**Zuul Station – PPA Assignment 2 Report**

Harry Tennent

K1890723

**Zuul Station**

Zuul Station is a text-based adventure game. The setting of the game is a crashed space station, hundreds of thousands of miles away from Earth. In the crash, the oxygen supply to the station was cut off and now there is a limited amount of oxygen left until the station runs out. The aim of the game is to find some codes for the escape pods and take them to the escape pod room. Once this is done, the player has won. The player has 3 minutes to do this before the oxygen for the station runs out. However, there are items in the game, such as oxygen tanks, that can add different amounts of time to the game timer and delay the end of the game, as long as you can find them. The player has a backpack with a limited capacity and can pick items up until this capacity is reached. If the player wants to pick something up but they don’t have enough space, then they can use the item, they can drop items into the room, or they can give the items to other characters. These characters are randomly created and added to the game. They can move between rooms and have a backpack which stores the items that the player gifts them. There are 10 rooms in the game. There is a: living quarters, main hall, laboratory, reactor, medical bay, kitchen, store room, gym, escape pod room, and a transporter. The player can walk throughout all of these rooms freely. The transporter room is different from the other in the fact that it is “broken” and transports the player to a random room in the space station. The player has to find the escape pod room, along with the codes, to win.

**Room Map:**

Main Hall

Escape Pods

Transporter

Kitchen

Medical Bay

Reactor

Gym

Store Room

Laboratory

Living Quarters

**Base Tasks:**

* **The game has several locations/rooms.**
  + The ‘zuul-better’ project already had a number of rooms implemented in it as well as a separate Room class which made making new rooms easier. To complete this task, I changed the room pre-existing in the game to suit the theme of the game I was making. By the end of creating the game, I had 10 rooms which were: a living quarters, main hall, laboratory, reactor, medical bay, kitchen, store room, gym, escape pod room, and a transporter room. The only room that was different from the rest was the transporter room which I implemented as part of one of the challenge tasks.
* **The player can walk through the locations.**
  + This was already implemented in the code I was given and I just adding exits to each of the rooms that I added so that the player could move between rooms.
* **There are items in some rooms. Every room can hold any number of items. Some items can be picked up by the player, others can’t.**
  + In order to implement this, I created a new class to represent an item. Every item has a name, a weight, a boolean value for whether it can be picked it up, and a bonus time associated with using the item. All of this data was stored in a hash map where the key was a String of the item name, and the value was an array list of strings. Within the array list, I stored the item weight, the boolean value of whether it can be picked up and the bonus time. These values were stored as strings because the ArrayList can only take 1 type for all of its values. The values were always in the same order which made it easy to retrieve specific values for a particular item when required. Initially, I had 2 separate classes for handling items. One stored all of the data about each of the different types of items and another class represent a single item that was initialised by that first class. However, I found I was duplicating many methods and it made the implementation of methods in the Room classes complex because the Room class had many Item objects, but it needed to retrieve data from the Items class as well. As a result, I decided to simplify it down to one class. I created methods to set random values to the fields of the item class so that when the Room class is created, it can create however many random Item objects it needs. As well as this, I made methods to check a string of an item name is actually an item, and methods to retrieve specific data about specific items. For example, an accessor method to get the weight of an oxygen tank.
  + The Room class was linked to the Items class. When a room was created it created up to 5 random items per room but this can be edited so any room could hold any number of items. These items were put into a hash map for that particular room with their name as the key. The keys of the hash map are then printed when the player enters the room to show which items are in the room.
  + The Game class was linked to the Items class. The pickupItem(), useItem(), dropItem(), and giveItem() use the Items class. Items in the room can be picked up by the player and added to their backpack, provided they have enough space in their backpack and that the item has been initialised so that it can be picked up. Using an item adds the bonus time to game timer. Dropping the item returns it to the room and removes it from the player’s backpack. Give item removes it from the player’s backpack and gives it to a character.
* **The player can carry some items with him. Every item has a weight. The player can carry items only up to a certain total weight.**
  + To implement this functionality, I created a separate class called Player. In this class I created a “backpack” field which was an array list of strings. Accessor methods allowed the Game class to get individual items from the backpack at specific indices or get them by name. There are also mutator methods so that the Game can add items to the player’s backpack, provided that there is enough space available. To check the space in the backpack, I made constant “MAX\_WEIGHT” which is initialised to 100. Then, when items are added to the backpack, I check the current backpack weight against this constant to make sure there is enough space. If the player wants to drop or give an item, the current weight is decremented. If there is not enough space in the backpack, then the player cannot pick up the item.
* **The player can win. There has to be some situation that is recognised as the end of the game where the player is informed that they have won.**
  + To implement the win condition, I made a method in the Game class which tests the condition every time an item is used. For the player to win, they must find the codes in the rooms and then take them to the escape pod room where they then use them and win the game. The testWinCondition() method checks if the player is in the escape pod room and whether they are trying to use the codes item. If both of these conditions are true, then the winCondition variable is set to true. Then, the value of this variable is checked in the main while loop of the play method(). If the winCondition is true, the loop exits, a congratulations string is printed and the game exits.
* **Implement a command “back” that takes you back to the last room you’ve been in.**
  + To implement this feature, I added a field to the Room class which tracked the previous room. When the player moves into a new room, the previous room field is to current room using a mutator method in the Room class. Then, when the back command is received, a method in the Game class is run which checks that there is a previous room to go to, and then moves the player into the previous room.
* **Add at least four new commands (in addition to those that were present in the code you got from us).**
  + The new commands I implemented were: pickup, use, inventory, drop, back, timer, give. To add these commands, I edited the array of commands in the CommandWord class. Then I had to extend the processCommand method in the Game class to recognise each of these new commands. When a command is parsed using one of these commands, the processCommand method calls another private method in the Game class to execute that command’s functionality. The pickup command allows the user to take items from the room and them to their backpack. The use command adds a bonus time to game timer if there is a bonus time for the particular item. The inventory command prints out a string of all of the items in the player’s backpack to the console. Drop allows the user to return items to room and remove them from the backpack which is useful if the player wants to pick up items but they don’t have enough space in their backpack. I explained the functionality of the back command above, but it moves the player back into the room they had previously been in. The timer command prints out a string of the time remaining before the oxygen supply in the space station runs out. The give command is part of one of the challenge tasks and allows the player to give an item in their backpack to one of the other characters in the game.

**Challenge Tasks:**

* **Add characters to your game. Characters are people or animals or monsters – anything that moves, really. Characters are also in rooms (like the player and the items). Unlike items, characters can move around by themselves.**
  + To do this, I created a new Character class. The class is similar to the player class because it has a name field, and a backpack. There is as a currentRoom field which holds which the room that specific character is in. There is also an array list of possible names so the name of the characters could be randomised. I included accessor and mutator methods for the name, and currentRoom fields. I also used the addBackpackItem from the player class to facilitate the functionality of the “give” command. The character class is created by the Game class. A method in Game makes a random number of characters up to a maximum. Another method prints out which characters are in the room with the player. To get the characters to move, I made a method called moveCharacter in Game, which randomly chooses an exit out of the available exits in the room the character is in and then sets the characters currentRoom field to the room through that exit. This is called every time the player moves so that all the characters can move about through the whole game.
* **Extend the parser to recognise three-word commands. You could, for example, have a command give bread dwarf to give some bread (which you are carrying) to the dwarf.**
  + I completed this task with the “give” command. This command allows the user to specify an item in their backpack and give it to a specified character in the same room as them. The item is removed from the player’s backpack and added to the character’s backpack. To work with three word commands, I extended the Parser class to recognise 3 words and return 3 the words or null if there were less than 3 words. Then in the give item method in Game, I check that there are 3 words in the command. If the player has the item they say, and they are in the same room as the character they want to give the item to, the item is exchanged. Otherwise, it fails and a string explaining why it didn’t work is printed to the user.
* **Add a magic transporter room – every time you enter it you are transported to a random room in your game.**
  + I created a new room called the transporter room. Then, in the method that handles the user moving between rooms, I included a conditional statement that checks if the next room is the transporter room. If it is, a string explaining to the user that they went into the transporter room is printed and a randomly selected room out of an ArrayList is selected to be the new current room for the player. The back command still works with this added functionality as it returns them to room they were in before they went into the transporter room.
* **Others. You can invent additional challenge tasks yourself.**
  + I also added a timer to the game. This was part of a separate class called timer. I added because it adds more pressure and fun to the game. The timer is initialised to only be 3 minutes but with the use of items, the player can make the game last longer while they try and find the codes. As I had not implemented a timer before, I used some code from Stack Overflow, which I have referenced and credited in the comment of the methods. I also extend the code from that code further so that it worked for the game. I made methods to print out the time as a string and add bonus time.

**Code Quality Considerations:**

* Coupling
* Cohesion
* Responsibility-driven design
* Maintainability

For each of the following code quality considerations, give and explain an example in your project where you considered it: coupling, cohesion, responsibility-driven design, maintainability.

A walkthrough of your game, consisting of the commands that need to be entered to complete/win the game.

Known bugs or problems (Note: for a bug in your code that you document yourself, you may not lose many marks — maybe none, if it is in a challenge task. For bugs that we find that you did not document you will probably lose marks.)

A copy of the source of all classes.