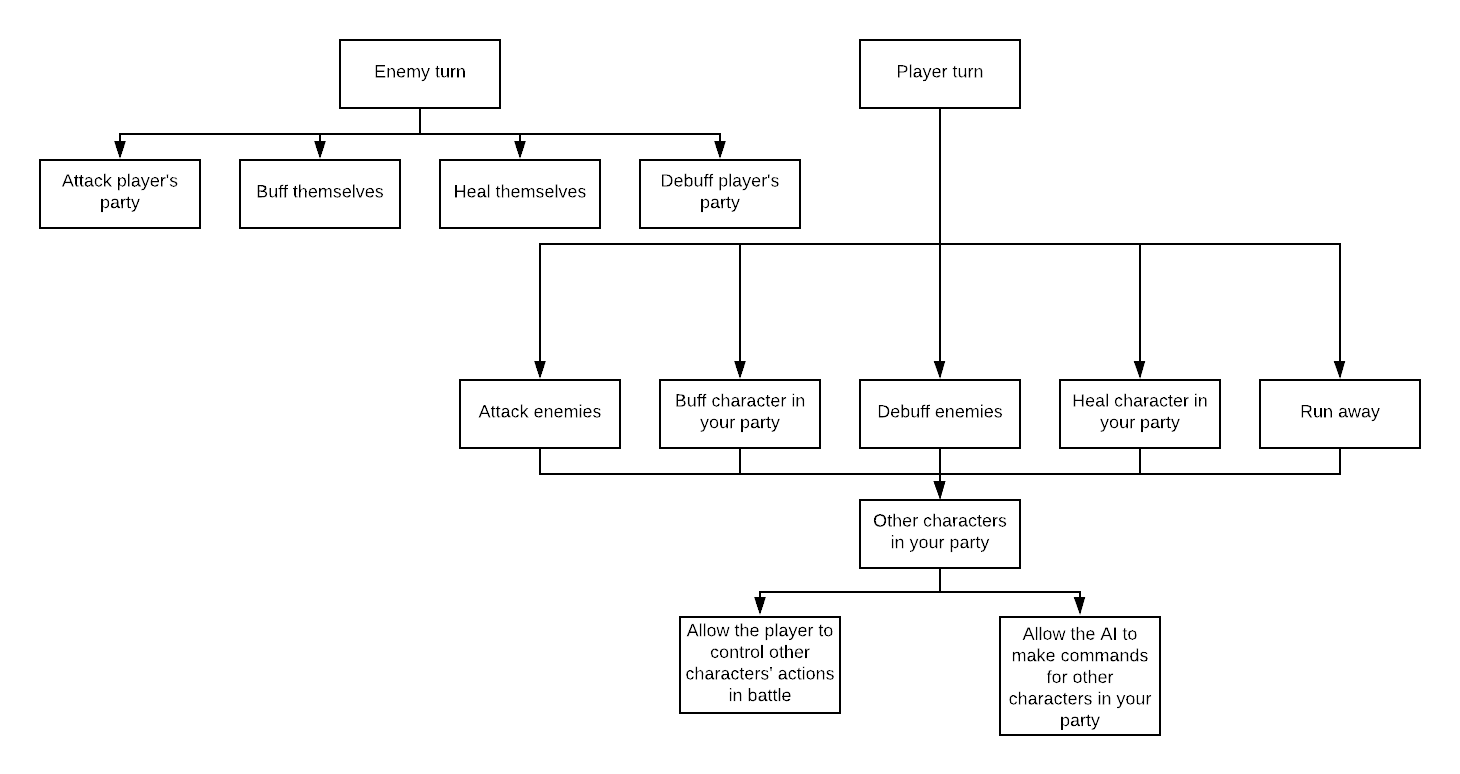
**Design of Solution**



**Decompose the Problem**

The full game is being split into ‘New Game’ and ‘Continue Game’. This is because the option you choose determines how the program finds the values for the variables, and these are the 2 main menu options that you get when starting up the program. If ‘New Game’ is chosen at the main menu, the variables are reset to their default value, starting the game from the very beginning. If ‘Continue Game’ is chosen and you already have a save file, the program will find the values of the variables when you last saved to that file from a separate database. It also shows that any dialogue in the opening cutscene, and any cutscenes during the game, will be shown on the python shell to make it much easier to program by just printing it to the screen, rather than trying to print it directly to the game window while you are playing.

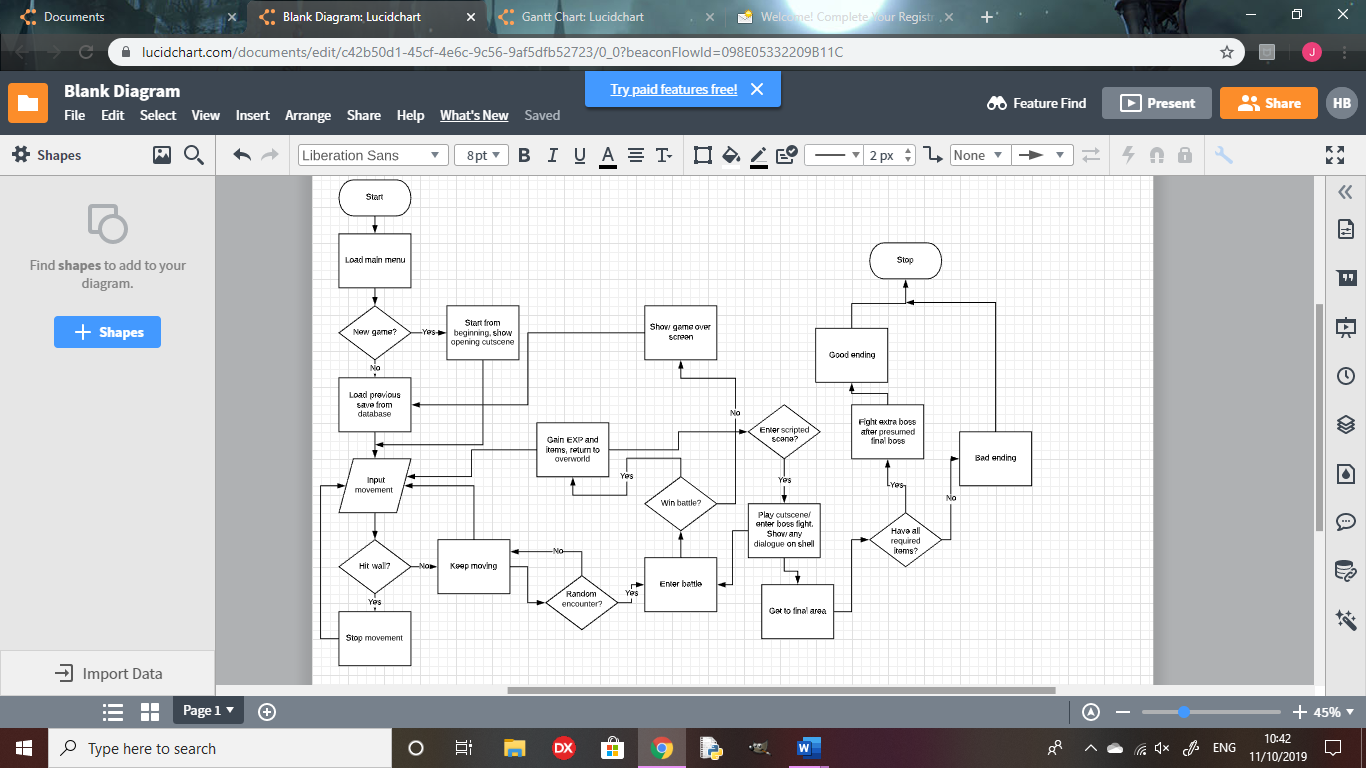
The game from here is split into 2 main sections: Traversal of world and battles. These are the 2 main things you will encounter during your playthrough. The ‘traversal’ section is split into movement, encounters and scripted events since those are what will happen while you are traversing the world; you may come across other characters for a sidequest, being a scripted event once you talk to them, and, most of the time, there is a random chance for the player to enter a battle while moving.

The scripted events are split into dialogue, scripted fights(ie boss fights) and cutscenes. The dialogue and cutscenes go together as the cutscenes will have dialogue shown on the python shell along with sprite animations. The boss fights are required for the player to proceed in the story, so you are unable to run away from the fight. There will also be some dialogue during boss fights along with a phase change when the boss has lost a certain amount of health. The scripted events are split this way because they are the only sorts of scripted events I plan on incorporating into my game.

The battle system will be split into the enemy turn and the player turn. The enemies will have a few unique attacks to attack the player’s party with, and they will have spells to heal themselves, buff themselves or debuff a character in the player’s party. The player turn will include all of the commands that the enemies can do, but the player will be able to choose what spells each character can have with the use of items, similar to Final Fantasy VII, they will be able to run away from the battle and, outside of battle, the player will be able to choose whether or not to control the other party members during battles. The battle system is split up this way because it means I can program all of the different commands one at a time, and test them against each other.

Enumeration is used throughout my game to calculate things such as damage given by attacks and health lost by attacks, along with the stats of different characters. Visualisation is used throughoit the whole game with the game window showing the game world using sprites that will be made from scratch.

The HIPO chart shows that I am able to implement procedural abstraction to make my game, as each main section is split into different parts to make it easier to manage, such as the battle system being split into enemy and player turns, and then those sections being split into the individual commands that can be run by each character.



**Describe the Solution**

There will only be 4 areas in the game, meaning there won’t be too many different sprites for the overworld that I will need to make, and there will be 2 exclusive enemies to each area, and 2 enemies that will show up over more than 1 area, so I will only need to make sprites for 10 enemies. As there aren’t too many unique sprites to create compared to bigger projects, I believe my target is achieveable to make each and every sprite for the game.

There will be 2 sidequests in each area to flesh out the world with small storylines for the characters you encounter. These may include teaming up with side characters or fetch quests for them, giving some insight into the inhabitants of the world. This will not require too much alteration to how the game is played, just some added dialogue for the stories, meaning I will be able to fit 8 sidequests throughout the game.

To reward exploration, there will be 3 items to acquire in order for the player to achieve the true ending, and there will be a full optional area based on Bloodborne, with a boss fight with Ludwig the Accursed/Holy Blade, with a phase change halfway through the fight. There will also be a fight with the Orphan of Kos at the end of the Bloodborne area, scaled with endgame damage to add more challenge to the player. The Orphan of Kos will also have a phase change halfway through the fight, similar to Ludwig. This area will be harder than the others, but will allow the player to gain useful items and gain lots of EXP to level up. This will be achieveable for me as it will be easy to place the items in each place and I won’t need to think of a full new set of sprites, as I can base them on the base material from Bloodborne.

NG+ will not be too hard to implement as I will be able to make it a boolean variable and scale damage and level of enemies based on whether or not the player is in NG+ or not. There will also be a new boss to encounter in an area that you be able to get to in your first playthrough, but there will be no boss there if you aren’t in NG+.

I will be able to balance the game well by testing the amount of damage each enemy and party character to check it will not be too hard or too easy for the player to make it through, so it will require some decent strategy to overcome, but won’t be impossible to beat without grinding levels for a long time.

Examples of usability features will be easy movement options and quick, fluid battle controls. This is because I want the player to ave as fun a time as possible with the game, and be able to pick it up and play very quickly, with easy-to-follow controls, like single key presses in battles to attack an enemy or use items. Easy movement options and battle controls will mean the player will get invested into the game as soon as they pick it up, as there will be no frustration due to controls.

There will be no HUD for traversal if the menu hasn’t been opened by the player, as it will be unnecessary clutter for the screen, and it would block the view for the player. For battles,the only display other than the sprites and backround will be the health of each character and enemy, and it will show who’s turn it is with an arrow above the character who’s turn it is. It will also show the attacks of the party character who’s turn it is, with a key next to each move to show the key for the player to press for the character to use that move. This will make the game easier to play for the player, as it will always show which key the player will be able to press and will show the effect tied to that key, so the player won’t need to memorize a lot of different controls in order to control the game effectively. The main menu will be quite basic, showing the title of the game and the 2 options: “New Game” and “Continue Game”.

There will be some boolean variables in my game, such as whether or not the player is in NG+, and if the player has beaten a boss fight, or entered a scripted scene.

There will be lists for enemies, party characters, bosses, items and walls so that, if, for example, I need to add wall, I can create the object and add it to the walls list, and iterate through the list for collision and scrolling, so that I don’t need to copy and paste every wall into each method, making my code much more efficient.

Most of the variables, however, will be real or integers, such as damage dealt by the player or enemies, and different stats for each character, like health, damage or defence.

There will be no need for validation in my game since I will make the code so that it only checks for correct inputs from the player so, if any invalid data is input by the player, nothing will happen.

Once the player saves the game, the values of every variable will be stored into a separate database to be called upon if the player dies in battle, or starts the program back up again and chooses “Continue Game”

**Describe the Approach to Testing**

My program will be fully object-oriented, so every function to be implemented into the game will be a method in one of the classes, such as movement options being a method in my player class. This will make it much easier to test each section of the code since every part in the HIPO chart will either be its own method or part of a larger method, so if something is not working, it will be easy to tell which part of the code will need to be altered. It will also mean that the code will be structured in an easy-to-read way, making it easier for a tester to look through the code to find bugs and make any alterations.

Once each section, described above in my HIPO chart, has been coded, such as traversal controls or placing sprites into the world, I will run the program to test whether or not everything works together and, if not, I will alter the code and test it again until everything up to that point works in tandem with all the rest.

I will be able to generate normal data in battle by using the exact key inputs to run certain commands for each character. For example if ‘S’ causes the player to attack the enemy, I will input ’S’. I can generate extreme data by entering the correct key but not in the same case as shown in the game. Using the example above, I can input ‘s’ instead of ‘S’. I can generate abnormal data by inputting a key that will not work at that time. For example, if ‘S’ is the only key that will work, I can input ‘Q’ and it should either output an error message or do nothing instead of causing the program to crash.