Developing “space escape” text adventure

When coming up with the idea for my text adventure I wanted a setting in which moving between spaces back and forth made sense. With this, I came up with the idea of using a small environment inside a spaceship which would encourage moving between spaces and provide easy explanations for certain spaces being locked and requiring keys to continue. With the space setting set up the driving reason for the player was then decided to be an escape drawing inspiration from escape rooms and is normally seen as a good driving force especially when the reason is unknown as the enemy is never seen in my text adventure allowing the player to create their own image of them. Two main objectives were then drawn from this: get to the escape pod and destroy the ship (using the reactor)

With this decided I then started designing what approach I would use to create it. I settled on using a data-driven system with a text file for each room in the ship totalling 8 rooms (Cryopod, Cryopod room, hallway, Medbay, armoury, captains quarters, reactor room and escape pod room). This is also the average of the recommended amount of rooms mentioned in the brief. to create these text files I also wanted to make a system which streamlines the room creation process to help me visualise the information. To achieve this I created a separate python script which can be used to create rooms and populate them with items.

This system allowed me to have potentially a near infinite amount of items per room and a maximum of four doors per room (north east south west). It also allowed items to be marked as keys, items to end the game when used and custom interaction phrases for using items allowing for potentially endless interactions. Having this functionality was useful for meeting the assignments brief requirement for item commands such as “look at item”. There are also a total of 8 interactable items in the game including the keys.

With this system in place, I then created the rooms and required another system for setting the default room to load and what happens when a game-ending item is used. To solve this I created a default room file which will provide this information to the text adventure.

The next step was to create the actual text adventure python file, starting with reading these text files. With the way they are created some information can be assumed such as key requirements are always at the bottom of the file allowing for them to be checked easily, room directions always being stored in North East South West and finally the room description being the first line followed by the number of items in the room. The number of items in the room will then be used to read each item with each one using 5 lines. This meant that rooms can be read relatively easily and put into lists containing dictionaries.

An issue however with having all the interactions and information in the text files was that the game could not be completed or even move out of the first room until all the user input interpretation was finished. This meant up until the application was practically finished I was unaware of any issues in rooms other than the starting room.

The main gameplay loop for the text adventure uses 6 main input options: search room, which was used to display all the items in the room, interacting with items which uses each items specific interactions to use them, help which would display current commands (will not display item ones until the room is searched), check inventory which will display the keys the player has and finally opening doors which also accepts commands such as “go north” using the pre-set order of doors from the text files. The game will also say when a command is not understood as mentioned in the brief.

The gameplay loop itself consists of 2 main while loops the first being while the game has not been finished (triggered by a game-ending item being used). Followed by a second enclosed while loop which is used to make actions within one room. Opening a new room would exit the second while loop and use the first to load up the new room. This design avoids repetition of code and allows for as many commands the user wants to enter until the game ends.

For user input and reading text file data, I decided to use a system which removes all punctuation, line breaks and sets it to lower-case. I then used a comparison to find specific words in user inputs to decide the next steps. This system prevented issues where I may have capitalisation for some items or too many spaces in user input, it also prevents any errors from being caused as a result of user input.