IJP Assignment 2

Christopher Sipola (s1667278)

28th November 2016

Contents

1 Class design **1**
- 1.1 `MainProgram` . . . 1
- 1.2 `Controller` . . . 1
- 1.3 `World` . . . 2
- 1.4 `Location` . . . 2
- 1.5 `Item` . . . 3
- 1.6 `MoveCommand` (`enum`) . . . 3

2 Significant model design choices **3**
- 2.1 Modeling the user or navigator . . . 3
- 2.2 Orientation and location . . . 3

3 Additional comments **4**
- 3.1 External resources . . . 4
- 3.2 Publishing screenshots . . . 4

1 Class design

1.1 `MainProgram`

References: `Controller`

Referenced by: None

Loads the FXML file, sets the scene, and creates the `Controller` object.

1.2 `Controller`

References: `World`, `Location`, `Item`

Referenced by: `MainProgram`

Main class that interprets the commands from the GUI, passes the command to the world so it can update, and then updates the GUI based on the updated world.

With each update to the world, the menu items in the two menus—"Inventory" and "Pick up item"—are removed and then recreated using the updated item information. Instead of each item having a dedicated `ImageView` object, there are three `ImageView` "slots" which are populated with the images of the available pick-up items whenever (1) the location changes; or (2) the user takes an action to pick up or drop an item. The number of items in a location is therefore capped at three, although there is no limit on the number of items that can be held in inventory.

1.3 `World`

References: `Location`, `Item`

Referenced by: `Controller`

Creates the world and manages the all data related to the locations, connections between locations, the current location of the user, the current orientation of the user (i.e., the direction the user is facing), the pick-up items available at the current location, and the user's inventory items. When the user initiates the *Forward* command, the current location in the world is changed while the orientation remains the same. Conversely, if the user initiates the *Left* or *Right* commands, the orientation is updated while the current location remains the same.

Orientation is modeled as a cyclical variable from 0 to 3, where each increase in 1 modulo 4 (e.g., 0 to 1, or 3 to 0) represents a clockwise turn. Orientation is used in conjunction with `Location` methods to establish movement between locations and to identify the main image to display for the current location.

The logical map for the world used in this project, with its 11 locations and 5 items, can be seen in figure 1.

1.4 `Location`

References: `Item`

Referenced by: `Controller`, `World`

Holds information regarding pick-up items, images for the location, and connected locations. Each location image and each connected location is associated with a direction. Like orientation in `World`, direction is modeled as cyclical from 0 to 3. The images for the locations are sourced from the "images" subdirectory and set in the constructor.

1.5 `Item`

References: None

Referenced by: `Controller`, `World`, `Location`

Refers to an item that can be picked up or dropped. Has a description and an image.

1.6 `MoveCommand` (`enum`)

Consists of three fields: `LEFT`, `RIGHT`, and `FORWARD`. Used to pass the user's move command from the controller to the world and its methods. For example, the `updateOrientation` method in the `World` class takes a `MoveCommand` as an argument to determine how to change orientation.

2 Significant model design choices

2.1 Modeling the user or navigator

One alternative design choice was to create an instance of a new class `Navigator` which would manage information regarding the user's held items and orientation. This would allow the `World` class to refer to just the location data and the navigators (and not the user's place within the world), perhaps increasing the cohesion of the class.

However, with this design, I found that it was often a bit more involved to get basic information needed to update the GUI. For example, the current program needs to call a simple method from the `World` class to get the available set of pick-up items. However, with a `Navigator` class, one needs to call a `World` method that retrieves the current location of the navigator, and then use a `Location` method to extract the available pick-up items from that location.

2.2 Orientation and location

In the initial design, I tried to model direction with its own `Direction` class or enum. This abstraction would keep the internal representation of the direction hidden and would ensure directions updates result in legal values. However, I found that having the directions as enum (e.g., `Direction.NORTH`) seemed to require more `switch` or `if/else` statements to change direction according to move commands. I also found the field labels did not add much additional information to the integer representations.

3 Additional comments

3.1 External resources

The sources of the images are embedded in the comments of the `World` class, since this is where the items are instantiated.

3.2 Publishing screenshots

You may publish screenshots from this application.

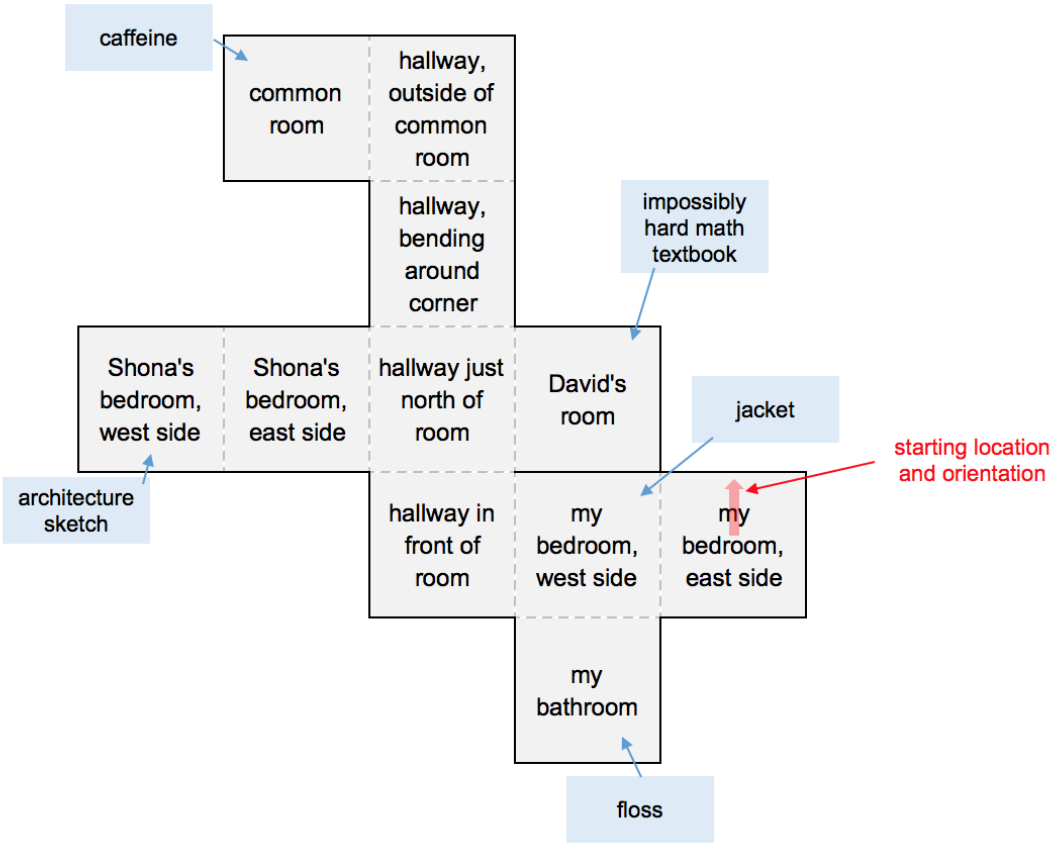

Figure 1: Logical map of world used in project. Each square represents a location. Connected locations are separated by a dotted line. Light blue squares indicate pick-up items.