Agaros and the Cursed Realm

Text Adventure Game

Final Report

CS 467: Online Capstone project

August 12, 2020

Francisco Muniz – Matthew Saunders

With hard work, dedication, and sacrifice, our team was able to bring to life Agaros and the Cursed Realm. Named after a Duke of Hell, Agaros and the Cursed Realm is a text-based adventure game that follows a hero's journey as they seek to recover their memory, and in doing so, stop the demon Agaros from overtaking the realm. Inspiration was taken from current games and hit fantasy movies, but overall the theme of the game was our own. The entire project was written in C++ in order to capitalize on the use of inheritance and polymorphism which drove the flow of the game. While we are both proficient in C++, the design and logic driving the game presented an interesting challenge as we sought to implement a fantasy world that met the project requirements.

The overall design of the game is similar to the classic game, Colossal Cave Adventure. The game will print a description to the user of their surroundings and will prompt the user with a ">" to have them enter a command on what they wish to do. Agaros and the Cursed Realm implements a text parser, which will analyze and interpret the user's commands and perform/display the appropriate action. Using this functionality, the user will enter commands and guide the hero through the realm in search of the truth. Agaros will lead the user and hero throughout the realm, presenting problems and puzzles that the user must solve before being able to advance. Agaros also has the user face 1-on-1 fights with monsters, minions, and demons to save the realm from darkness, all while finding out who they really are.

Agaros and the Cursed Realm was created using only C++. Our text-based adventure game is pre-compiled into an executable named agaros. It has been tested and runs on the flip Oregon State University servers. The "game_code" folder contains all the source files. We have also included a makefile to assist in compiling the game. Simply type "make" in the flip terminal to compile the game.

And to run the game on a flip server, simply enter the command:

./agaros

As you run the executable "./agaros", you will be presented with the title screen for the game, and prompted to start a new game, or to load a previously saved game. If you select load game, it will check the "saveData" directory inside your current directory for the necessary information and load it into the game. Otherwise, you will start a brand new game.

A new game opens up with a prologue scene, giving some backstory to the game. The scene is set as King Elias has just finished summoning Agaros, in hopes of bargaining

with it to save Queen Helen who has been poisoned. Spoiler alert, it goes wrong, and the King ends up possessed by Agaros.

The game then fast forwards approximately three and a half months, where the hero wakes up in a house with no memory of what happened, or who you are. The game then officially starts, and you travel through the realm in search of help to regain your memories, and eventually save the realm from the possessed king, Agaros.

As we mentioned before, the game incorporates a text-parser to interpret the users commands. The user will input commands, such as “go east” to navigate, or “talk to tree” to interact with the world around them. There are 18 rooms the user will be able to navigate to, with features in each room the user can interact with. There are also various items around the realm that are needed to access other rooms in the game. The following walkthrough will lead you on how to complete the game.

**********SPOILER ALERT**********

- The game starts with the user in the Home room.

- In home , check the bookshelf to get the TEXT item.

-Go east to Holmhaven then go south to Depietto farm.

-Go east to the Valley and pick up some mushrooms.

-You may also interact with the boulders feature to reveal the stone item, which is needed later.

-Go back west to Depietto Farm then go south to the Black Forest.

- Talk to the merchant to trade the mushrooms for the jerky.

-Go back north to Depietto Farm then back east to the valley. Use the Jerky. The dog will run home to the farm. Interact with the boulders to get the dirty stone.

-Go back west to Depietto farm. Talk to farmer to get the oar.

- Then go back north to Holmhaven, then east to the Zemya river.

- Use the oar or interact with the boat to use it and get the sword.

-Go south to the valley then south again to the mountains. This will prompt a battle. After the battle you will be poisoned and end up in Nagorny village. Talk to the black smith and get his token. Talk to the alchemist to reveal the power stone(Must be carrying the dirty stone).

-Go back to mountains and pick up the ring from your previous battle. Put the power stone in the shrine, then go back to Holmhaven, either via the valley and river, or forest and farm.

-In Holmhaven, use the ring with the old man to get the goggles and use the token in the market to get the ale. Go north with the goggles on to go to the wastelands. Use the text on the tablet to translate the text.

-Go back to Nagorny village and talk to the black smith to get your sword fixed. Head west through the mountains and west through the forest and west again to the cave. A second battle will be prompted. After the battle, use your translated text on the wall to reveal the path to the dungeon. Take some water from the cave spring.

-In the dungeon, talk to the prisoner and give her water to reveal the secret about the tree in the forest. Hit the torture rack twice and pick up the spring. Use the spring to go to the lab due south.

-In the lab, fight the scientist and pick up the chemicals. Go back to the dungeon and use the chemicals on the foyer door to open it.

- Go east to the foyer, then east again to the dining hall. This will prompt another battle with The Butcher.

- Go north through the bed chambers and west to the terrace. Fight Argos's General and get his heart.

-Proceed back through the castle to the dungeon then back through the caves until you are back at the forest. Talk to the tree to reveal the secret pyxis. Go to the shrine in the mountains and use the pyxis and heart on the shrine to get the powered pyxis.

-You are now ready for the final battle. Proceed back to the castle back to the terrace. The tower room is now unlocked. Beat the final boss and we hope you enjoyed the game.

****

The above was a walkthrough on how to complete the game, however, there are various features in all the rooms that the user can interact with. As the user travels from "room" to "room", a description of the location will print out. Rooms you can travel to will print in blue, features in magenta, and items in yellow. You may use a variety of verbs on the features for different interactions. Verbs may be "look", "hit", "touch", and others you can find in the game.

In regards to the items available in the game, the hero is given a bag that can hold 3 items max, so you must be smart in regards to what items you wish to carry with you. You can drop items to make room for other items, and the dropped item will stay in the room you dropped it in, unless it is a mushroom or water. Those will just disappear.

There are certain features, that when interacted with, will provide a little more backstory, such as looking at the columns in the desert/wastelands. In certain rooms, a flashback will trigger, describing a memory from the hero prior to them losing their memory, that will help paint a picture as to what happened leading up to your amnesia event.

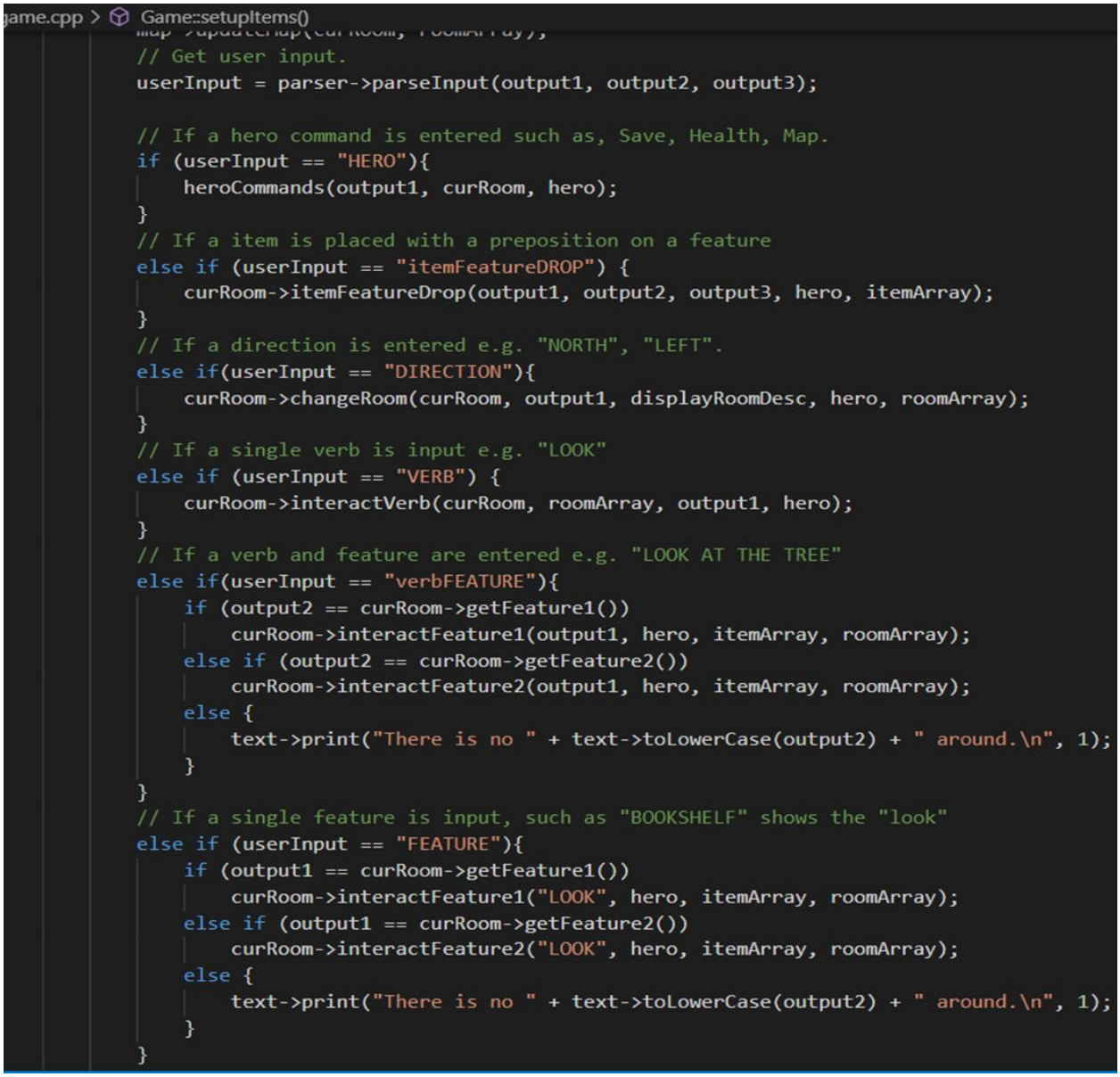
We sought to incorporate a fantasy element to our game, provide some mystery and puzzleness, and be as interactive as possible. We hope the above assists you in enjoying the game as much as we enjoyed creating it.

As mentioned before, the entire program was written in C++, not only because we were both proficient in it, but so we can take advantage of the polymorphism and inheritance capabilities of the programming language. We wrote an extensive list of classes that we incorporated throughout the game.

The Game class is the engine behind the game. This class runs the game and houses all the objects we use throughout it. We initiate a Game object in our main.cpp file and call the Game::run() to start the game. In here, the game creates itself. We have Game functions that will create all the Room objects we need, and will link them up appropriately. It will create the appropriate Character objects, and create a Parser and TextFormatter object.

```
38  class Game
39  {
40      protected:
41          std::string userInput;
42          std::string output1;
43          std::string output2;
44          std::string output3;
45          Room* home;
46          Room* currentRoom;
47          Room* caves;
48          Room* castle;
49          Room* desert;
50          Room* forest;
51          Room* river;
52          Room* mountains;
53          Room* town1;
54          Room* town2;
55          Room* valley;
56          Room* diningHall;
57          Room* tower;
58          Room* terrace;
59          Room* bedroom;
60          Room* dungeon;
61          Room* foyer;
62          Room* lab;
63          Room* farm;
64          Room* roomArray[18];
65          Character* hero;
66          Character* demon;
67          Character* minion1;
68          Character* minion2;
69          Character* minion3;
70          Character* minion4;
71          Character* minion5;
72          Item* ring;
73          Item* sword;
74          Item* book;
75          Item* mushrooms;
76          Item* goggles;
```

The Parser class is what interprets the users commands. It will prompt the user and take in the input. From there, the Parser object will run it through our ArrayClass object looking for a match. Depending on what array the input is found it, helps us determine what type of input the user entered. Whether it is an interaction with an item, a feature, or changing room. Depending on the type of input, the Parser object appropriately returns the verb, feature/or item, direction, along with the object being interacted with. These return values are then used in the Game object to call the appropriate function to interact with the item, feature, or direction as intended.



The Parser class uses a separate ArrayClass object. The ArrayClass class houses all of the arrays containing strings of various inputs the user can enter. We have

an array associated with items, verbs, features, directions, rooms, and synonyms. These are what the parser class compares the input to and determines what the input/command is wanting to do.

The TextFormatter object is created in most of the main classes, not just the game class. This class is responsible for printing text to the console at different speeds and colors. Based on the values passed into the print() and changeColor() helps us add a different feel and vibe to the game.

The Character class is our parent class for the derived Hero, Minion, and Demon classes. It contains simple information like the name, but more importantly, has the attack and defend functions that help simulate the fights. These attack and defend functions were declared as pure virtual functions in the parent class, so we can modify them individually for each derived class. By having the derived classes inherited from the base Character class, we were able to call a base Character object from other functions, and still pass in the Hero, Minion, or Demon object. This made it extremely easy to design functions that could be used in various rooms with different character objects.

The Room class is also a parent class. We created a separate class for each of the rooms in the game, and they are all derived from the Room class. We did it this way for the same reasons as the Character class. The Game class has a function called gameLoop that takes in a Room object and runs a loop performing the various commands returned by the parser object. The loop will continue to run on the same Room object, unless the user changes room. Then the loop is run with the new room. This is possible due to the fact that all the rooms are derived from the same Room parent class with the same functions.

There are smaller classes that still play an important role in our game. The item class has all the information for the various items you can interact with throughout the game. The bag class has a bag data structure to hold the hero's and each room's items. And the filesystem class is responsible for saving and loading the data in a current game.

In order to manage the large amount of classes and individual files, we implemented a test driven development process alongside Git and GitHub for version control. Git was extremely beneficial in allowing us to work on separate parts of the project to continue moving forward.

Matthew Saunders led our team in development and time put in. Matthew developed the Parser class, TextFormatter class, and the filesystem class on his own. These features allowed our game to be what it is today. Besides these accomplishments, Matthew played an integral part in implementing the logic behind key scenes in our story, and how certain features and items would play a bigger role in the game. Lastly, he also tested and debugged a majority of our code as it was being implemented, and took it upon himself to refactor a lot of our code into smaller classes, which improved readability and efficiency in our program.

Francisco Muniz thought of the idea of using inheritance and polymorphism to create the overall structure of the game. He then implemented this into the base Character and Room classes, which would serve as the foundation for the rest of the game. He also developed the functionality to make it possible to have characters in simulated fights, which are a key part of the story. Lastly, he came up with the general idea for the theme of the game, and started the process of implementing the story into the rest of the game.

Since our team was short handed one developer, both Matthew and Francisco split the tasks of updating all the rooms with the features, items, and logic that would go into the game.

At the start of the project, we had our third team member moved to a separate project so we knew we would be pressed for time. Our original project plan incorporated the main functionality we have in our game, but the order in which things were completed, or who completed the task changed all the time. Since we were only two developers, as soon as one finished their assigned task, they would move to the next one. We had an overall plan on what we needed to have accomplished before we moved on, and it was more important to us to have that completed, rather than adhere to the original plan and who was in charge of what. We also restructured the order of the rooms, the rooms themselves, and some of the items we initially intended to use. After developing the story line, we realized it would work better in the way we have it now. We removed some rooms from the inside of the castle, and added more rooms and features to the outside world. We had also not anticipated having so many classes, but after refactoring a lot of our functions to more appropriate classes, we saw the benefits of cleaner code and increased readability, which helped the development moving on.Even though we switched quite a bit of our story, we are satisfied with the end product of Agaros and the Cursed Realm.

This project was unlike any other we had undertaken before. We were able to implement the software development process, from start to finish, on an idea that had

set requirements and time demands. Not only were we given the opportunity to hone our own C++ skills, we were given the chance to learn new coding and style techniques from one another. We learned the importance of clear communication and implementing proper continuous integration to maximize our efficiency when developing a large scale project. The project itself was time consuming, and there were plenty of sacrifices made, but the joy of creating something that was ours, and watching it grow to fruition is unlike any other feeling. While we were designing a game, we still received the experience of developing a program for a client in a real world situation, which will now translate to our future careers as developers. And for that we are grateful. We hope you enjoy the game and are able to save the realm from Agaros's darkness.