[Table of Figures 2](#_Toc183454204)

[Abstract 4](#_Toc183454205)

[1.0 Introduction 4](#_Toc183454206)

[*1.1 Existing Systems* 4](#_Toc183454207)

[*1.2.1 Proposed System* 5](#_Toc183454208)

[*1.3 Software Engineering Model* 6](#_Toc183454209)

[*1.4 Purpose* 6](#_Toc183454210)

[*1.5 Project Objective and Scope* 7](#_Toc183454211)

[*1.6 Technologies & Tools* 7](#_Toc183454212)

[*1.7 Users* 7](#_Toc183454213)

[2.0 Analysis of the System 7](#_Toc183454214)

[*2.1 Activity List* 7](#_Toc183454215)

[*2.2 Context Diagram* 8](#_Toc183454216)

[*2.3 Data Flow Diagram* 9](#_Toc183454217)

[3.0 Database Design 10](#_Toc183454218)

[*3.1 Player Data* 10](#_Toc183454219)

[*3.2 Save & Load Functionality* 11](#_Toc183454220)

[*3.2.1 Save Functionality* 11](#_Toc183454221)

[4.0 Functionality and Implementation 14](#_Toc183454222)

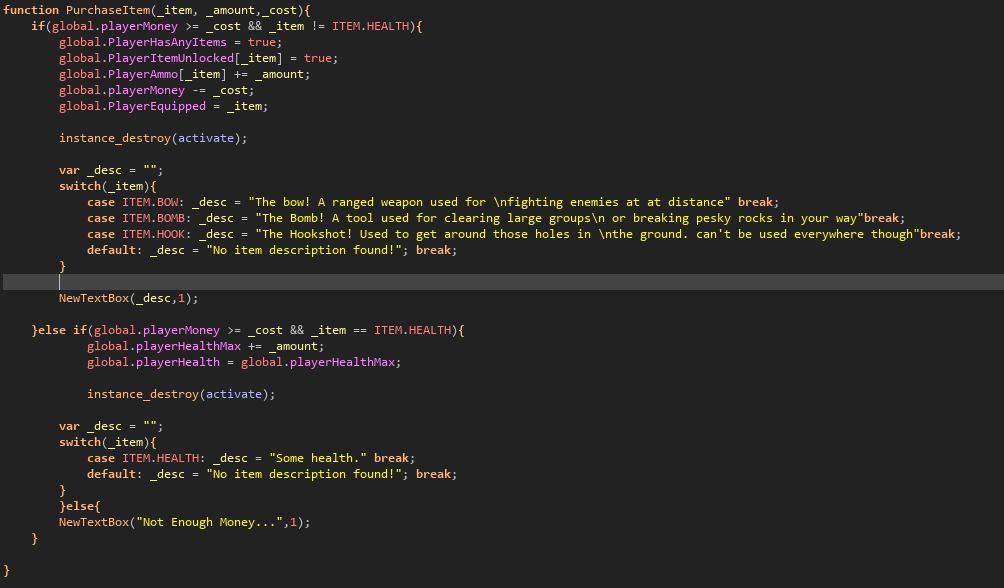
[*4.1 Features* 14](#_Toc183454223)

[*4.1.1 HurtEnemy function* 15](#_Toc183454224)

[*4.1.2 Slime Create Function* 16](#_Toc183454225)

[*4.1.3 KingSlime Create Function* 16](#_Toc183454226)

[*4.1.4 Dialogue Responses function* 17](#_Toc183454227)

[*4.1.5 Purchase Item* 19](#_Toc183454228)

[5.0 Testing 19](#_Toc183454229)

[6.0 Conclusion 20](#_Toc183454230)

[*6.1 Limitations* 20](#_Toc183454231)

[*6.2 Future Enhancements/Recommendations* 21](#_Toc183454232)

[7.0 References 22](#_Toc183454233)

# Table of Figures

Figure 1: Context Diagram ………………………………………………………………………9

Figure 2: Data Flow Diagram……………………………………………………………………10

Figure 3: Player Data…………………………………………………………………………….11

Figure 4: Save Functionality…………………………………………………………………….12

Figure 5: Load Functionality…………………………………………………………………….14

Figure 6: Hurt Enemy Function…………………………………………….……………………16

Figure 7: Slime Create Function…………………………………………………………………17

Figure 8: KingSlime Create Function……………………………………………………………17

Figure 9: Dialog Responses Function……………………………………………………………18

Figure 10: Activate Hat Cat Function……………………………………………………………19

Figure 11: Purchase Item………………………………………………………………………...20

# Abstract

Our team built an action role-playing game based on the enjoyment of those who like to play games. Users would play as the main character who has come from a distant planet to eliminate some enemies lurking within the local cave near the town. As this was a single player game that is saved locally, we have excluded the admin feature from our project. Our goal was to create a fun and enjoyable experience for those who play our game by providing them with enemies and small puzzles to do as they make their way towards the final boss. This game was created on Gamemaker Studio 2 using the Gamemaker Language as its main language on both front and backend.

# 1.0 Introduction

## *1.1 Existing Systems*

* The Binding of Isaac (<https://bindingofisaac.fandom.com/wiki/The_Binding_of_Isaac_Wiki>)
* A roguelike video game that tells the story of the playable character Isaac as he fights through floors of enemies. The story loosely follows the biblical story of “the Binding of Isaac '' and how the main character is being sacrificed to a higher being. The enemies and item drop in this game are all randomized as the roguelike genre entails.
* Hollow Knight (<https://www.hollowknight.com/>)
  + A game where the player controls the knight in battles against enemies. The player is able to collect/learn new abilities as they progress through the levels.
  + Genshin Impact (<https://genshin.hoyoverse.com/en/home>)
* Genshin Impact is a mobile/PC game that combines the role-playing genre with the gacha functionality to create a new experience for players. Its incentives the player to continue playing to build up points to be able to obtain new characters/weapons through a lottery like system
* The Legend of Zelda
* A very popular role-playing game created in the 1980s by Nintendo. It is mainly a role-playing game that tells the story of a hero in a fantasy world that attempts to save the princess. The main focus of this game is to fight monsters and solve puzzles to complete the overall quest of saving the princess.

### *1.2.1 Proposed System*

Our game would be a pc-based game where the player would go through levels to defeat the monsters that have started to roam around the world they have landed on. After entering the dungeon, the dungeon and enemies would be randomized and the loot dropped would also be randomized. Ultimately, the goal would be to escape the dungeon that increases in difficulty with every floor that the player completes. We would also be giving the player different weapons with different effects (either an upgrade in attack or defense, or a different ranged weapon).

*1.2.2 Changes to the Proposed System* Due to our inexperience, we decided to exclude the randomized dungeon. Although the functionality works, getting the player to spawn in properly was an issue and was something we could not overlook in our final draft. Our lack of time also prevented us from creating and adding more sprites/npc/weapons because of the amount of time that it would take to create and animate these features. Instead, we created a static dungeon with a final boss feature at the end of it that would drop the quest item asked from the NPC at the start of the world.

## *1.3 Software Engineering Model*

*Spiral Model:* The software lifecycle model we use is the spiral model. The spiral model has a repetitive approach. It moves in a circular manner where the project passes through four phases over and over again, in the form of a spiral, until we finish completing the game. By doing this, it gives us several opportunities to improve our game, look for bugs/issues, and evaluate what we did right and wrong. This ensures that we can encounter and resolve any problems before going onto the next step of the project.

The four phases of the spiral are:

1) Determining objectives: which involve designing and planning.

2) Identifying risks.

3) Development and testing: by playing the game and looking for improvements or bugs.

4) Planning the next iteration: which is to understand what we did right, what went wrong, and then use that information to refine the game while repeating the cycle and starting the cycle over again with new objectives for the next part.

The advantage of the spiral model is that it is a flexible, low risk model which allows us to easily modify, test and ensure that parts are satisfactory before continuing on with the rest of the project.

## *1.4 Purpose*

The purpose of our game was to create a game with a story that would be both entertaining and interactive. We wanted the player to have fun with the world that we created and want to adventure farther into the dungeon to see how many times they can go in and just what they could find inside.

## *1.5 Project Objective and Scope*

Our main goal was to keep the player captivated with the randomization and features of our game. There are many little secrets within the game like what the player could collect from breaking a rock and how many slimes could the player defeat before they are finally defeated themselves. The eventual play was to be able to create a system where the player was able to play as different characters. Instead, there is one main character that can wield many different weapons in order to defeat the boss inside. The game was made to be played on a PC or laptop as well.

## *1.6 Technologies & Tools*

We used the Gamemaker IDE as our main programming IDE and testing center as there is also a built-in debugging tool. We also mainly used Discord for our main form of communication and Gmail as a secondary form. In order to share files back and forth, we used Google Drive to give each other the most updated version of our project. In order to create our DFD and Context diagrams, we used Lucidchart. In order to find solutions on part that we were stuck on, we relied on Reddit.com, Yoyogames.com, and Youtube.com

## *1.7 Users*

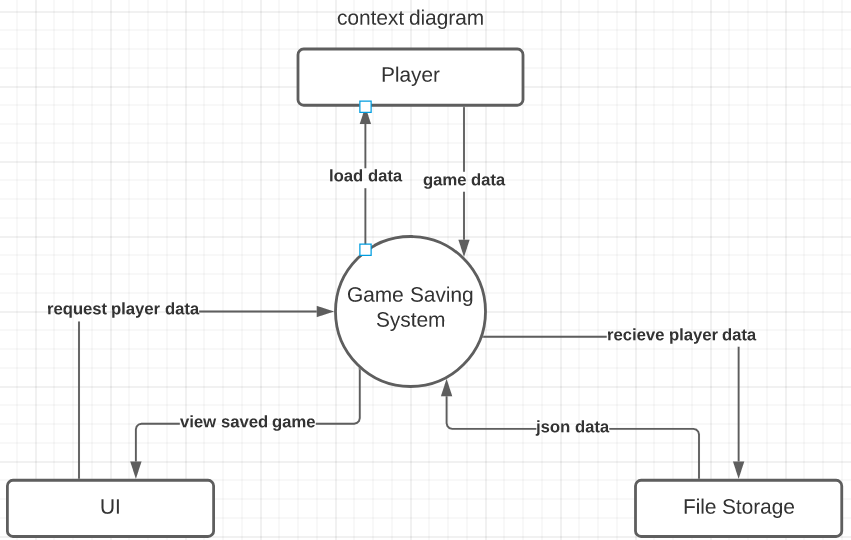
Our game is a single player rpg, therefore the only user would be the player.

# 2.0 Analysis of the System

## *2.1 Activity List*

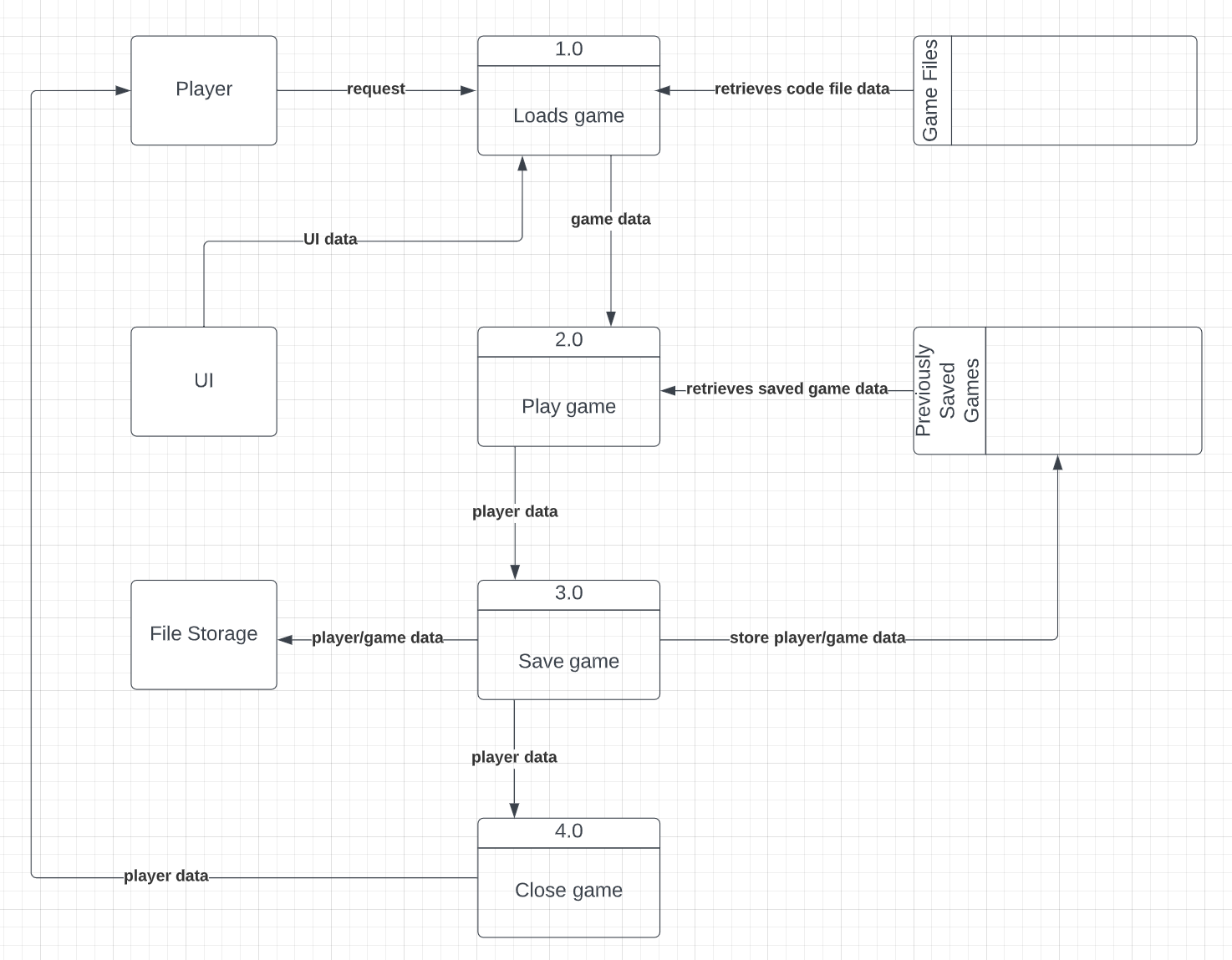
1. The player opens the game and navigates to the save files
2. If there is a save file, then the player can continue their game from their last save point. If not, they can open a new save file.
3. The player can activate the quest given by the NPC and navigate to the dungeon
4. After entering the dungeon, the player can defeat enemies and make their way through the gaps
5. The player defeats the enemy boss at the end of the dungeon and makes their way back to the NPC with the quest related item drop that only the boss enemy drop
6. The player returns the item to the NPC and the quest is completed

## *2.2 Context Diagram*

  
Figure 1: Context Diagram

Our saving system flows data between the player, UI, and file storage. The flow of data is initiated from the player. When a player saves a game, that data is then sent to our game saving system which is then received by the player’s file storage. In the instance when a player wants to go back to a saved game, the kson data is loaded in from the file storage and the UI gets this data and displays the saved game slots to the player. Finally, the player selects and loads the game data and creates a session.

## *2.3 Data Flow Diagram*

  
Figure 2: Data Flow Diagram

Our data flow diagram shows how player and game data are transferred in each process. When the player opens the game application and the game loads, it will take UI data and data from its game files to open to the title page of the game. From there, players can continue their progress and open any previous games that were saved or play an entirely new game from the beginning. Once the player spends some time playing it and accumulates some game data, they can save their progress in one of the saved game slots which will also update some game files. After that, they can close the game.

# 3.0 Database Design

## *3.1 Player Data*

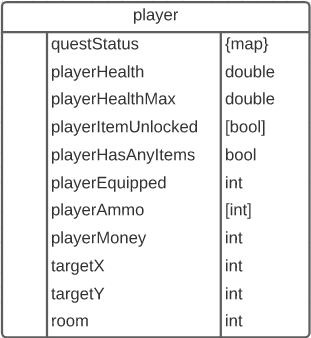


Figure 3: Player Data

The player data contains data that includes:

playerHealth: The player’s health.

playerItemUnlocked: What weapons and items the player has in their inventory.

playerAmmo: The amount of ammunition the player has in each of their weapons.

playerHasAnyItems: Whether a player has an item equipped or not.

playerMoney: The amount of money the player has.

targetY: The Y position of the player.

targetX: The X position of the player.

room: The room the player is in.

playerEquipped: Which item or weapon does the player has.

questStatus: The quest name and its completion.

An example of how a player’s data would look like in JSON format.

{

"playerHealth": 5.0,

"playerItemUnlocked": [ 0.0, 1.0, 1.0, 1.0, 0.0 ],

"playerAmmo": [ -1.0, 9.0, 14.0, -1.0, -1.0 ],

"playerHasAnyItems": 1.0,

"playerMoney": 4.0,

"targetY": 112.0,

"room": 9.0,

"playerEquipped": 3.0,

"targetX": 296.0,

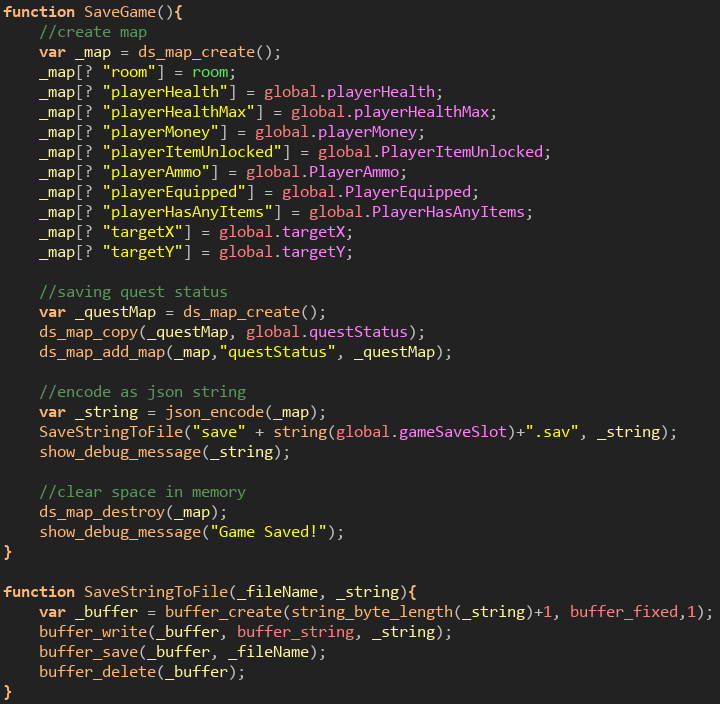
"questStatus”: "TheHatQuest": 0.0 },

"playerHealthMax": 3.0

}

## *3.2 Save & Load Functionality*

## *3.2.1 Save Functionality*

  
Figure 4: Save Functionality

The saving system works by using the ds\_map data structure which is used to hold key-value pairs. Most player data only contain numeric values such as ‘playerHealth’, ‘playerMoney’, and ‘playerAmmo’. Although, there is other data that needs to be nested in other data structures such as arrays and even other ds\_map. For instance, the ‘playerItemUnlocked’ key and the ‘playerAmmo’ key holds arrays while the ‘questStatus’ key stores another map that contains the player’s quest completions. We could not directly reference the questMap global variable, otherwise, when the instance of the saved game data was destroyed to save memory, any other references to that object would be destroyed as well. This feature is more of a design choice in GameMaker because it is important to de-reference all data structures that are not being used to avoid any memory leaks. So, what we did was create a temporary map and then copy over the map data from the global ‘questStatus’ variable to the newly created temp map called \_questMap’. After that, the \_questMap’ data was assigned to a key-value pair in the saved game data. The map is then encoded as a json string so it could be written on a file.

*3.2.2 Load Functionality*



Figure 5: Load Functionality

For the loading functionality, it performs the reverse steps of the saving function. We first look for the file and see if it exists. If it does, then we are going to load the json from the file. After that it is just assigning the file data to our global variables. A small caveat when decoding the kson is that it returns maps but also lists. We need to convert those into their original structure which are arrays. We must loop through each element in the list and copy it to the arrays. The same goes for ‘questStatus’ data. After that the player is then transitioned into the room they previously saved from.

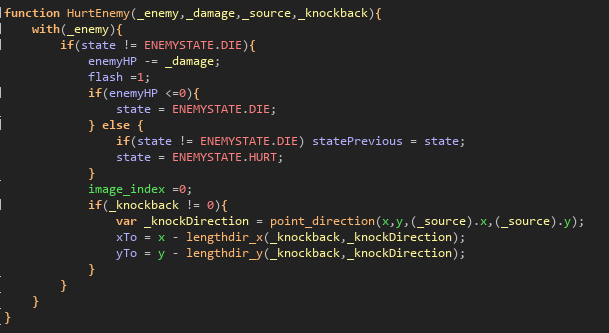
# 4.0 Functionality and Implementation

## *4.1 Features*

Saving: a feature that we thought would be the most important in our game. The play consists of constantly collecting gold, defeating enemies, losing/gaining health, or buying items from the shop. This was especially important if the player wanted to fight the boss multiple times and save right before the boss’s room to shorten the trip. It was also a design choice to add a shop inside the dungeon, as in some games there is a mysterious trader that just so happened to set up shop before the boss’s lair.

Combat/Mob drops: this feature to us was the most important as this is where most of the enjoyment of the game comes from. This was one of the more difficult features to program into the game as well as there was a lot of collision that had to be calculated between the enemy and the player’s weapon. Below there is a clip of code used in reference for the combat of the game. The enemy is dealt with a fixed amount of damage from the player depending on the weapon used. Bombs carry the most damage with the arrows being the second most powerful and the standard melee attack being the third strongest, and the grappling hook doing no damage. The way that we have programmed the sprites in the game, a different animation is a different state. For enemies, there is an idle, wander/chase, attack, and dying state. So long as the enemy state is not in its DIE state, the enemy can continue to do and receive damage. There is also a knockback that happens between the player and the enemy. This must happen or else the player and the enemy would simply clash into each other until the other dies. For the boss, the disparity is most notable because the boss can displace the player much farther than the player to it.

### *4.1.1 HurtEnemy function*

  
 Figure 6: Hurt Enemy Function

We also wanted to be able to randomize and limit the items the enemies and objects around the world were able to drop. If the player was starting out in the world with 0 gold and weapons, we did not want to break a pot to result in the pot dropping some arrows. Therefore, we made it so that as long as the player does not have a certain weapon, the ammo for that weapon would not be dropped. We also wanted it so that the enemies could drop varying different items. Whereas with the boss of the game, we wanted it so that it would drop multiple different items including the quest item.

### *4.1.2 Slime Create Function*

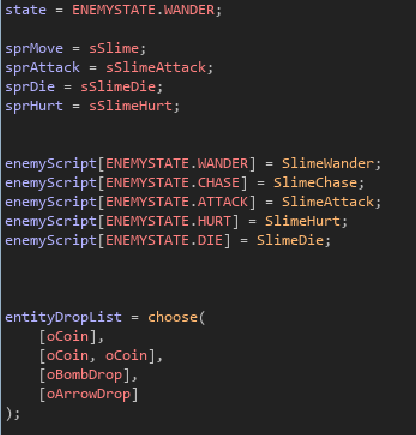
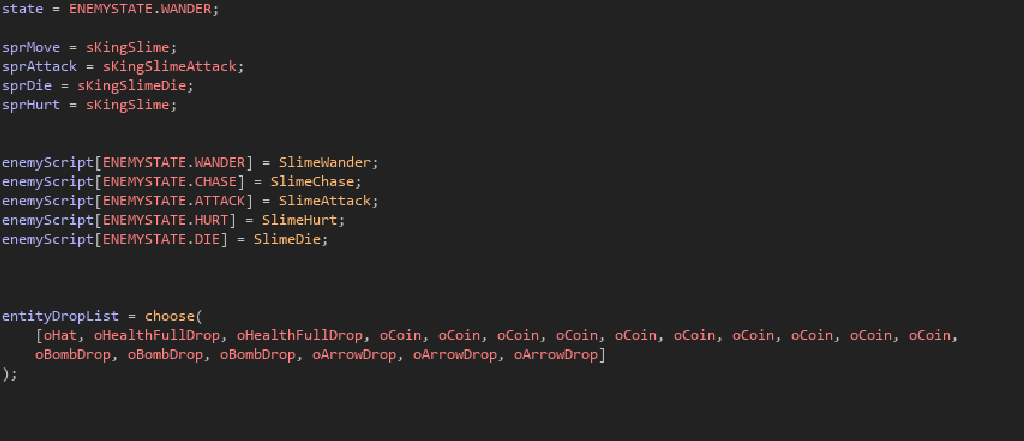


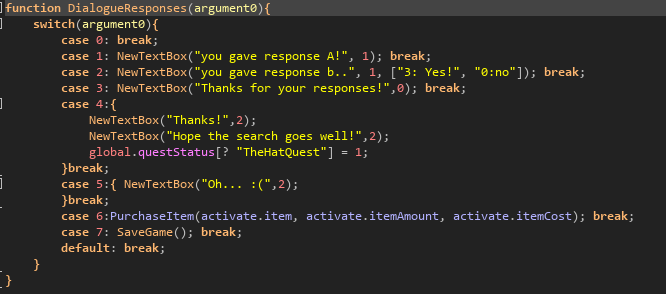
Figure 7: Slime Create Function

### *4.1.3 KingSlime Create Function*

  
 Figure 8: KingSlime Create Function

Dialogue: This was especially important for our game as this is a way that the player can interact with signs and NPCs. The simplest way that we found to create dialogue was using switch functions that would change based on the NPC, interaction, and response.

### *4.1.4 Dialogue Responses function*

  
 Figure 9: Dialog Responses Function

Quests: Like the dialogue, the quest functionality is also set up with switch statements. We believe that giving the player a motive or quest will keep them engaged and want to put more time into the game. The Hat was created as an object that the play could physically interact with and carry around the dungeon and village area. Once the hat was returned to the NPC, the NPC’s sprite would change to the sprite with a hat and would stay like that permanently for the rest of the game as it is dialogue would also stay at “Thanks once again” after completing the quest  
 *4.1.5 Activate Hat Cat Function*



Figure 10: Activate Hat Cat Function

Shop: we decided that to collect the items needed for the overall quest, that the player would be able to purchase it from the shop. We set up a system where the items would be laid out in front of the player and the player would be able to interact with it and receive a small description, once again using switch statements. The shop will reject your request to purchase the item if the player also does not have enough money. This is also where we came up with the idea of adding a potion that would increase the max health of the player. If the player comes in and purchases the health potion at full health, the potion increases the max health. If the player is missing some health, the health potion can return some health.

### *4.1.5 Purchase Item*

Figure 11: Purchase Item

# 5.0 Testing

As per the spiral model, whenever a new functionality like rolling or breaking grass came up all the members would go in a test the game before we were allowed to move on. This was so that we could all attempt to “break” the game and produce fixes before it became a bigger issue later. This was especially helpful when implementing the dialogue feature. As for some members, there would be an additional repeated message towards the end and for others there would not be any. We had to make sure that it was playable on all different computers and play styles before moving on.

Our focus was to make the game enjoyable for all people, so we ourselves play tested the game to make sure that it was feasible and just at the right level of difficulty. One issue that we ran into was just how many enemies existed on the floor and whether or not it was possible to defeat them all both entering and leaving a room. As well as the difficulty of the final boss. Some of our members were able to defeat the boss while others struggled. All members were able to beat the boss at least twice and we decided that at that point it was at an acceptable level of difficulty.

Lastly, there would be times when we needed to make sure certain functions were working properly that did not require us to play through the entire game and collect coins. So, there would be times where we would give ourselves money and large amounts of health so we could test those functions.

# 6.0 Conclusion

## *6.1 Limitations*

Although Gamemaker Language is very similar to Java, there are a lot of functionalities that we now have to get used to as a team. For everyone, this was an entirely new experience working on both a graphic frontend and functionally challenging backend. Both were very heavily intertwined with each other, which made it difficult to make big changes once something was complete.

Although the functionality of the randomized dungeon is still encoded in the game currently, it is not being called because the player cannot properly spawn within the dungeon which meant that we could not get the enemies to spawn in properly either. The way that the player spawns into any room, the player object is hardcoded to spawn in a specific area and the new instance of the player takes over. Because the dungeon is randomized, we cannot hardcode the player’s spawn point, as we would run the risk of spawning in a wall or out of bounds. We attempted to find a way around that issue but as time ran out, we made the decision to simply leave this aspect of the game behind.

Another limitation we had was a lack of sprites and animations. Although a lot of what we have in the game currently are assets we have borrowed to create the background and foreground, we wanted to have at least the main player to be our own creation. There are 14 sprites for the main character used for different actions such as running, lifting, dying, etc. each with a maximum of 16 frames per animation. The main character took longer than we expected to animate and therefore we had to drop the idea of creating other playable characters with different functionalities.

Lastly, our database was much different than what we expected to achieve by the end of the project. As stated, we do have a local database running for the use of the game, it is created and there. Yet connecting that database to Gamemaker was more difficult than we expected. After doing research, most people said that it was simpler to save our game file locally than on a database server. The way the information is saved as an array with different data type values, it would have been harder on us to separate that data and then recombine it when loading and saving, therefore we made the decision to go with what was recommended.

## *6.2 Future Enhancements/Recommendations*

One definite feature that we should add into the game is an option to be able to delete a previous save. While testing, we would have to go back and start from the beginning at times to assess out certain options. This would result in us using all our save files and then having to go into our computer’s files and delete our previous saves manually. This would be an option for us and for players who enjoy being able to play the game as many times as they want.

Another item would be to reimplement the features that we had to leave out like the randomized dungeon and the gacha system. The gacha system was left behind because of a lack of time in animating new player spirits, and therefore would not be as difficult to implement. The randomized dungeon on the other hand was an issue that we were having all around. It is not a matter of whether it is possible, it is a matter of our lack of experience with Gamemaker that limited us. Once we figure out what the actual problem was with the randomized dungeons' spawn points, we would be able to implement the enemies, boss, and the room transitions as well.

Lastly, we would love to implement more gameplay and expand on the map and overall world of the game. This would include more characters/npc, more weapons to use, and more enemies to defeat

# 7.0 References

“The Binding of Isaac Wiki.” *Fandom*, https://bindingofisaac.fandom.com/wiki/The\_Binding\_of\_Isaac\_Wiki.

*Genshin Impact – Step into a Vast Magical World of Adventure*, https://genshin.hoyoverse.com/en/home.

*Hollow Knight*, https://www.hollowknight.com/.

Spalding, Shawn. “RPG ASSETS FOLDER.” *Google*, Google, https://drive.google.com/drive/folders/19X1JLO7MNGV-z8XL1qlVAeSin01Z4v6R?usp=sharing.