Camelopardalis: Final Report

Team Members:
Aaron Boutin
Babatunde Ogunsaju
Jessica Spokoyny

Introduction:

From project genesis up to the very end, this project saw a vast amount of change. Most of what it is started as a well conceptualized plan from week one, then organically grew and evolved as each individual not only learned new and better concepts both academically and empirically, but from the experience of working with each other.

Over the last 8 weeks, our team has developed and refined a murder mystery themed command line adventure game. We had a great time developing the story and characters and later faced some challenges when deciding how to implement the different aspects of the game. Overall, we worked very well together and created an environment where we were all getting our hands dirty in each others' code.

As you read on, you will see what the game looks like from a user's perspective and usage instructions that allow you to take the journey through the game from beginning to end. From debugging with GDB to collaborative development in Cloud 9, we used C++ with different tools and libraries to accomplish the completion of our game. With graphical examples, we'll show you a bit of what we did, as well as clarify what each team member accomplished for this project, bringing together the roles of Engine Developer, Data Format Developer, and Command Parsing Developer.

The User's Perspective:

Our program is a text-based, interactive adventure-style game. The user starts up the game to find that he/she is a detective from the big city called in to a small town to investigate an unexpected death. A young man, Bobby, had died and his death is questionable, leaving everyone a suspect. It's the detective's job to go around town taking clues and evidence from different rooms and saving them in his/her inventory. Eventually, the detective gathers enough evidence to arrest the guilty party. The game starts in the local bar, The Thirsty Whistle, where Bobby died but has since been removed to the morgue.

The user can control the detective through intuitive text commands using the command line interface. The user enters in natural language sentences, using specific verbs to perform the following actions:

- **Go** <room/direction> - allows the detective to travel between rooms (note that only travel between adjacent rooms is permitted)
- **Look** - gives a general description of the detective’s current room including which object and people are present
- **Look at** <object> - gives a description of a specific object in the detective’s current room or the detective’s inventory
- **Take** <object> - allows the detective to take an object from the current room and place it into his/her inventory (note that only certain object can be taken from rooms)
- **Drop** <object> - allows the detective to remove an object from his/her inventory and leave it in the current room
- **Help** - lists all of the verbs the game understands (note that this doesn’t list hidden verbs)
- **Inventory** - lists all collected clues and evidence in the detective’s inventory
- **Hint** - provides the detective with a list of valid room-specific actions to take in the current room (note that not all actions and objects are specified as some are hidden)
- **Talk** <character> - allows the detective to speak with any character in the current room
- **Arrest** <character> - allows the detective to incarcerate a character and accuse him/her of murdering Bobby
- **Use** <object> - allows the detective to interact with an object using another object (for example: use lockpick on door)
- **Knock** - allows the detective to knock on a door
- **Edit** - allows the detective to view, add, or remove from the suspect list
- **Layout** - allows the detective to see a map of the town (note that not all rooms will be present as new maps are displayed when hidden rooms are discovered)
- **Drink** - allows the detective to drink if the current room has beverages available
- **Load** - loads a saved game from file
- **Save** - saves the current state of a game to file

Hidden verbs:

- **Hack** <object> - allows the detective to unlock any piece of technology

- **Insight** <object> - allows the detective to see if an object is important in the grand scheme of the game

Usage Instructions:

- (See README.md to run the game)
- Layout
- help
- Talk to Trent
- (OPTIONAL) Drink a beer
- Go to Trent's House
- (OPTIONAL) Drink the chicken
- Go South (should now be at Moose's Front Porch)
- East (should now be at Police Station)
- (OPTIONAL) hint
- (OPTIONAL) Look at abe
- Talk to abe
- Take bobby's phone
- West (should now be at Moose's Front Porch)
- Knock on the door
- West (should now be in Moose's Living Room)
- Talk to moose
- East (should now be at Moose's Front Porch)
- East (should now be at Police Station)
- South (should now be at Beth's Front Porch)
- (OPTIONAL) Use the lockpick on the door
- Knock on the door
- East (should now be at Beth's Living Room)
- Talk to beth
- Talk to Claire
- West (should now be at Beth's Front Porch)
- North (should now be at Police Station)
- Talk to abe
- Take the warrant
- South (should now be at Beth's Front Porch)
- West (should now be at Sam's Front Porch)
- Knock on the door
- use the lockpick on the door

- (OPTIONAL) use the lockpick on the door
- West (should now be in Sam’s Living Room)
- West (should now be in Sam’s Bedroom)
- Look at the desk
- Take the code
- Look at the code
- Use the code on the laptop
- Take the emails
- East (should now be in Sam’s Living Room)
- East (should now be at Sam’s Front Porch)
- North (should now be at Moose’s Front Porch)
- West (should now be in Moose’s Living Room)
- Talk to sam
- East (should now be at Moose’s Front Porch)
- East (should now be at Police Station)
- North (should now be in Pharmacy)
- North (should now be in Morgue)
- Talk to frank
- Take the Autopsy report
- South (should now be in Pharmacy)
- Talk to rory
- Take the pickup receipt
- South (should now be at Police Station)
- South (should now be at Beth’s Front Porch)
- East (should now be in Beth’s Living Room)
- Talk to beth
- Talk to Claire
- West (should now be at Beth’s Front Porch)
- West (should now be at Sam’s Front Porch)
- South (should now be at Bobby’s Front Porch)
- use the lockpick on the door
- West (should now be at Bobby’s Living Room)
- South (should now be in Bobby’s Office)
- Look at the desk
- Take the loan application
- Look at the loan
- North (should now be at Bobby’s Living Room)
- East (should now be at Bobby’s Front Porch)
- North (should now be at Sam’s Front Porch)
- North (should now be at Moose’s Front Porch)

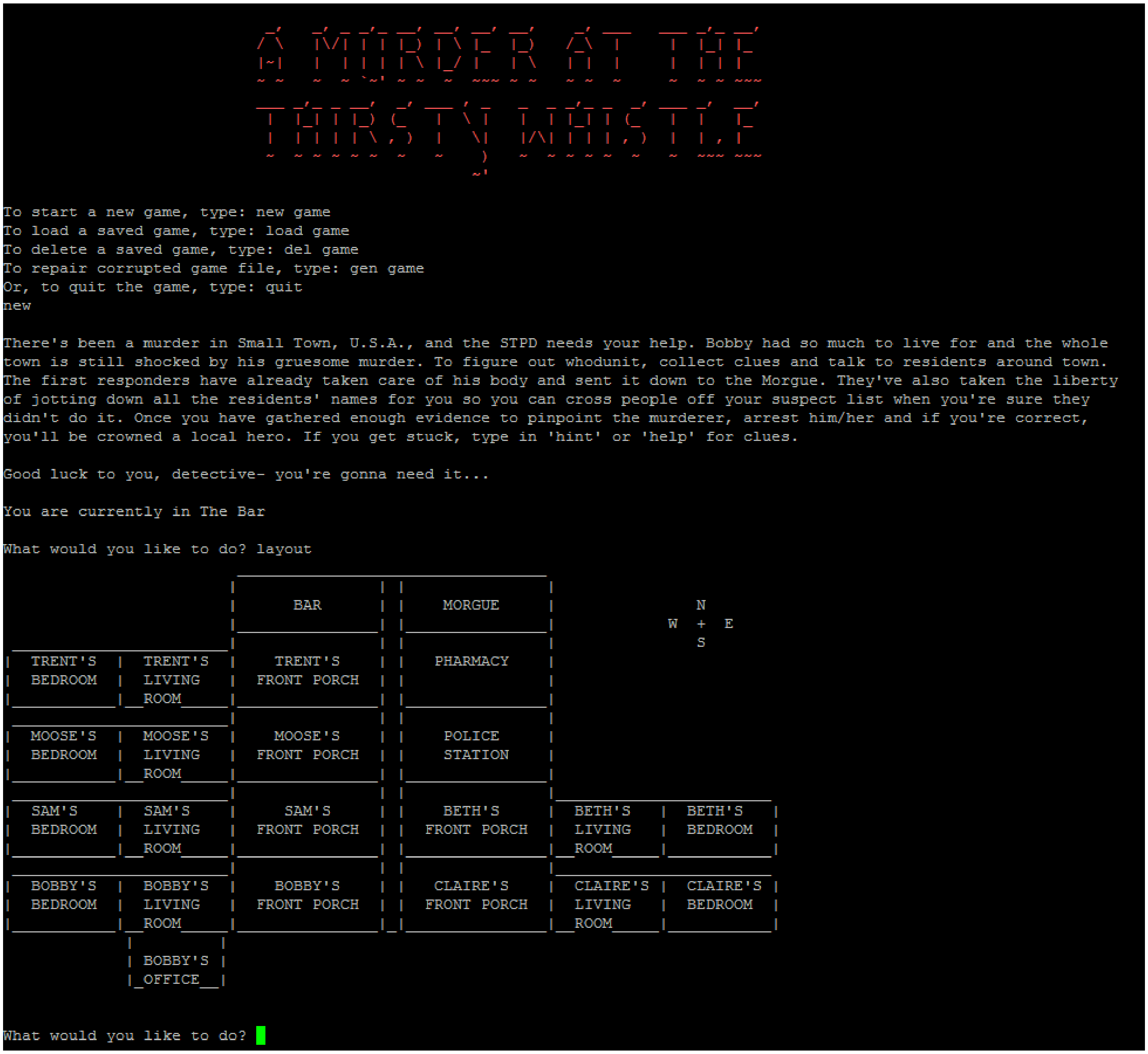
- North (should now be at Trent’s Front Porch)
- North (should now be at the Bar)
- Talk to trent
- North (should now be at the hidden room)
- Use Flashlight to see
- Take trevor’s wallet
- South (should now be the Bar)
- Talk to trent
- South (should now be at Trent’s Front Porch)
- Knock on the door
- Use lockpick on the door
- West (should now be at Trent’s living room)
- West (should now be in Trent’s Bedroom)
- Talk to trevor
- East (should now be at Trent’s living room)
- East (should now be at Trent’s Front Porch)
- East (should now be at Pharmacy)
- South (should now be at Police Station)
- South (should now be at Beth’s Front Porch)
- East (should now be at Beth’s Living Room for event)
- West (should now be at Beth’s Front Porch)
- South (should now be at Claire’s Front Porch)
- Use lockpick on Claire’s front door
- East (should now be in Claire’s living room)
- East (should now be in Claire’s Bedroom)
- Look at the dresser
- Look at the key
- Insight on key
- Take the iron key
- inventory
- (OPTIONAL) Look at the photos
- (OPTIONAL) Insight on pictures
- West (should now be in Claire’s living room)
- Use the key on door
- South (in Claire’s hidden bunker)
- Look at the shrine
- Take the bottle
- Arrest Claire

Alternate Ending:

- (Start a new game)
- At any point in the game, type in “Arrest the cat”

The next list of steps will show other verbs not shown in the main example, this is primarily because of save and load:

- (start new game)
- South (at Trent’s porch)
- South (at Moose’s porch)
- East (at police station)
- Hint (notice warrant under list of things to take)
- Take the warrant
- Hint (notice warrant is gone)
- Go west (moose’s porch)
- Inventory (notice warrant)
- Drop the warrant
- Hint (notice it is in list of things to take)
- Inventory (warrant is gone)
- East (at Police station)
- Save
- (enter name of game)
- (start game, but instead of new, type in load)
- (enter number of name of saved game)
- (You will be at Police station where you saved)
- Hint (notice warrant is not there)
- West (to moose’s front porch)
- Hint (there’s that warrant, right where you left it)
- Take the warrant
- Inventory
- edit
- list
- edit
- remove
- Abe



```
                            _,   _, _ _,_ __, __, __, __,   _, ___   ___ _,_ __,
                           / \   |\/| | | |_) | \ |_  |_)  /_\  |     |  |_| |_
                           |~|   |  | | | | \ |_/ |   | \  | |  |     |  | | |
                           ~ ~   ~  ~ `~' ~ ~ ~   ~~~ ~ ~  ~ ~  ~     ~  ~ ~ ~~~

                           ___ _,_ _ __, _, ___ _,  _   _ _,_ _  _, ___ _,  __,
                            |  |_| | |_) (_   |  \ |   | | |_| | (_   |  |   |_
                            |  | | | | \ , )  |   \|   |/\| | | | , )  |  | , |
                            ~  ~ ~ ~ ~ ~  ~   ~    )   ~  ~ ~ ~ ~  ~   ~  ~~~ ~~~
                                                 ~'

To start a new game, type: new game
To load a saved game, type: load game
To delete a saved game, type: del game
To repair corrupted game file, type: gen game
Or, to quit the game, type: quit
new

There's been a murder in Small Town, U.S.A., and the STPD needs your help. Bobby had so much to live for and the whole
town is still shocked by his gruesome murder. To figure out whodunit, collect clues and talk to residents around town.
The first responders have already taken care of his body and sent it down to the Morgue. They've also taken the liberty
of jotting down all the residents' names for you so you can cross people off your suspect list when you're sure they
didn't do it. Once you have gathered enough evidence to pinpoint the murderer, arrest him/her and if you're correct,
you'll be crowned a local hero. If you get stuck, type in 'hint' or 'help' for clues.

Good luck to you, detective- you're gonna need it...

You are currently in The Bar

What would you like to do? layout
                         _________________________________
                        |               | |               |
                        |      BAR      | |    MORGUE     |               N
                        |_______________| |_______________|            W  +  E
 _______________________|               | |               |               S
|  TRENT'S  |  TRENT'S  |    TRENT'S    | |   PHARMACY    |
|  BEDROOM  |  LIVING   |  FRONT PORCH  | |               |
|___________|__ROOM_____|_______________| |_______________|
 _______________________|               | |               |
|  MOOSE'S  |  MOOSE'S  |    MOOSE'S    | |    POLICE     |
|  BEDROOM  |  LIVING   |  FRONT PORCH  | |    STATION    |
|___________|__ROOM_____|_______________| |_______________|
 _______________________|               | |               |_______________________
|  SAM'S    |  SAM'S    |     SAM'S     | |    BETH'S     |  BETH'S   |  BETH'S   |
|  BEDROOM  |  LIVING   |  FRONT PORCH  | |  FRONT PORCH  |  LIVING   |  BEDROOM  |
|___________|__ROOM_____|_______________| |_______________|__ROOM_____|___________|
 _______________________|               | |               |_______________________
|  BOBBY'S  |  BOBBY'S  |    BOBBY'S    | |   CLAIRE'S    |  CLAIRE'S |  CLAIRE'S |
|  BEDROOM  |  LIVING   |  FRONT PORCH  | |  FRONT PORCH  |  LIVING   |  BEDROOM  |
|___________|__ROOM_____|_______________|_|_______________|__ROOM_____|___________|
             |         |
             | BOBBY'S |
             |_OFFICE__|


What would you like to do?
```

Figure 1: shows the start of game after issuing "layout" command**.**

The following describe the file structure of the game files loaded and played. Game files are loaded from text files with .json extension located in a top level folder labeled "new_game". The main files are the room files, in the Room folder, which have a json structure representing the state of Room objects in the game. As a room file is loaded, a

Room object is instantiated and manipulated. Figure 2 shows the contents of a room file with key variables used in a room object. The room file contains such variables as the room name, lond description (ld), short description (sd), a list of several other objects as well as pointers to other rooms.

```
{
    “room”: {
        "polymorphic_id": 1073741824,
        "ptr_wrapper": {
            "id": 2147483649,
             "data": {
                "name": "samFP",
                "displayName": "Sam's Front Porch",
                "ld": "You stand before Sam's house. The house, blue with white trim, seems well
                            cared for. In the front is a lawnmower \nresting amid a half-mown lawn.
                            You hear some wind chimes sound from the porch as birds play in a small
                            birdbath \npartially visible around the side of the house. A large oak tree
                            towers up from behind the house, and you bet your \nbottom dollar it has a
                            tire-swing. An orange cat lay sprawled across the warm concrete porch, it's
                            desire to relax \nkeeping the curiosity for the nearby birds at bay.\n",
                "sd": "You stand at Sam's porch. The chimes continue to sound melodically, the cat
                            remains in a heavy laze that doesn't come \nclose to being disturbed by
                            your nearby movement.\n",
                "hint": " - Look at the wind chimes\n - Look at the birdbath\n - Look at the cat\n - Go
                            into the house (west)\n - Go to Moose's house (north)\n - Go to Bobby's
                            house (south)\n - Go to Beth's house (east)\n",
                “visited”: 0,
                "myObjects": [ … ]
                "myRequiredObjects": [ … ],
                "myPeople": [ … ]
                "north": { … }
                "south": { … }
                "east": { … }
                "west": { … }
             }
        }
    }
}
```

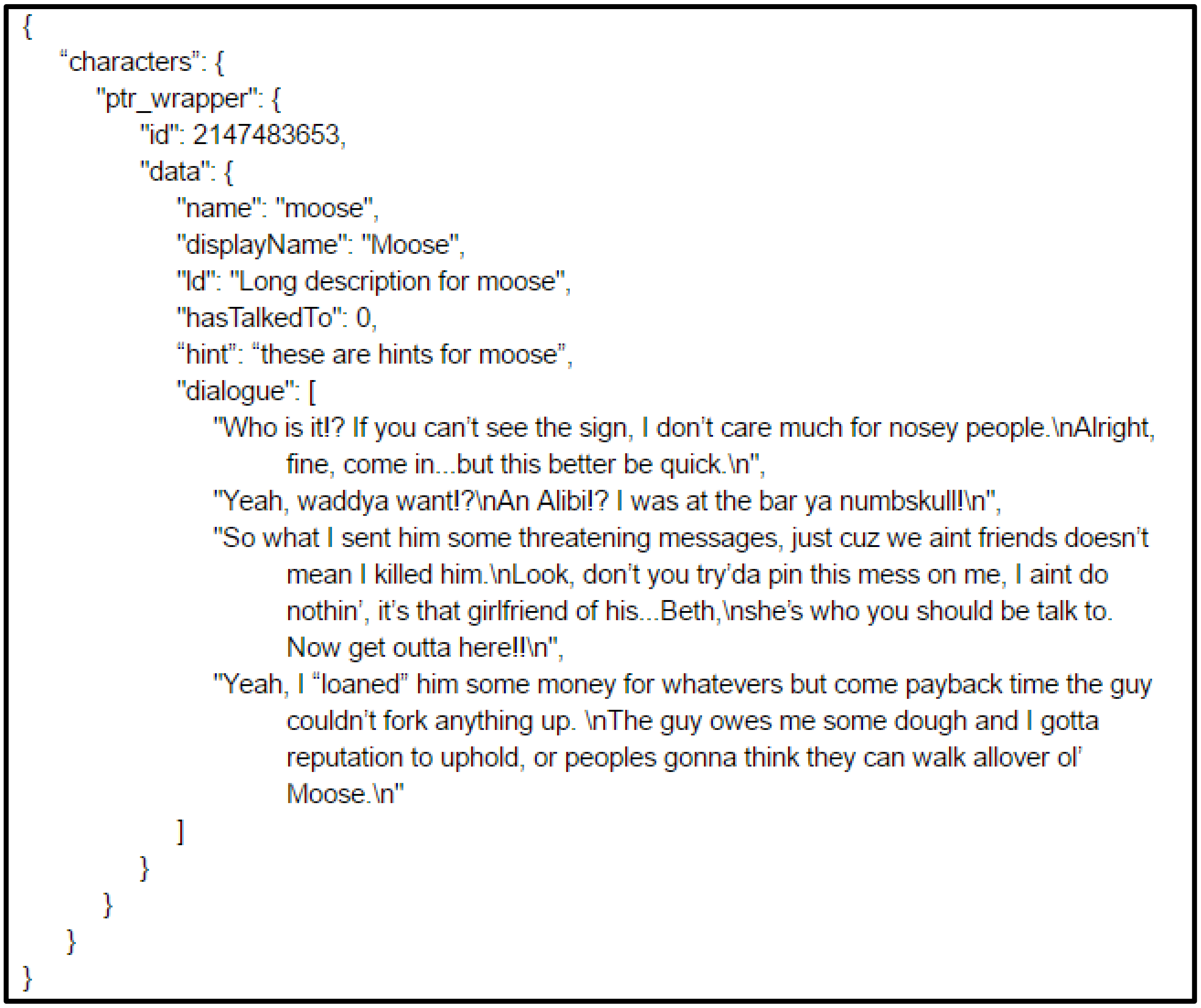
Figure 2: shows an example of a Room json file which contains key variables (some fields collapsed due to size) from a Room object**.**

Next is the player’s file, detective, which contains the state of player at startup and save time. Similar to the room files, the detective data is also represented in json format and holds information such as other objects possessed by the player, a list of suspects and the current room. Once a detective file is read into the game, it’s data is used to create an object to represent the player. As the game proceeds, data in the player object gets updated/manipulated and is preserved, if the game is saved, by writing it out to a file specified by the player. Figure 3, below, shows the content of a sample player file.

```
{
    “detective”: {
        "myObjects": [ … ],
        "suspects": [ … ],
        "currentRoom": {
            "polymorphic_id": 1073741824,
            "ptr_wrapper": {
               "id": 2147483651,
               "data": {
                   "name": "bar",
                   "displayName": "The Bar",
                   "ld": "You step into the Thirsty Whistle, the aura of death still lingering about. The
                           establishment has all the expected \ntrappings of a small town bar, with
                           dated decor, a lack of windows, and a faux taxidermized creature one could
                           at least \ngather was part eagle and part snake. The place was barren,
                           death has that affect. Along the walls are booths, with \nsome round tables
                           and accompanying chairs around the center. To your left, a stage that is
                           clearly the focal point of \nkaraoke Wednesdays. To the right is the actual
                           bar. Behind it you see what must be the bartender. In the back you see a
                           \ncorner booth sectioned off with police tape, and nearby a small hall with a
                           door to a closet. To the East is Main Street.\n",
                   "sd": "You enter back into the Thirsty Whistle. Trent stands behind the bar
                           working with inventory. To the north is a small \ncloset. To the East is Main
                           Street.\n",
                   "hint": " - Talk to Trent\n - Look at the trash\n - Look at the stuffed creature\n -
                           Look at the chair\n - Look at the table\n - Look at the booth\n - Go to
                           Trent's house (south)\n - Go to the Morgue (east)\n - Drink a beer\n",
                   “visited”: 0,
                   "myObjects": [ … ]
                   "myRequiredObjects": [ … ],
                   "myPeople": [ … ]
                   "north": { … }
                   "south": { … }
                   "east": { … }
                   "west": { … }
               }
            }
        }
    }
}
```

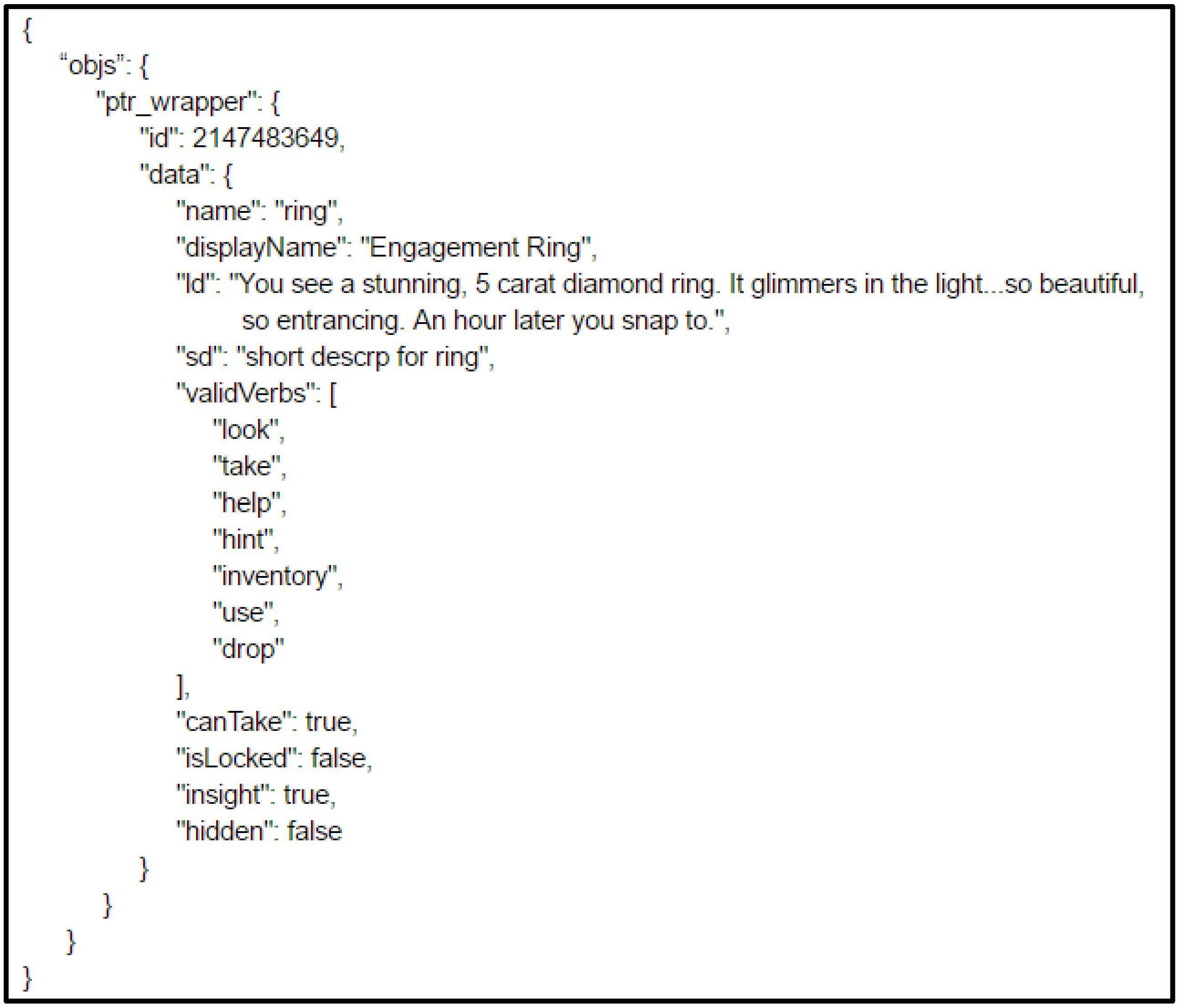
Figure 3: shows an example of a Detective json file which contains key variables (some fields collapsed due to size) from a Detective object**.**

In addition to the room and detective files, character and objs files were also saved in game folders to keep a complete set of the game state. The Character and Objs files hold pertinent state information for the respective objects in the game and figures 4 and 5 show a sample of each element's file.



```
{
    “characters”: {
        "ptr_wrapper": {
            "id": 2147483653,
            "data": {
                "name": "moose",
                "displayName": "Moose",
                "ld": "Long description for moose",
                "hasTalkedTo": 0,
                “hint”: “these are hints for moose”,
                "dialogue": [
                    "Who is it!? If you can’t see the sign, I don’t care much for nosey people.\nAlright,
                            fine, come in...but this better be quick.\n",
                    "Yeah, waddya want!?\nAn Alibi!? I was at the bar ya numbskull!\n",
                    "So what I sent him some threatening messages, just cuz we aint friends doesn’t
                            mean I killed him.\nLook, don’t you try’da pin this mess on me, I aint do
                            nothin’, it’s that girlfriend of his...Beth,\nshe’s who you should be talk to.
                            Now get outta here!!\n",
                    "Yeah, I “loaned” him some money for whatevers but come payback time the guy
                            couldn’t fork anything up. \nThe guy owes me some dough and I gotta
                            reputation to uphold, or peoples gonna think they can walk allover ol’
                            Moose.\n"
                ]
            }
        }
    }
}
```

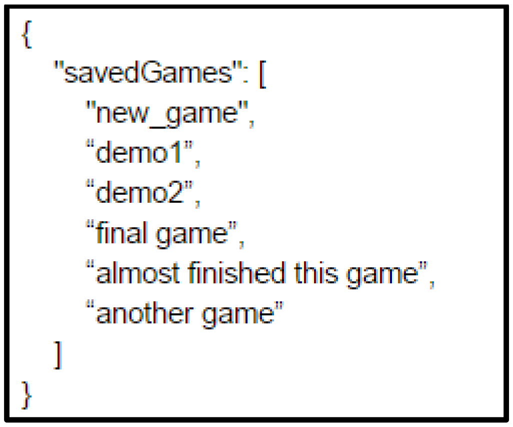
Figure 4: shows an example of a character json file which contains key variables from a Character object**.**



```
{
    “objs”: {
        "ptr_wrapper": {
            "id": 2147483649,
            "data": {
                "name": "ring",
                "displayName": "Engagement Ring",
                "ld": "You see a stunning, 5 carat diamond ring. It glimmers in the light...so beautiful,
                       so entrancing. An hour later you snap to.",
                "sd": "short descrp for ring",
                "validVerbs": [
                    "look",
                    "take",
                    "help",
                    "hint",
                    "inventory",
                    "use",
                    "drop"
                ],
                "canTake": true,
                "isLocked": false,
                "insight": true,
                "hidden": false
            }
        }
    }
}
```

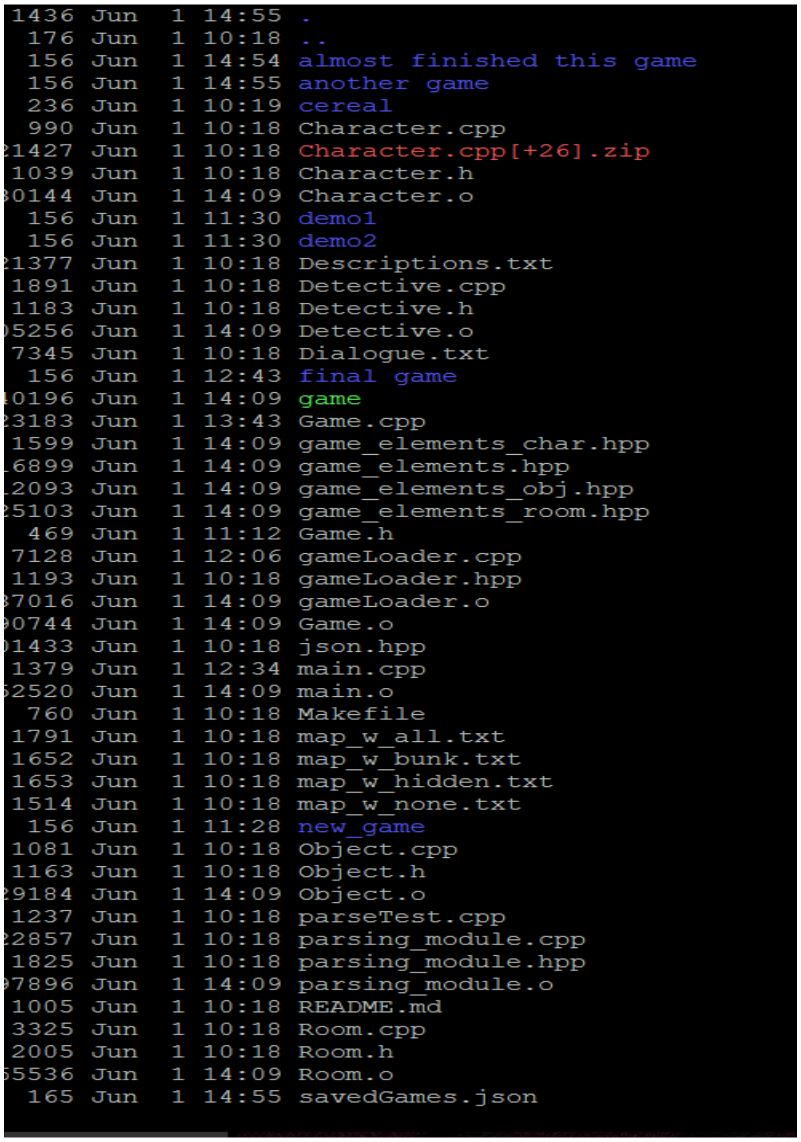
Figure 5: shows an example of an Object json file which contains key variables from an Object object**.**

The last file used by the game is the savedGames.json file which is used for housekeeping on files that are saved or removed. Not only does this help keep track of our list of games, but it also tells us where to load said game states from, as shown in figure 6. Figure 7 shows the corresponding Linux environment where the games are stored.



```
{
   "savedGames": [
      "new_game",
      "demo1",
      "demo2",
      "final game",
      "almost finished this game",
      "another game"
   ]
}
```

Figure 6: shows an example of a savedGames json file which contains a list of saved game files. “new_game” is always the default game**.**



```
1436 Jun  1 14:55 .
 176 Jun  1 10:18 ..
 156 Jun  1 14:54 almost finished this game
 156 Jun  1 14:55 another game
 236 Jun  1 10:19 cereal
 990 Jun  1 10:18 Character.cpp
21427 Jun  1 10:18 Character.cpp[+26].zip
1039 Jun  1 10:18 Character.h
30144 Jun  1 14:09 Character.o
 156 Jun  1 11:30 demo1
 156 Jun  1 11:30 demo2
21377 Jun  1 10:18 Descriptions.txt
1891 Jun  1 10:18 Detective.cpp
1183 Jun  1 10:18 Detective.h
)5256 Jun  1 14:09 Detective.o
7345 Jun  1 10:18 Dialogue.txt
 156 Jun  1 12:43 final game
10196 Jun  1 14:09 game
23183 Jun  1 13:43 Game.cpp
1599 Jun  1 14:09 game_elements_char.hpp
16899 Jun  1 14:09 game_elements.hpp
12093 Jun  1 14:09 game_elements_obj.hpp
25103 Jun  1 14:09 game_elements_room.hpp
 469 Jun  1 11:12 Game.h
7128 Jun  1 12:06 gameLoader.cpp
1193 Jun  1 10:18 gameLoader.hpp
37016 Jun  1 14:09 gameLoader.o
)0744 Jun  1 14:09 Game.o
)1433 Jun  1 10:18 json.hpp
1379 Jun  1 12:34 main.cpp
52520 Jun  1 14:09 main.o
 760 Jun  1 10:18 Makefile
1791 Jun  1 10:18 map_w_all.txt
1652 Jun  1 10:18 map_w_bunk.txt
1653 Jun  1 10:18 map_w_hidden.txt
1514 Jun  1 10:18 map_w_none.txt
 156 Jun  1 11:28 new_game
1081 Jun  1 10:18 Object.cpp
1163 Jun  1 10:18 Object.h
29184 Jun  1 14:09 Object.o
1237 Jun  1 10:18 parseTest.cpp
22857 Jun  1 10:18 parsing_module.cpp
1825 Jun  1 10:18 parsing_module.hpp
97896 Jun  1 14:09 parsing_module.o
1005 Jun  1 10:18 README.md
3325 Jun  1 10:18 Room.cpp
2005 Jun  1 10:18 Room.h
55536 Jun  1 14:09 Room.o
 165 Jun  1 14:55 savedGames.json
```

Figure 7: Linux environment view of saved game files.

The System & Software:

[Files' portion]

Our game has a static set of files which describe the state of a default/new game. These files are organized within a top-level folder new_game/, as well as sub-folders Character/, Room/ and Obj/, to keep various elements' files separate. Upon starting a new game, the static files are read into the program and instances of all objects - Room, Character and Objs - are created based on the contents of the files. At anytime during the game, the user can elect to save a game state by typing "savegame" and providing a string name for the game to be stored as. This event causes the program to create a game folder structure, labelled with the user string name, then proceed to write the state of all objects to their respective folders and files. A user can also choose to load a previously saved game with "loadgame" command which presents the user with a list of saved games to select from. There is an option to delete a saved game when the program is first run. With this option, a user can choose from the list of saved games which the remove. Removal of a game deletes the entire folder for the specified game.

What We Used:

- Google Hangouts for communication
- Google Docs/Drive for collaboration on text deliverables
- Cloud9 for code repository
- Flashback Express for video recording and editing
- Notepad++ for code editing
- Flip servers for running local builds and verification
- First used jsoncpp for serialization/deserialization – required specific gcc version
- Cereal library for object serialization/deserialization – much better to use.
    - Cereal's use of smart pointers etc.

Jessica had previous experience with Cloud9, and given its collaborative-based setup, Tunde and Aaron felt it would be a great tool to use. Cloud9 provides a linux OS CLI that you can test code on, as well as data storage, all in an environment we all can actively modify together. It helped us to ensure we were all using the same files as we modified and versioned. We did have an issue with Cloud9 that was difficult to truly understand, but issues can arise when multiple people keep files open and others save or quit or some combination that led to the deletion of changes and, at times, the entire contents of a file. This took a while to realize, and while we may have learned to prevent the issue, it still arose towards the very end of development.

Notepad++ turned out to be a very power tool for working with code on the Flip servers. We used Flip not only to ensure that the project compiled and ran on it, but to also test individual sections of code before integrating them fully onto Cloud9. Two main benefits to Notepad++ came with plugins. One allowed us to connect to our OSU Flip accounts and edit files directly in Notepad++ instead of using VIM. We have no issues with VIM, but it was easier to use Notepad++. Additionally, Notepad++ has another plugin which lets you compare two similar files side by side as it highlights lines that differ between them. This proved helpful when performing version control for cloud9 and making sure the files we were each individually using were the most up-to-date ones.

A description of what each Team member accomplished.

[Aaron] - Command Parsing Developer

- Created a Command Parsing Module (CPM) from which the game would take in user input and extract the intent of it by a vocabulary of approved words for use separated into relevant nouns, verbs, and prepositions. The module deals with upper and lower case words so the player doesn’t need to. It has the ability to deal with certain things that require more than one word to describe, such as “Trevor’s Wallet” or “Trent’s House”, to distinguish between a location, object, or person when applicable.
- The CPM also contains the action() function used by the game. It stands as the link between the module and the game mechanics, where the previously extracted intention is passed into it and then the various verbs are acted upon.
- Implemented the function sandsOfTime(), which is the event driver for the game. The many if-statements within check for various changes throughout the game to cause their own as a way to represent time.
- Wrote nearly all of the Room, Object, and Character descriptions, as well as the initial dialogue for all characters.
- Designed the speak() function to allow for appropriate, dynamic dialogue with the characters in the game.

[Babatunde] - Data Format Developer

- Investigated the concept of serialization/deserialization of objects in C++. The original idea was to work with JSON objects in C++ and then save those objects to files. After several experiments with different libraries and APIs, it became clear that we did not need to work with JSON objects as we planned, but rather serialize our C++ objects instead. This is where the the Cereal library comes into play.

- Implemented serialization/deserialization methods for our classes – Each class that we were interested in serializing required a serialize() method to archive the variables of interest. This method was implemented for Room, Character, Detective and Object classes.
- Implemented save game objects methods – wrote routines to support the savegame mechanisms which include prompting user for a name, checking if name already exists and serializing all objects to their respective files and folders.
- Implemented load game objects methods – wrote routines to support loadgame mechanism which include providing the user with a list of current saved games to load into memory. Handled reading all files from a game folder and creating object instances based on their content in memory, AKA deserialization.
- Implemented delete game objects methods – wrote routines to support deleting a game folder and removing it from savedGames.json.
- Implemented structure for creating game elements – setting up a MACRO framework for easily instantiating new rooms, characters and objs. Organized our set of objects into a packaged form that made it easy to pass around various functions.

[Jessica] - Engine Developer

- Created the Character class and implemented functionality to tally the number of times the detective has spoken with that character, to store that character's dialogue options, and to store the character's information.
- Created the Detective class and implemented functionality to keep track of hidden rooms discovered, to tally drink and arrest counts, to add, remove and list items from inventory, to add, remove and list names from the suspect list, to track the current and previous rooms of the detective, and to store the detective's information.
- Created the Room class and implemented functionality to add, remove and list the objects and people located in that room, to create directional connections between rooms (n/s/e/w), and to store the room's information.
- Created the Object class and implemented functionality to track whether the object can be taken from its current location, whether it is locked/unlocked, whether it is useful in the grand scheme of the game, whether it is hidden/visible, and to store the object's information.
- Created the Game class and implemented configGame() to connect rooms, and place people and objects in their appropriate locations.
- Created main function to introduce the user to the game and call on the proper init/load game methods.

- Used the game_elements setup (created by Tunde) and descriptions (created by Aaron) to populate/instantiate all Characters, Objects, Rooms and the Detective.
- Implemented hint verb - printing out the people, objects, takeable objects, and connections for a given room.
- Implemented layout verb - created all maps and wrote functionality to call the correct map based on which hidden rooms have been discovered.
- Performed unit and system tests to report bugs and fix them.

Project Deviation from Initial Plan:

In addition to the 14 verbs we originally planned to use, we added 5 more to enhance gameplay and removed 2 which were no longer useful. We decided not to implement ‘enter’ and ‘break in’ because ‘knock’ and ‘use <lockpick>’ made them redundant (breaking in is the same as using the lockpick on the door).
The verbs we added were: **‘layout’** (to make navigating easier), **‘insight’** (to let the player see which items are important), **‘edit’** (to utilize the suspect list efficiently), **‘drink’** (to add some humor to the game), **‘hack’** (a secret verb used to crack any technology).

We also added a significant number of objects so that each room has several items to inspect and examine.

We originally considered using the JSON for Modern C++ API, but after playing around with it, decided we were much better off using cereal, a header-only C++11 serialization library. Cereal allowed us to take c++ standard library types and reversibly turn them into JSON. Early versions of our executable were only compatible with g++ versions 4.9 or higher because the API required it, which was problematic. After switching to cereal, we were able to eliminate that restriction and run our game on the flip server without specifying a version of g++.

As the project developed, we got rid of the verb and dialogue classes. This came in large part due to how the parsing module would deal with the verbs and their functionality, instead of using composition to create a group of verb objects for each individual object out there. We felt this saved in complexity and space, and resulted in enhanced efficiency.

Conclusion:

As you can see, we created a command line interface game with a murder mystery theme that allows users to control the detective and navigate the game using natural language. We wrote a range of functions that dealt with parsing input, giving meaning to it, giving structure to the game, saving and loading data, and tying it all together into game engine functionality. We used a set of unique verbs to control the way in which the player interacts with the game world around them, and a set of usage instruction to illustrate the breadth of these interactions with the game.

We explored different tools, such as Cloud9 and Notepad++ to collaborate and edit files with, and applying the third party library CEREAL to implement saving and loading data with JSON in C++. We used the well-known open source debugging tool GDB to examine variables and work through problems that arose in our code.

The figures show both a user perspective of the command line interface, as well as how we structured various game objects and saved their state in JSON format for generating new games, loading, and saving.

Creating this project was both fun and challenging. The experience showed us how much a project can change over the duration of its creation, and how much that change is influenced by the people involved. Different methodologies quickly present themselves as we share viewpoints and ideas. Having a well thought out plan helped guide the project on a desirable path, but our experiences along the way led to unforeseen changes in game structure, implementation of certain functionalities, and the way in which we deal with user input.