CS 467 Capstone Course
Final Report
Winter 2018

Adventure Team Final Report

Adventure Team Members

- Michael Heinemann
- Stephen Liu
- Heidi Uphoff

Introduction

Our project group set out to create a text adventure-style game, complete with text/language parser that understands a number of simple phrases. It is like the original Colossal Cave Adventure but with our own interactive fiction storytelling where each member contributed in the creation of the settings, obstacles, enemies encountered by the player of the game. We had group projects in the past in other classes but none of them at the scale and the length of time it takes to complete as this class. One of the most important things in working as a group and being able to easily keep track of the changes made by each group member is utilizing a version control system. We decided on using github which allowed each member to see the git commits, have the ability to create branches, the ability to merge with the master branch, roll back code changes, and writing descriptive git commits. Since each of us have different work schedules and time zones, group messages and group meetings also played an important role in our communication.

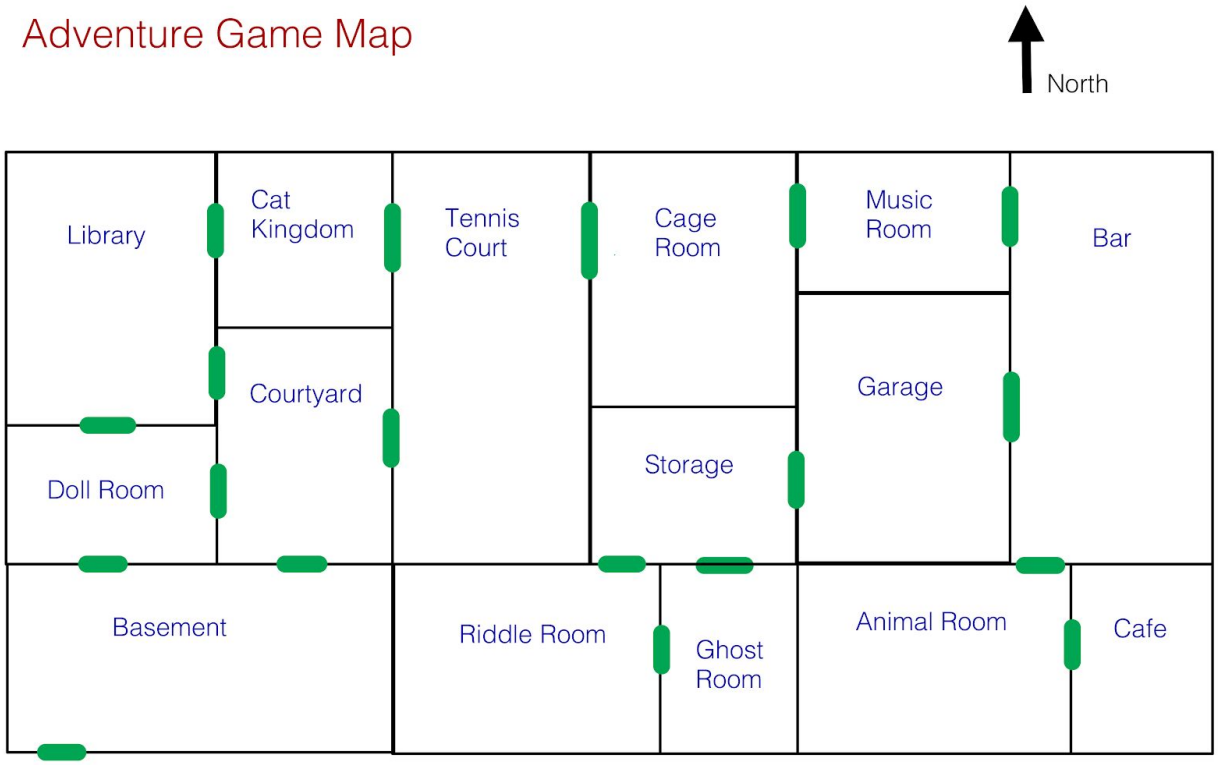
We felt a great sense of accomplishment as we saw the different pieces start to come together. Along the way we had to make multiple changes. Things such as the way our data is written in the room files or the way we wrote our structs needed to be changed so the program can work properly. Working in C with reading/writing text files can be a tedious process. An example of a bug we encountered is an extra space left in a text file resulted in a segmentation fault (core dumped). We utilized tools such as valgrind to help identify memory errors.

Description of Program

In the most basic terms, our program places the user in a starting room. The goal is to make it to the end room. The user is given a text description of each room with two main features that the user can perform certain actions on. Sometimes the feature is an obstacle that the user needs to overcome. Obstacles can be an enemy the user needs to defeat or something that blocks the user's progress. Other times the feature reveals an object that the user can place into their inventory and is necessary to conquer said obstacles. Below is the Adventure Game Map layout.

CS 467 Capstone Course
Final Report
Winter 2018

Adventure Game Map



Usage Instructions

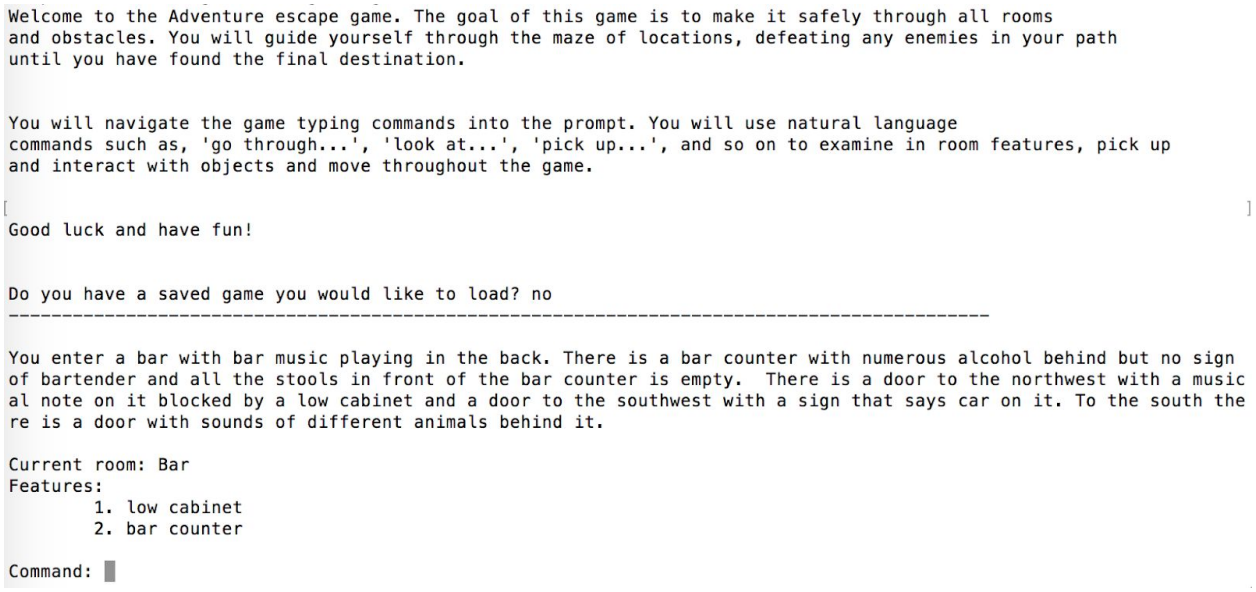
Compiling Software

```
make gameEngine
gameEngine
```

Starting The Game

The game starts off with an introduction that describes the goal of the game and the commands the player can use to navigate the game. For those that never played a text adventure-style game and needs further help, they can enter the command “help” and a set of verbs and their synonyms the game understands will be listed on screen.

Then the player will be asked if he/she has a saved game that they will like to load. If the player answers no, a new game is created. The player is first told that they are in the Bar room to start off the game and is given a text description of the bar room that includes the two main features of the room and the various exits of the room.



Saving and Reloading Games

At any point during the game, the player can enter the command “save”, and the game will ask the user to create a save name. After a save name is entered, the game will notify the player that the game has been saved and exits the game. If the player attempts to use a save name that already exists, a notification will be displayed to the screen and they will be asked to enter a save name once again. When the player restarts the game, they will be asked if they will like to load a saved game. If the player answers yes, the game will ask the player to type in the save name. The game will load the saved game state. Which means that the player will start off in the room they were in with all the objects they had in their inventory, objects that they dropped in a room remain in that room, and all obstacles and enemies defeated stay defeated.

Moving Through Rooms

Movement through the rooms is handled from user input through the command line. To navigate through the game the user can enter commands such as “go north”, “depart sw”, or “exit east door”. Users can also just type the cardinal direction and the game will send them through the correct door. The game will also recognize descriptions of doors instead of the cardinal direction. Directions to and descriptions of each door is provided in the room descriptions. Stop words can be handled, so including words like “through” and “to” should not inhibit movement through a door. If any input is not recognized, for example the user tries to go through a north door when one doesn’t exist, an appropriate error message will be displayed to the user.

Examining Features

Throughout the game in each room the user will have available certain features that can be examined. If the user types in “look at” plus the feature name, a short description of that feature will be output to the screen. Users can also try to hit, move, or open features. What happens after the user types those commands depends on the specific feature. Some features are

actually enemies or obstacles that the user will have to overcome in order to proceed in the game.

Overcoming Enemies and Obstacles

Certain enemies and obstacles must be defeated for the player to advance forward in the game. For example, in the game there is a magical beast that is guarding a piano that contains the key to open the locked door ahead. The magical beast must be defeated first for the player to be able to take the key from the piano to unlock the door ahead. The magical beast can only be defeated with a magical wand so if the player did not pick up the magical wand when they encountered the magical beast, they will need to back track and look for the magical wand.

Interacting with Objects

Players can take objects from rooms. This adds it to their inventory. They can later drop objects from their inventory in other rooms or use them against features. To see any objects that have been dropped the user can type in the command “look at room” this will display the room name and features again, as well as any dropped objects on the floor.

Complete List of Supported Verb Commands

- look - repeats the long description of a room
- look at - takes a closer look at specified room feature
- go - moves user through specified exit
- take - adds specified object to user's inventory
- drop - removes specified object from user's inventory
- help - displays list of supported actions to user
- look at inventory - lists user's current inventory
- hit - hits specified object or feature
- open - opens specified object or feature
- move - moves specified object or feature

  *There are also various synonyms to the words above that will work as well.

Cheatsheet Playthrough Instructions

1. Game starts off in the Bar room. Look at the bar counter and the game will tell you that there is a flashlight there. Take the flashlight.
2. Go south to Animal Room. Look at the skeleton. Take a bone. Take a second bone so that your inventory has two bones. Hit the dog with the bone and the dog will run away, leaving the doors free.
3. Go east to the Cafe. Look at the table and take the magical wand.
4. Go back west to the Animal Room and back north to the Bar.
5. Go southwest into the Garage. Use the previous method of hitting the dog with the bone to tame it. Look at the car. Open the car trunk to reveal catnip. Take the catnip.

6. Go west to the Storage room. Hit the spider without using an object to defeat it. Look at the shelf to reveal a tennis racket. Take the tennis racket. Go back east to the Garage then east again to the Bar.
7. In the Bar Look at the low cabinet to notice that the low cabinet can be moved. Move the low cabinet to access the door to the northwest.
8. Go northwest into the Music Room. Hit the magical beast with the magical wand. Look at piano. Open the piano to reveal a key. Take the key. Use the key which will unlock the locked door to the Cage Room.
9. Go west into the Cage Room. Hit the bat with the flashlight to defeat the bat.
10. Go west into the Tennis Court. Hit serena william's serve with tennis racket to defeat serena williams. ("hit serena server with tennis racket")
11. Go northwest into Cat Kingdom. Defeat the lion by giving it the catnip.
12. Go west into the Library. Look at book of wallpaper. Take the book. Hit the librarian with book to defeat him.
13. Go south into Doll Room. Look at the life size barbie. Hit the life size barbie to defeat it.
14. Go south into the Basement look at the chest. Open the chest to reveal a crowbar. Take the crowbar. Use it to open the door southwest to exit to the outside ("hit lock with crowbar").
15. Go southwest. Congratulations! You reached the end of the game!

The walkthrough above shows one path that can be executed to complete the game, but there are also other paths that can be taken. You can use the map given in this document to help navigate the rooms.

Team Member Contributions

Michael Heinemann - *Game Engine Lead Developer*

The development of the game engine was a very time consuming part of this project as many things had to be thought out well to accommodate changes and actions throughout the game. The process started by making sure the small things worked, such as movement from room to room, proper descriptions being displayed, proper use and functionality of other files the game engine relied on, and basic user input. Once this part was fleshed out we were able to add in more functionality, working function by function, until all desired action and playthrough was reasonably accommodated. Every action that can be completed within the game was refactored into its own function for easier testing and debugging. Along the way, changes to different elements and files were edited and changed to make sure every function could be implemented with as many possible "room" scenarios as possible. Making each function to be able to work with every room was a challenging process as errors and inconsistencies would come up as different scenarios were addressed. This would result in many changes being made throughout all files to different data variables and elements to make sure everything worked smoothly and consistently. The game engine development ended up being a lot of trial and error. By this I mean each function was developed to work as originally intended from our initial planning. As expected, nothing operates properly first try so edits need to be made in certain areas and changes in others. Once it seems that a function is working, putting it through

rigorous testing in all available rooms often resulted in more situations and bugs that needed to be addressed. This sometimes resulted in functions or data having to be completely rewritten or simple fixes in some lines of code. Being consistent with our data was important to limiting mistakes in the game. As expected the game engine relies on both the command parser, and room loader files to get certain data. As a result edits also had to be made within these files to accommodate the game engine.

Testing and debugging was an important part of the development of the game engine. This resulted in a lot of time being taken to run the game through multiple different scenarios making sure everything worked as it should. Sometimes it was determined files needed to be rewritten and sometimes only easy fixes were necessary. Not only did we have to check for bugs within the code, but also make sure everything would make sense from the users perspective. As this is a text based game, the user relies on the information that is being displayed to them so making sure everything lined up, and the game could be easily navigated was important.

Stephen Liu - *Data Files and Data Loading Lead Developer*

This is a large project with many interacting parts. The Project Plan helped us lay out a plan on how to approach such a large project by breaking it down into chunks. First I reviewed projects I have done in the past that pertained to reading/writing to files and went over C I/O functions such as fseek, fgets, strtok, etc and wrote the RoomLoader programs. However, along the way we encountered a bug. The text we read from file when string compared with the string we are seeking was not equivalent. Using strlen, it was determined that when the text is being read from the file, an extra space is added at the end. Michael and I spend a long time trying to figure out why this was happening but in the end Michael came up with a work around by replacing the space with a '\0'. Michael started building the foundation of the game engine. Because of the division of labor, Michael hardcoded some of the information to test the game engine. Heidi pointed out that this violated the Technology Requirements so I was tasked with connecting RoomLoader programs with the game engine. This involved changing some aspects of the Room struct. Originally struct Room had a struct Room *Exits[MAX_EXITS] member that was changed to a char *Exits[MAX_EXITS];. We each designed 5 rooms and needed to connect all 15 rooms together. Along the way we modified the room connections and stories a few times and as a result, the file contents needed to be updated. We set a goal of having all the rooms connected and being able to travel through all the rooms through different paths from the start room all the way to the end room. After that was accomplished. The next thing Michael started to build the engine for is the ability to use verbs on the features in the room. Since we needed to have objects interact with features, an object.txt file was created. Object information (such as description and startroom) from the room files were pulled out and placed in object.txt. Information pertaining to what the object is used for was also taken out from each room-x.txt and placed into object.txt. I went ahead and created ObjectLoader files that contained an object struct and an inventory struct. Adding some codes and modifications in the GameEngine files, allowed the player to enter the commands to take an object, place it in

their inventory, and drop the object in the same or a different room. All the while, object struct, inventory struct, and room structs are all updated accordingly. Checks were also added to make sure only objects in the current room can be taken. I also encountered a segmentation fault bug I couldn't figure out but later Michael used valgrind and was able to identify the cause to be not allocating enough space so everything else was being thrown off. The calloc(255, sizeof(char)) was changed to calloc(500, sizeof(char)). I have heard of valgrind but have not used it before. So I looked over the valgrind manual to supplement gdb. As we add new features, we kept noticing new changes that need to be made for things to function correctly. As changes were made, Heidi also needed to make changes in the parser. We also discussed about redesigning the map connections/features/objects to create multiple paths to victories that makes the player have to traverse through more of the rooms instead of being able to avoid some rooms to advance. The next task for us to tackle is the interaction of objects with features. I proposed a way to add this into our code but as we discussed more and was writing the code, Michael noticed some issues with out data from the room files that would cause inconsistencies. In the end, Michael created another struct for features in the RoomLoader files. This fixed the inconsistencies we were getting. So we changed all of our fifteen room-x.txt file to the new format and removed some of the exits we talked about removing. Each room was then added back individually and tested before moving on to adding the next room to ensure everything is working properly. We added in the ability for the user to input the exit to the next room using the description of the exit besides cardinal directions and without "go". This required adding additional exit description information in the room-x.txt to be read by the program and a getExit function added inside the CommandParser files by Heidi. A lot of time was spend testing the program and ironing out the bugs.

Heidi Uphoff - *Command Line Parser Lead Developer*

Development of the command line parser included several phases that overlapped. The first was research into natural language parsers, which merged with research into search engine input and processing since they share similar features. The second phase was code writing to incorporate what was learned in the research phase with what was needed for the software. Lastly, development needed testing and refactoring. Weekly changes were needed to update the parser to match the changes the teammates made to their code as well as to correct any bugs discovered during testing as the program became more mature. The parser was refractored and commented to follow general guidelines set forth by professors at Oregon State University and those from the book *Clean Code* by Cecil Martin.

The parser takes user input and processes it in such a way that it returns the verb and noun the user most likely intended. The parser returns one verb and two nouns to the game engine, although the game engine only uses the first noun in the program. Input is lowercased, stripped of punctuation, and stripped of stop words such as prepositions and articles. First the parser searches for verbs. There are ten verbs utilized by the game. The parser looks for these as well as catches some synonyms. Next, the parser looks for exits both for cardinal directions and for

exit names. After that, the parser seeks out any features and objects used in the game. Finally, the parsers catches any other nouns.

This team member completed other tasks to assist her group and contribute to the project as a whole. She set up the initial github repository and documentation for collaboration. She worked on the room data for 5 rooms and helped the other team members merge all of the room data together into a cohesive game. After the first few weeks of development, weekly tasks included team code review, testing, and feedback.

List of Languages and Other Development Tools

- Programming Languages - C
- Version Control - git and githb https://github.com/endtoendpaper/adventure-game
- Development Servers - OSU's flip servers

Description of How Software Functions

Room data, objects, inventory, and saved game states are stored in text files in sub-folders in the same folder as the executable program code. When a new game is started, this data is loaded into structs and used by the Game Engine.

./
  adventure-game *(executable C file)*
  /rooms *(sub-folder to store room files)*
    room-1
    room-2
    room-3
    room-4
    room-5
    room-6
    room-7
    room-8
    room-9
    room-10
    room-11
    room-12
    room-13
    room-14
    room-15
    inventory
    objects
  /gamestates *(sub-folder to store saved game state information)*

gamestate-1
gamestate-2
….

Three C files handle loading the game data from the text files into the Game Engine and saving new gamestates into new text files

- RoomLoader.c
- SaveGame.c
- ObjectLoader.c

The command line parser is run with one C file.

- CommandParser.c

The Game Engine which runs the game from start to finish is run with one C file. It utilizes all of the other C code files.

- GameEngine.c

Conclusion

In seven weeks, the Adventure Team completed their text command line adventure game modeled from classic text adventures like Colossal Cave and Zork. Each team member spent over 100 hours developing the software for a total of over 300 development hours. The project was split into three main parts: the game engine, the text files and data loading, and the command line parser. Each team member took the lead on one of these parts, but also spent much of their time learning each other's code, contributing to the world building for the game, testing, and debugging. The team had to research new topics in order to develop a natural language parser and to debug the code for difficult to find memory and string bugs. Work on this project enabled the team to polish the coding and software engineering skills they earned during their time as Oregon State University students.