IPPO Assignment 2

> *A Java project for the second assignment for the IPPO module at the University of Edinburgh.*

- Student exam number: b136325
- Course: Introduction to Practical Programming with Objects.
- Date: Wednesday 27th November 2019.

Contents

Overview

The Java project for the second assignment for the IPPO module at the University of Edinburgh is a visual navigation application. The application enables navigation between `walls` (represented by images) within a `room`, along with navigation between `rooms`. In addition, standalone items can be 'picked up' from a room or 'put down' into a room.

The application (using the `PlayersEdinburghService`) provides a means of visually navigating through several of the streets close to the university.

The application has been constructed using the IntelliJ editor (Community Edition 2019.2) on an Apple MacBook Pro with OS Catalina (version `10.15`). Java version `11.03` has been used, along with version `18.9` of the Java Runtime Environment. Lastly, Gradle version `5.6.2` has been used, as well.

Please note that due to a bug within IntelliJ I was unable to access the `Scene Builer` facility. Consequently, not only were the .fxml views generated by hand, but also - unfortunately - their layout may be rather crude. Further information about the IntelliJ bug can be found at the following link: `https://intellij-support.jetbrains.com/hc/en-us/community/posts/360004371099-Scene-Builder-does-not-work-anymore-after-update-to-2019-2-version`.

Navigation

Navigation within the application is composed of two types of change. On the one hand, the player's direction can be modified in compass directions: NORTH, EAST, SOUTH or WEST (assuming that there are walls for those directions within a specific room). On the other hand, when the wall associated with a given direction links to another room, then the player's 'forward' button will become enabled, and clicking on the forward button enables the player to walk between rooms. Thus, the navigation is composed of both directional (compass) and movement (forward) changes.

Changes from the initial design

For several weeks prior to beginning the initial design, I had been learning (at work) about client side, JavaScript based MVP (and related) frameworks, such as React with Redux state management. Client side perspectives of state are rather different from their server side equivalents, and, unfortunately, I rather stupidly blurred the two perspective within my initial design, producing a rather confused concept.

That being said, I reworked the initial design, and the application is relatively close to the revised concept. The one major difference concerns the structure of the .fxml based views and (within the application) how they are nested together.

Entry point

`main/java/ippo/assignment2/Main` is the entry point, and it calls `main/java/ippo/assignment2/App`. `App` derives a data service (using the `app.service` property from app.properties). App also interacts with the main view, `MainViewer.fxml`, and its associated controller, using the FXML loader.

Models

The application makes use of five models. They are as follows:

- Direction: an Enum class of directions.
- Item: a model representing both 'player' and 'room' items.
- Player: this model contains references to the current room, along with the 'player' items. The player model also supports navigation methods, such as 'turn' and 'moveForward', along with item related methods, such as 'pickUp' and 'putDown'. In addition the player model implements the Observable interface.
- Room: This model contains walls. It may also contain items.
- Wall: a wall model may contain an image. It may also contain a reference to another room.

The code for the models can be found within `main/java/ippo/assignment2/models` and the associated JUnit tests can be found at `test/java/ippo/assignment2/models`.

Collections

The application makes use of the following two collections:

- Items: a collection of Item models, which makes use of a `HashSet` data structure. Per collection, Items are unique.
- Walls: a collection of Wall models keyed by a Direction, and which makes use of `HashMap` data structure.

The items associated with both players and rooms are stored in Items collections. The Wall objects within a Room are stored using a Walls collections.

The code for the collections can be found within `main/java/ippo/assignment2/collections` and the associated JUnit tests can be found at `test/java/ippo/assignment2/collections`.

Views

The views for the application have been constructed using nested FXML markup. `MainViewer.fxml` is the parent view, and it defines the structure of how the child views are organised. The child views have been designed with regard to a function separation of concerns. That is, each view performs a separate front-end task.

The child views are as follows:

- `HeaderViewer.fxml`, which defines the components within the applications's header.
- `ImageViewer.fxml`, which will be used to display the current view or image.
- `NavigationViewer.fxml`, which handles user navigation.
- `PlayerItemsView.fxml`, which displays the players items. It also provides a 'putDown' button.
- `RoomItemsView.fxml`, which displays the room items. It also provides a 'pickDown' button.

There is a one to one relationship between views and controllers, and the controller associated with each view has been specified using the `fx:controller` tag.

The code for the views can be found within `main/reaources/fxml`.

Styling

The styling of the views uses the Twitter Bootstrap CSS theme. This is a widely used theme, familiar to both developers and users.

The code for the theme can be found at `main/reaources/css/bootrap2.css`.

Controllers

With the exception of `MainController`, all of the child controllers inherit from `AbstractObserverController`, which means that they implement the Observer interface. The child controllers will thus be updated when the Observable model, player, changes. Each of the child controllers has an `updateView` method and it is that method that perform the update following a change to the player model.

The child controllers are as follows:

- HeaderController
- ImageController
- NavigationController
- PlayerItemsController
- RoomItemsController

The code for the controllers can be found within `main/java/ippo/assignment2/controllers`.

In addition, integration tests have been written for the controllers and the views using JUnit and TestFX. The integration tests can be found within `test/java/ippo/assignment2/controllers`.

Controllers and data

As mentioned above, main/java/ippo.assignment2.App makes uses of a data service (identified using the app.service property from app.properties file). The property name is passed to the ServicesFactory in order to create the appropriate service, which by default is PlayersEdinburghService. Once App has accessed the service it passes a player model to MainController via the `setPlayer` controller method. In turn, MainController passes the player to the child controllers. It also registers the child controllers to be updated when the player changes.

The code for ServicesFactory can be found at `main/java/ippo/assignment2/factories/ServiceFactory`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/factories/ServiceFactory`.

In addition, the code for the PlayersEdinburghService can be found at `main/java/ippo/assignment2/services/PlayerEdinburghService`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/services/PlayerEdinburghService`.

In addition, the PlayersService is available. However, it was primarily constructed for testing purposes, and it does not contain main images.

Properties

Configuration properties (such as those defined within main/reaources/properties) are made available within the application via the PropertiesSingleton class.

The code for the singleton can be found at `main/java/ippo/assignment2/properties/PropertiesSingleton`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/properties/PropertiesSingleton`.

Resources

With the exception of the views, which are loaded using the FXMLLoader, all other resources are loaded using methods found within the ResourceFileLoader.

The code for the loader can be found at `main/java/ippo/assignment2/loaders/ResourceFileLoader`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/loaders/ResourceFileLoader`.

JSON

When the application makes use of data build from JSON, the data can be found within `resources/JSON`. The JSON is parsed using the `JsonFileParser`, and the output of the parsing process is passed to the `JsonModelBuilder`, which constructs appropriate models.

The code for JsonFileParser can be found at `main/java/ippo/assignment2/parsers/JsonFileParser`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/parsers/JsonFileParser`.

The code for JsonModelBuilder can be found at `main/java/ippo/assignment2/builders/JsonModelBuilder`, and the associated JUnit tests can be found at `test/java/ippo/assignment2/builders/JsonModelBuilder`.

At present, the models constructed by JsonModelBuilde are made available via the `PlayersJsonService`. To use this service, set the `app.service` config property to `PlayersJsonService`.

JsonModelBuilder constructs items and rooms, along with room related content and views. Unfortunately, however, it does not yet handle 'placeholder' values; such as those within the sample JSON. That being said, the usage of JsonModelBuilder via the PlayersJsonService works well with the existing architecture of the application.