02/11/2020
K20082471
k20082471@kcl.ac.uk

Coursework 2: World of Zuul

Emre Salur

User Level Description

In my version of world-of-zuul, the aim of the game is to take items from different rooms to construct a time machine. There are two players, the main player and the professor who helps you to build a time machine. The players and items are created in a map that includes eleven rooms. The user controls the main player through a text panel and can go into different rooms, take items, drop items, and give items to other players with specific commands. Each item has a weight and the player can carry items to a certain limit. The player must find all the required items to build a time machine and give it to the professor in order to win the game.

Implementation Description

In the default world-of-zuul-better file, there are five classes. In my version, there are seven classes: Game, Room, Item, Player, Command, CommandWord, and Parser. The Game class is the main class in which the user must create an object of this class and call the play() method. The Game class implements classes Room, Item, Player, Command, and Parser. The Room class implements the Item class since each room can include multiple items. The Player class implements classes Item, Room, and Command to allow the player to change rooms, carry items, and take action according to the command. Additionally, the Parser class, which reads user input, implements the CommondWords to check the list of known commands.

02/11/2020
K20082471
k20082471@kcl.ac.uk

Base Tasks

- **The game has several locations/rooms:** The game already had multiple rooms and I added other rooms such as a bedroom, bathroom, magic transporter room, football facility, and food market by creating a Room object in the Game class.
- **The player can walk through the locations:** Already implemented in the code but I edited the goRoom(Command command) method to check if the player is in the magic transporter room or in the same room with another player.
- **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:** Created a separate Item that constructs the class with three parameters: the description(String), the weight(int), and whether it's pickable or not(boolean). The Room class implements the Item class by having an addItem(Item newItem) method. Each room has an ArrayList of type Item holding the items in the room and is updated when an item is either initialized, taken, or dropped.
- **There has to be some situation that is recognized as the end of the game where the player is informed that they have won:** When the player finds all the correct five items and gives them to the professor, the professor builds a time machine and the player wins.
- **Implement a command "back" that takes you back to the last room you've been in:** I created a Stack of type Room in the game class. A stack is a data structure that has similar purposes to an array or a list. The stack stores the rooms the player entered. The stack keeps updating when a player enters a room by pushing the latest room on top of the stack. So, when a player calls the "back" command, it pops the room on top of the stack and sets the current room to the previous room by peeking. By using a stack, the back command can be used multiple times in a row.
- **Add at least four new commands:** Five new commands: back, take, drop, items, give
  Back: takes the player to the previous room.
  Take: the player takes an item.
  Drop the player drops an item.
  Items: prints out the attributes of all the items that are carried by the player.
  Give: gives an item directly to another player.

Challenge Tasks

- **Add characters to your game. Characters are also in rooms (like the player and the items). Unlike items, characters can move around by themselves:** I created a separate Player class which constructs the player with the following parameters: name of the player(String), and the room of the player(Room). Also, each player has a maximum weight capacity(int), a current weight(int), and an ArrayList of Items which stores all the items the player is carrying. The professor moves to a different room after interacting with the player. Creating a player class allows adding more characters such as people or animals if we are aiming to develop the game even more.
- **Extend the parser to recognize three-word commands:** In my code, most of the commands can be done with two words but the "give" command requires three words. For instance, *give wood professor*. It follows an action-item-player format since there might be multiple items to give or

multiple players to give. To allow this, I created another String variable thirdWord in the Command class helping to return it or check if there is a third word. Furthermore, the scanner inside the Parser class gets the third word by checking if the tokenizer has the next string after the second word and returns the command.

- **Add a magic transporter room – every time you enter it you are transported to a random room in your game:** Added an if condition in the goRoom(Command command) method to check if the player enters the magic transporter room. If entered, the program uses a list that contains all the rooms and generates a random integer between 0 and the number of total rooms. Then, the player goes to a room with respect to the random index number in the room list.
- **Creating a magic cookie that increases the player's maximum weight capacity:** Created a cookie item in the food market room and if the players enter that room, he/she consumes the cookie which increases the maximum weight capacity by 5 kg. Finding this cookie helps the user to finish the game faster as he/she can carry more items all at once.

Code Quality Considerations

Coupling

We aim for loose coupling since one of the purposes of having different classes is to have different methods and variables that are not all dependent on each other. In my program, I chose the Game class as the superclass and aimed to implement most of the classes in there so there is less coupling between other classes. The Room class does not require information from classes Player, Command, CommandWords, or Parser even though a room can contain a Player or a specific command typed in the parser can change the attributes of a room. Instead, the Game and Player classes get the required command and make updates to the room accordingly. Thus, not all classes have to be directly related to each other and this results in lower coupling.

Cohesion

We aim for high cohesion so that a class or method is responsible for one single logical task. If I wanted to, I could have had only the Game class and write all my code there. But, that would have been unbelievably difficult to read and make changes to. In my version, each class has a unique purpose and has different methods. The Game class initializes all the other classes and starts the game, the Parser class reads user input, the Room class creates a room, and the Item class creates an Item, etc. Besides, the methods in each class have a unique purpose and are independent of each other. There is a different method to take an item, to drop or an item, to give an item, and to print all the items carried.

02/11/2020
K20082471
k20082471@kcl.ac.uk

Responsibility-Driven Design

Responsibility-driven design is the concept that each class should be responsible for manipulating or processing its own data. For example, the Room class handles the short description, long description, and the exits of each room created as well as its directions. The only way to access these characteristics, a class must create an object of the class. This design matches with the principles of Object-Oriented Programming and makes it simpler to organize a program.

Maintainability

Maintainability is a key component of programming since it is hardly the case where a program is finished after the first attempt. Most of the time, we as developers will work with others and continue to develop our program. Having low coupling allows us to keep making adjustments and improve. For instance, by having separate Item and Player classes we can create as many items or players without changing how the game works logically.

Walkthrough

There are two characters, a player and a professor. The user controls the player with a text-based panel. The player is spawned into an eleven room world. The player finds the professor in a lab, understands what items are required, starts searching for it while the professor goes to the office and waits for the player. The player goes to different rooms and takes items without exceeding the maximum weight capacity. If the player cannot take an item, he/she drops items or gives some of the items to the professor and comes back to the room. The player finds all the items. All pickable items are given to the professor. The text-based panel shows that the time machine is built and that the player has won.

Bugs or Problems

I have not faced any bugs or problems in its latest version but please let me know in the feedback so I can improve.

Sources

Barnes, David J., and Kölling Michael. *Objects First with Java: a Practical Introduction Using BlueJ*. Pearson, 2016.

“Stack Class in Java.” *GeeksforGeeks*, 17 Aug. 2020, www.geeksforgeeks.org/stack-class-in-java/.