COMP3207 2017/18 ECS-MUD

Richa Ranjan

University ID number: 29757282
ECS User Name: rr2n17

ABSTRACT

This coursework deals with the implementation of a *Multi-User Dungeon* (MUD). MUD is a multiplayer real-time virtual world, described entirely by text. The application is deployed on Heroku and is up and running on the url below:
Heroku application address: https://comp3207-cw1-1718-rr2n17.herokuapp.com/
GitHub Repository address: https://github.com/ECS-COMP3207-1718/rr2n17-CW1
Note: The git repository is hosted privately on the above url, thus users would be challenged for credentials.

Node.js and Javascript: A Good Fit?

For the implementation of MUD, I have used Node.js as the application Runtime environment, and Javascript for the application code. Node.js being open-source and light-weight are key advantages. Javascript is one of the most widely used, and easily understood scripting languages. The basic functionalities of this application were easy to understand and were a big help to get started with the implementation of other features. To sum up, this has definitely been a good fit for deploying MUD.

Challenges encountered

In spite of the easy implementation of the basic MUD features, there were obstacles, certainly. Here are some challenges that I faced while programming this application:
1) **Callbacks:** The callback method used across the application was not too efficient in my view, as it was difficult to access their members outside of the callback function.
2) **examine:** From the description, it was not clear as to what exactly is to be done if the object we are trying to examine is a PLAYER, a THING, or even an EXIT. I made a few assumptions (explained further) based on my understanding of the MUD functionalities.
3) **"@link":** In general, the *@link* command was challenging, as it required a lot of cross referencing, and there were a number of cases to be handled. Consequently, the testing was also tedious, as all of the underlying logic and constraints were to be taken care of. This command required the maximum effort.

Assumptions

1) **"@set" :** I can set flag for an object, only if I own it.
2) **examine:** For the *examine* command, these are a few assumptions that I made:

(i)**PLAYER:** If a player is trying to examine another player, I have programmed my application to display a "*Permission denied*" string on the game console. This is because, the examine command displays all the details of the player, including the password, which, in my view, was not right. That is why, if a player types `examine <player>`, and if the *player* tries to examine any other player, according to me, the permission should be denied.

(ii)***'me'* and *'here'*:** In the commands description of examine, the cases of *me* and *here* have not been specifically asked for. I have handled those cases in my code, as they are keywords, and if a player types `examine me` or `examine here`, the controller should display the player's details and the player's location details respectively. Additionally, there is a limitation to the program regarding this, which I have described in the next section i.e. "Limitations".

(iii) ***ownerId* check:** As the description of the *examine* command says that it can be carried out only on objects you own, that are visible to you, I have included a check on the *ownerId* of the object being asked for examining.

(iv) **examine 'EXIT':** Specifically, examine conditions for an EXIT have not been mentioned. So, I have programmed the application to behave in the same way, as the *examine* for a 'THING' would behave.

Limitations

In terms of the functionality, given below are a few limitations of the application, in my view:

1) **"@create":** When a player tries to create an object, if an unusual name like "%$#" is entered, a message saying "*invalid name*" should be displayed on the controller, but that does not happen. This means, that the function *isNameValid* in the *Predicates.js* file, is defined in such a way that it does not handle these special characters in a name.

2) **"@name":** Executing '`@name <object>= `' removes the reference to that object. Further, there is no way to access that object other than using its id.

3) **look:** This command could not be used in case of the *examine* command. The results required in case a player tries to examine a 'ROOM' or a 'PLAYER", was to display the details of the ROOM/PLAYER, followed by their respective contents in the format: [`type name`]. Thus, *look* could not be reused, as it displays only the description of the object. As a result, a similar piece of code had to be re-written.

4) **findPotentialMUDObject :** This function has been defined to find the objects whose names partially match with the name being searched for, but with a condition, that the object has to be in player's visibility. So, when this function was used in the *examine* command, the "failure" string argument for this function call was "`examineUnknown`". This might be logically true but, in *examine* because of this functionality, sometimes, when a player tries to examine an object which is not visible to him, the message displayed on the controller is "*Examine what?*", instead of "*Permission denied*", which would have been logically more appropriate. In particular, outside of *Commands.js*, I would also change the regular expression for name validation in *Predicates.js* while creating objects. I would define it in this way:

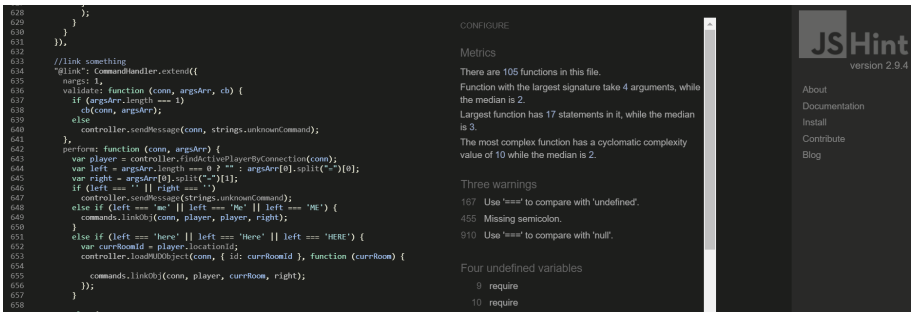
```
return str !== undefined && /^[^ -~]+$/.test(str) && str !== "me" && str !== "here";
```

Testing

1) Including "`console.log()`" to check the values of the variables at various stages.
2) Generating the *sample transcript* and running each command to check against their expected functionalities.

Commands	Strings	Commands executed	Terminals	Expected output	Actual output	Result
inventory	carryingNothing	`inventory`	Terminal 1:	`You aren't carrying anything.`	`You aren't carrying anything.`	Successful
take	cantTakeThat	`take Television`	Terminal 1:	`This Television is firmly attached to the wall!`	`This Television is firmly attached to the wall!`	Successful
			Terminal 2:	`rr23 tries to steal the telivision but it's firmly attached to the wall!`	`rr23 tries to steal the telivision but it's firmly attached to the wall!`	Successful
	taken	`take lamp`	Terminal 1:	`You have successfully taken the lamp!`	`You have successfully taken the lamp!`	Successful
			Terminal 2:	`rr23 has successfully taken the lamp!`	`rr23 has successfully taken the lamp!`	Successful
inventory	youAreCarrying	`inventory`	Terminal 1:	`You are carrying: lamp`	`You are carrying: lamp`	Successful

3)*JSHint*, an online Javascript testing tool was used to check the code quality:



Additionally, I have used "`try` and `catch`" for Exception Handling in some parts of the code.
Effort: Amount of time spent on the coursework: approximately 55 hours.

References

1. https://secure.ecs.soton.ac.uk/noteswiki/w/COMP3207-1718-cw1-guide
2. https://www.w3schools.com/js/