = """
CREATE TABLE IF NOT EXISTS stored_items (
item_id INTEGER NOT NULL,
```

```
save_id INTEGER NOT NULL,
amount INTEGER NOT NULL,
PRIMARY KEY(item_id,save_id),
FOREIGN KEY(save_id) REFERENCES save_data(save_id) ON DELETE CASCADE
);
"""

var _update_stored_item_amount = """
UPDATE stored_items
SET amount = amount + ?
WHERE item_id = ?
AND save_id = ?;
"""

var _get_stored_items = """
SELECT * FROM stored_items
WHERE save_id = ?;
"""

var _get_stored_item_amount = """
SELECT amount FROM stored_items
WHERE save_id = ?
AND item_id = ?;
"""

var _add_stored_item = """
INSERT INTO
stored_items(save_id,item_id,amount)
VALUES (?,?,?);
"""

var _count_stored_items = """
SELECT COUNT(*)
FROM stored_items
WHERE save_id = ?;
"""

var _remove_stored_item = """
DELETE FROM stored_items
WHERE save_id = ?
AND item_id = ?;
"""

var _get_slot_values = """
SELECT * FROM save_data
WHERE save_id = ?;
"""

var _set_level_value = """
UPDATE save_data
SET level = ?
WHERE save_id = ?;
"""

var _set_head_value = """
UPDATE save_data
SET head = ?
WHERE save_id = ?;
"""

var _set_chest_value = """
UPDATE save_data
SET chest = ?
WHERE save_id = ?;
"""

var _set_legs_value = """
UPDATE save_data
SET legs = ?
WHERE save_id = ?;
"""

var _set_weapon_value = """
UPDATE save_data
SET weapon = ?
WHERE save_id = ?;
"""

```

```
var _set_charm_1_value = """
UPDATE save_data
SET charm_1 = ?
WHERE save_id = ?;
"""

var _set_charm_2_value = """
UPDATE save_data
SET charm_2 = ?
WHERE save_id = ?;
"""


#endregion



#test
func test() -> void:

	#Open or Create a database
	db.path = "res://game_data.db"
	db.open_db()
	db.drop_table("users")
	if not db.query(_create_table_users):
		print("Error: users table unable to be created")
		return



	if db.query_with_bindings(_add_new_user,["user","pass","ans","salt"]):
		print("success")

	db.query_with_bindings(_get_user_data,["user"])
	print(db.query_result)
	db.close_db()

#Function for generating salt
func gen_salt() -> String:
	var salt = "string"
	var x = randi_range(5,10)
	#Hashing a random amount of times with random salt each time
	for i in range(2**x):
		salt = j_hash(salt,str(i*randi_range(1,10)))
	return 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

#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 with hashed password and answer
		return "InvalidUsernameError" #If user cannot be added then the username must be invalid
	return true
#Function for deleting a user
func delete_user(username, password):
	db.query_with_bindings(_get_user_data,[username])
	if len(db.query_result) == 0: # If user doesnt exist
		return "InvalidUsernameError"
	var user_data = db.query_result[0]
```

```
246         var hashed_password = j_hash(password,user_data["salt"])
247         if hashed_password == user_data["password"]: # Checking password hash against stored hash
248                 db.query_with_bindings(_delete_user,[username]) # Deleting User
249                 return true
250         return "IncorrectPasswordError" # If password doesnt match
251 #Function for logging in
252 func login(username,password):
253         db.query_with_bindings(_get_user_data,[username]) # Getting user data
254         if len(db.query_result) == 0: # If user doesnt exist
255                 return "InvalidUsernameError"
256         var user_data = db.query_result[0]
257         var hashed_password = j_hash(password,user_data["salt"])
258         if hashed_password == user_data["password"]: # Checking password hash against stored hash
259                 current_user_id = user_data["user_id"]
260                 return true
261         return "IncorrectPasswordError" # If password doesnt match
262 #Function for resetting the password
263 func reset_password(username, answer, password):
264         db.query_with_bindings(_get_user_data,[username]) # Getting user data
265         if len(db.query_result) == 0: # If user doesnt exist
266                 return "InvalidUsernameError"
267         var user_data = db.query_result[0]
268         var hashed_answer = j_hash(answer,user_data["salt"])
269         var hashed_password = j_hash(password, user_data["salt"])
270         if hashed_answer == user_data["answer"]: # Checking the answer hash against the stored
              hash
271                 db.query_with_bindings(_reset_password,[hashed_password,username])
272                 return true
273         return "IncorrectAnswerError" # If answer doesnt match
274 #Function for counting the stored_items in the save
275 func count_stored_items():
276         db.query_with_bindings(_count_stored_items,[current_save_id])
277         return db.query_result[0]["COUNT(*)"]
278 #Function for getting the stored items in the save
279 func get_stored_items():
280         db.query_with_bindings(_get_stored_items,[current_save_id])
281         return db.query_result
282 #Function for getting the amount of a stored item in the save
283 func get_stored_item_amount(item_id):
284         db.query_with_bindings(_get_stored_item_amount,[current_save_id, item_id])
285         return db.query_result
286 #Function for updating the amount of a stored item in the save by adding an amount
287 func update_stored_item_amount(amount, item_id):
288         db.query_with_bindings(_update_stored_item_amount, [amount, item_id, current_save_id])
289         return db.query_result
290 #Function for adding a stored item into the save
291 func add_stored_item(item_id, amount):
292         db.query_with_bindings(_add_stored_item, [current_save_id, item_id, amount])
293         return db.query_result
294 #Function for removing a stored item from the save
295 func remove_stored_item(item_id):
296         db.query_with_bindings(_remove_stored_item, [current_save_id, item_id])
297         return db.query_result
298 #Function for getting a slot value from the save
299 func get_slot_value(slot):
300         db.query_with_bindings(_get_slot_values,[current_save_id])
301         if len(db.query_result) == 0:
302                 return null
303         return db.query_result[0][slot]
304 #Function for setting a slot value in the save
305 func set_slot_value(slot, item_id):
306         match slot:
307                 "head":
308                         db.query_with_bindings(_set_head_value, [item_id, current_save_id])
309                 "chest":
310                         db.query_with_bindings(_set_chest_value, [item_id, current_save_id])
311                 "legs":
312                         db.query_with_bindings(_set_legs_value, [item_id, current_save_id])
313                 "weapon":
314                         db.query_with_bindings(_set_weapon_value, [item_id, current_save_id])
315                 "charm_1":
316                         db.query_with_bindings(_set_charm_1_value, [item_id, current_save_id])
317                 "charm_2":
318                         db.query_with_bindings(_set_charm_2_value, [item_id, current_save_id])
319         return db.query_result
320 #Function for settig the level value
321 func set_level_value(n):
322         db.query_with_bindings(_set_level_value, [n])
```

```
323         return db.quer_result
324 #Function for getting the save data entries for the current user
325 func get_user_save_data():
326         db.query_with_bindings(_get_user_save_data,[current_user_id])
327         print(db.query_result)
328         return db.query_result
329 func get_save_data():
330         db.query_with_bindings(_get_save_data,[current_save_id])
331         return db.query_result
332 #Function to add a new save file for the current user
333 func add_new_save_data(name, difficulty, hardcore):
334         db.query_with_bindings(_add_new_save_data,[current_user_id, name, difficulty, hardcore,
              1])
335         return db.query_result
336
337
338
339
340 #Function for setting up the database
341 func setup(path):
342         db.path = path
343         db.open_db()
344         #Creating Tables if they do not exist
345         if not db.query(_create_table_users):
346                 print("Error: users table unable to be created")
347                 return
348
349         if not db.query(_create_table_save_data):
350                 print("Error: save_data table unable to be created")
351                 return
352
353         if not db.query(_create_table_stored_items):
354                 print("Error: stored_items table unable to be created")
355                 return
356
357         db.query("PRAGMA foreign_keys = ON;") # Activates foreign key
358
359 func _ready() -> void:
360         setup("res://game_data.db")
361
362
363 func _exit_tree() -> void:
364         db.close_db()
```

inventory.gd

```
1  extends Node
2
3  var max_inventory_size = 100
4
5
6  func item_amount(item_id: String):
7          var query = Database.get_stored_item_amount(item_id)
8          if len(query) == 0:
9                  return 0
10         return query[0]["amount"]
11
12
13
14 func add_item (item_id, amount):
15         if amount == 0:
16                 return true
17         var item = load(item_id) # Loads the item
18         if Database.count_stored_items() >= max_inventory_size and not(item.stackable and
               item_amount(item_id) != 0): # Full Inventory
19                 return "FullInventoryError"
20         else :
21                 # Checks if you can stack the item and either adds a new entry or stacks it
22                 if item_amount(item_id) != 0 and item.stackable: # If the item is in the database
23                         Database.update_stored_item_amount(amount, item_id)
24                 else:
25                         Database.add_stored_item(item_id, 1)
26                         add_item(item_id,amount-1)
27                 return true
28
29
30 func remove_item(item_id, amount):
31         var amount_stored = item_amount(item_id)
```

```
        if amount_stored != 0: # If the item is in the database
                if amount_stored < amount: # Not enough items
                        return "ItemQuantityError"
                elif amount_stored == amount: # Exactly enough items
                        Database.remove_stored_item(item_id)
                else :
                        Database.update_stored_item_amount(-amount, item_id) # Removes the ammount of that item
                return true # Indicates that it was successful
        return "ItemQuantityError"


func unequip_item(slot):
        # Gets the slot value
        var value = Database.get_slot_value(slot)
        # Checks if there is an item to unequip
        if value != null:
                var success = add_item(value, 1)
                if (not(success is bool) and success == "FullInventoryError"): # Adds the item back to the stored_items/checks if the inventory is full
                        return "FullInventoryError"
                Database.set_slot_value(slot, null) # Sets the slot back to null
                return true
        return true # If there is no item to unequip we have succeeded


func equip_item(item_id):
        var slot: String
        var item = load(item_id) # Loads the item
        # Checks if the item is Equipable and in the inventory
        if not(item is Equipable) and item_amount(item_id) >= 1:
                return false
        # Gets the slot to equip it into
        if item is Armour:
                slot = item.body_part
        elif item is Weapon:
                slot = "weapon"
        else:
                if Database.get_slot_value("charm_1") == null:
                        slot = "charm_1"
                else:
                        slot = "charm_2"
        remove_item(item_id,1)
        var success = unequip_item(slot) # Checks the success of unequipping the item
        if not(success is bool) and success == "FullInventoryError":
                add_item(item_id,1)
                return "FullInventoryError"
        Database.set_slot_value(slot, item_id) # Equips the item
        return true
```

global.gd

```
extends Node
var difficulty: int = 1
var current_level: int = 1

func await_call(n:int,f:Callable):
        await get_tree().create_timer(n).timeout
        f.call(1)

func quit():
        get_tree().quit()
```

6.4 utils

dummy.gd

```
extends StaticBody2D
var damage_taken = 0

func _ready() -> void:
        add_to_group("enemies")

func take_damage(damage, damage_type):
        print("AAAAAAAAAAA")
```

```
        print(damage)
        print(damage_type)
        damage_taken -= damage
        print(damage_taken)
```

test.gd

```
extends Node2D


var db = preload("res://src/database.gd").new()

func _ready() -> void:
        if Engine.is_editor_hint():
                print("yay")
        print(Engine.get_singleton_list())
        var path = "res://utils/test.db"
        var dir  = DirAccess.open("res://utils")

        if dir.file_exists("test.db"):
                var err = dir.remove("test.db")

        db.setup(path)
        test_database()
        db.db.close_db()
        test_inventory()
        Database.db.close_db()

func test_database():
        #Criteria 6.
        print("6.1:")
        print((len(db.gen_salt()) == 64) and (db.gen_salt() != db.gen_salt())) # Test 6.1.1
        print(len(db.j_hash("password", "salt")) == 64) # Test 6.1.2
        print(db.j_hash("password", "salt") == db.j_hash("password", "salt")) # Test 6.1.3
        print(db.j_hash("password", "salt") != db.j_hash("Password", "salt")) # Test 6.1.4

        print("6.3")
        print(db.add_user("Hyrule", "Password", "Answer")) # Test 6.3.1
        print(db.add_user("Hyrule", "Password", "Answer") == "InvalidUsernameError") # Test 6.3.2

        print("6.7,6.8,6.10")
        print(db.login("Hyru1e", "Password") == "InvalidUsernameError") # Test 6.7.1
        print(db.login("Hyrule", "Password")) # Test 6.7.2
        print(db.reset_password("Hyru1e", "Answer", "password") == "InvalidUsernameError") # Test
             6.8.1
        print(db.reset_password("Hyrule", "answer", "password") == "IncorrectAnswerError") # Test
             6.8.2
        print(db.reset_password("Hyrule", "Answer", "password")) # Test 6.8.3
        print(db.login("Hyrule", "Password") == "IncorrectPasswordError") # Test 6.7.3

        print(db.delete_user("Hyrule", "Password") == "IncorrectPasswordError") # Test 6.10.1
        print(db.delete_user("Hyrule", "password")) # Test 6.10.2
        print(db.reset_password("Hyrule", "Answer", "password") == "InvalidUsernameError") # Test
             6.7.4


func test_inventory():
        Database.add_user("test", "password", "answer")
        Database.login("test", "password")
        Database.add_new_save_data(1,true)
        Database.current_save_id = Database.get_user_save_data()[0]["save_id"]
        Inventory.max_inventory_size = 1
        #Criteria 12
        print(Inventory.add_item("res://utils/test_item.tres", 2)) # Test 12.5.1
        print(Inventory.item_amount("res://utils/test_item.tres") == 2)
        print(Inventory.add_item("res://utils/test_item.tres", 3)) # Test 12.5.2
        print(Inventory.add_item("res://utils/test_weapon.tres", 1) == "FullInventoryError") #
            Test 12.5.3
        print(Inventory.item_amount("res://utils/test_item.tres") == 5)
        print(Inventory.remove_item("res://utils/test_item.tres",2)) # Test 12.5.4
        print(Inventory.item_amount("res://utils/test_item.tres") == 3)
        print(Inventory.remove_item("res://utils/test_item.tres",10) == "ItemQuantityError") #
            Test 12.5.5
        print(Inventory.remove_item("res://utils/test_item.tres",3)) # Test 12.5.6
        print(Inventory.item_amount("res://utils/test_item.tres") == 0)
        print(Inventory.remove_item("res://utils/test_item.tres",2) == "ItemQuantityError") #
            Test 12.5.7
        print(Inventory.unequip_item("head")) # Test 12.4.1
```

```
66          Inventory.add_item("res://utils/test_helmet.tres",1) #Test 12.4.2
67          print(Inventory.equip_item("res://utils/test_helmet.tres"))
68          print(Database.get_slot_value("head") == "res://utils/test_helmet.tres")
69          print(Inventory.item_amount("res://utils/test_helmet.tres") == 0)
70          Inventory.add_item("res://utils/test_helmet2.tres",1) # Test 12.4.3
71          print(Inventory.equip_item("res://utils/test_helmet2.tres"))
72          print(Inventory.item_amount("res://utils/test_helmet.tres") == 1)
73          print(Inventory.unequip_item("head") == "FullInventoryError") # Test 12.4.4
74          Inventory.remove_item("res://utils/test_helmet.tres",1) # Test 12.4.5
75          print(Inventory.unequip_item("head"))
76
77
78          Database.delete_user("test", "password")
```