Project 7: Semester Project – Final submission

Title of Project: Choose-Your-Own-Adventure Zombie World
Names: Tri Bui, Gianni Valentine

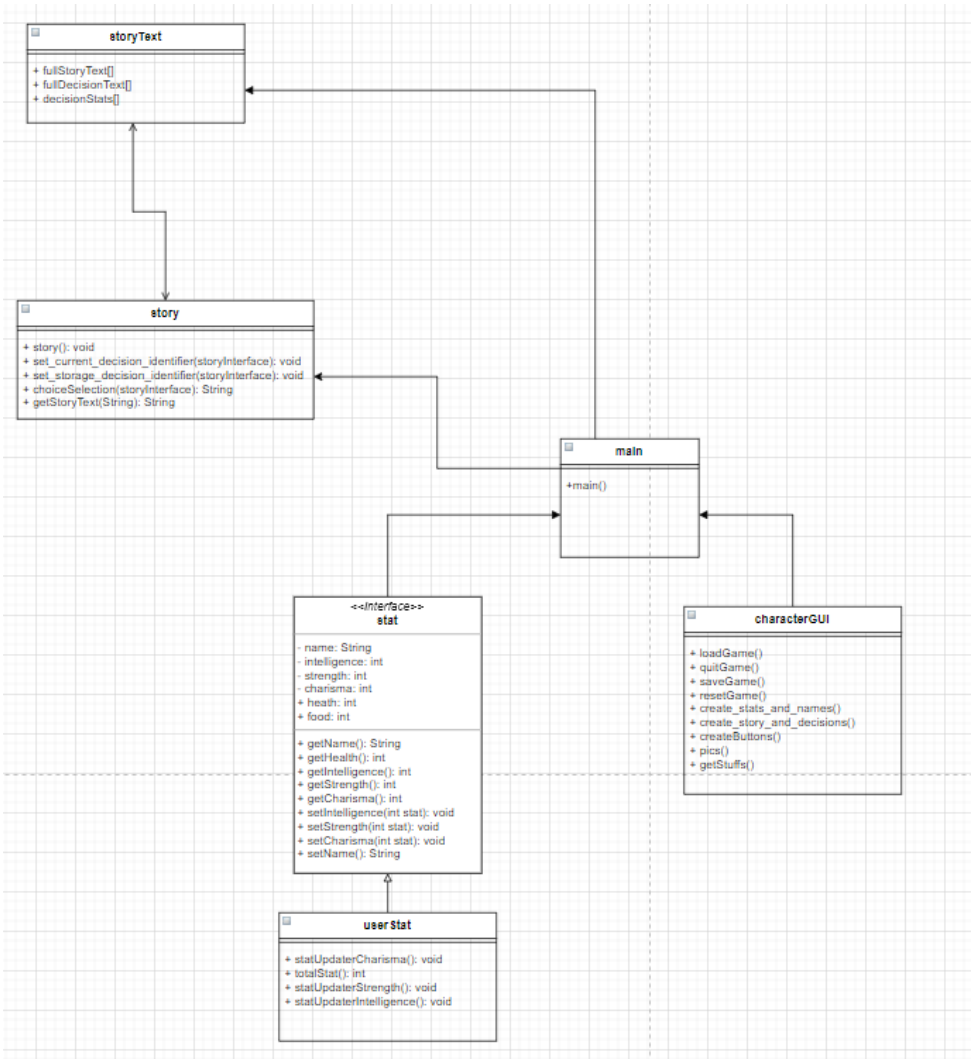
Final State of System Statement

Our project has been to create a text-based, choose your own adventure zombie game. From our initial planning phase till now, quite a bit about our project has changed, as often occurs in development processes. The full project is in a stable release state and we are proud of what we accomplished in the short two-week period. However, it is not fully complete, a few more updates would need to be rolled out. In terms of the features that currently exist. The GUI of the game is mostly completed, however, extra formatting would be required to make it look pretty. The GUI features a new game, a save game, a quit game, and load game buttons. Functionality-wise, the new game button, save game button, and quit game button works. The load game button has some outstanding bugs and doesn't work under some edge cases. This bug will be explained below. On the back end, there are two major features sets. The story interface and the character stat interface. The character stat interface is an interface/class that has getters and setters for each stat a user may interact with. Such as name, intelligence, strength, bravery, health, and food. We included a child class to manged some of the automatic updating of character stats. Now for the story interface, this is what makes the game a game. The GUI and database can be swapped out but the story interface is static. We implemented 3 major components in this class. First, is the temporary storage of a user's decision, each time a choice is made depending on the option selected either "a". "b", or "c" is appended to our decisions tree. An example would be the user selects the first option then the second then the first again, the decisions tree would be ["a", "ab", "aba"]. This then can be a reference to select the next available story and options for the user. The second component in the story interface would be the choiceSelection manager. This method accepts the option the user recently selected, updates the decisions tree storage, and requests the correct story, options, and stats to be posted. The third method is getStoryText, this contains the storyText, fullDecisionText and decisionStats. Using the predefined decisions tree we can iterate through each directory to find the requested data. This is where the iterator design pattern is implemented. Bellow are the two major issues we faced.

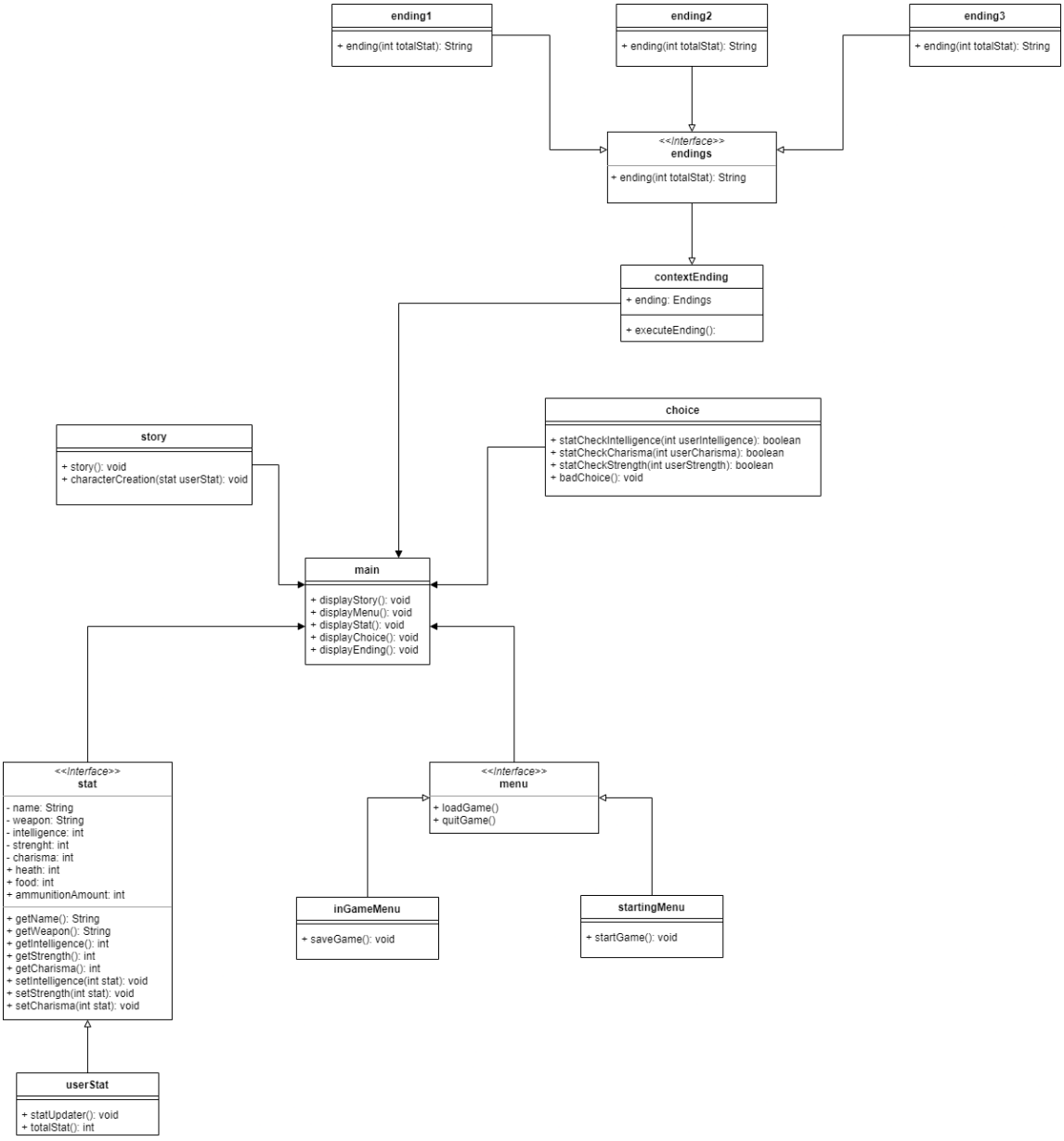
The load game button has some bugs. The reason for this is that we thought by replacing the decisions' text and story text the game would continue where it left off. However, the game instead resets to the beginning after you load in the decisions and story in the save file. This means that the decision the player picks in their save file has no impact on the game moving forward. This is caused by making the decision function to start from the beginning to the end of the game, without allowing access points so that the load function can start at where the player left off. This was in part fixed with our choice iterator but some issues still occur.

Another feature that we weren't fully able to implement is making a layout/ border. The reason for this is that creating a custom border for each section in the game is more complicated than we thought, as each section requires a different amount of spacings.

Final class diagram and comparison statement

- Final class diagram:
  - 

  - The design pattern that we used in this project is the iterator pattern. This pattern is used between the main function, storyText class and story class. According to the iterator pattern, storyText is the class that is seen as the container that contains all the story, decisions and stats for each decision. In the story class, there is a function called getStoryText. This function acts as an iterator that finds the corresponding story according to the decisions that the player has made along the way. This function is mostly used for the load function in the characterGUI.
- Project 5 class diagram:



- What has changed:
  - The main class only contains the main function in the final class diagram. This is because we already implement all the display functions through characterGUI.
  - The menu interface and its inGameMenu, and startingMenu has been replaced by a class called characterGUI. characterGUI contains all the menu functions of the menu interface, inGameMenu, and startingMenu classes. The characterGUI also contains other functions like create_stats_and_name. This displays the stats, and name for the player. Other functions are create_story_and_decisions, createButtons, pics, and getStuffs. create_story_and_decisions displays the current story, and current

decisions of that particular storyline. createButtons displays buttons like new game, load game, save game, and quit game. pics function displays a picture of the character. Finally, getStuffs function iterates through all the decisions being stored, which will return a unique key that when used with the load function, will load in the current story of the game.

- contextEnding, endings, ending1, ending2, and ending3 has been removed. That means the strategy pattern also didn't get used. Because of time constraints, we decided to put all the endings, storyline, decisions, and the stats that the player gains from picking these decisions into a class called storyText. Instead of the strategy pattern, we decided to use the iterator pattern. So the storyText function can be seen as the container class, while the story class contains the iterator that will iterate through all the decisions and storyline.

Third-party code vs original code

- We used some code from stackoverflow in order to create the function for the new game button. File: characterGUI.py Line: 379
    - This is the source: https://stackoverflow.com/questions/41655618/restart-program-tkinter/41655930
- To learn about how Tkinter work in general, I read this document about Tkinter
    - https://docs.python.org/3/library/tkinter.html
- To learn some extra stuff about the grid function in Tkinter, and how to place different labels. I watched a Youtube channel called Codemy.com, below here is their channel link
    - https://www.youtube.com/channel/UCFB0dxMudkws1q8w5NJEAmw

Statement of the OOAD process

Initially, we set out planning to use the strategy design pattern on our game endings. The thinking was there will only be 3 main endings and all else will be the same. However, after developing the story more endings were required. So it was no longer feasible to implement the Strategy Design pattern. That was the first issue. The second issue was continuing to try to implement the strategy design pattern. Time was wasted on creating classes to make it work. The third element was a positive one. While trying to implement the story and save functionality we discovered that the Iterator design pattern would work perfectly with our setup. Each story/decision has a unique identifier key, these keys need to be a reference for loading the game. So having a traverse be able to iteration through their would work. This was a pleasant surprise that something we just learned was immediately valuable in our code.

Demonstration

Link To Google Drive:
https://drive.google.com/file/d/1lEYtBYO1DV90jFMJDUBiT-P4GItN3RDG/view?usp=sharing