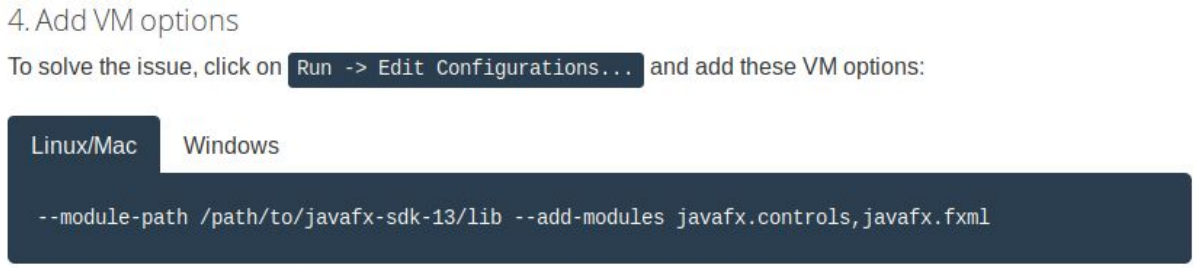
Date: 27/11/2019
Name: David Goerig (djg53)
School: School of Computing
Programme of Study: Advanced Computer Science (Computational Intelligence)
Module name: CO871 Advanced Java
E-mail: djg53@kent.ac.uk
Academic Adviser: DR P Kenny (P.G.Kenny@kent.ac.uk)

Assessment 2 - Behaviour parameterisation, Streams, and GUI

1. What setup and/or IDE should be used to run the project

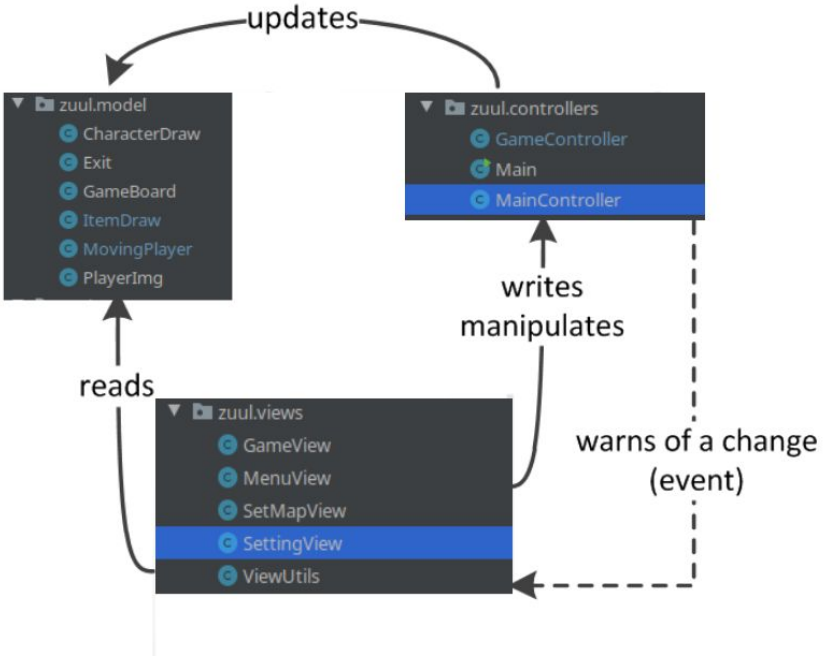
I used for developping my project Intellij. The only important thing is to add the library. I followed the official tutorial: https://openjfx.io/.



So no particular settings are needed.

2. The design choices I have made

I choose to use a classic MVP (model / vue / controller) pattern for the GUI. Each “page” (or vue) have is own vue (See figure 1 - 4), which is updated by controllers and who is displaying one or more models:



Using this design permit future developper to clearly distinguish each view, used models, and where they are updated.

Also I added as much behavior parameterization as possible. For example in room modifier (to permit to delete room with certain conditions, or to add object to room with certain conditions, etc.). These conditions for choosing rooms use Predicate so we can easily update the conditions, or add some filter:

```
private static HashMap<String, Room> filter(HashMap<String, Room> roomList, Predicate p) {
    HashMap<String, Room> result = new HashMap<>();
    for (Map.Entry<String, Room> stringRoomEntry : roomList.entrySet()) {
        Map.Entry entry = stringRoomEntry;
        if (p.test(entry.getValue())) {
            result.put(entry.getKey().toString(), (Room) entry.getValue());
        }

    }
    return result;
}

private HashMap<String, Room> filterRoomWithoutExit(HashMap<String, Room> allRooms) {
    return filter(allRooms, new Predicate<Room>(){
        public boolean test(Room a) {
            String[] directions = {"north", "east", "south", "west"};
            for (String dir: directions) {
                if (a.getExit(dir) != null) {
                    return false;
                }
            }
            return true;
        }
    });
}
```

3. Interesting features

In the map setter (where we can download new ones: Figure 4), if the map don't follow the format, the Error describe exactly where is the error and what caused it. Example:

The game can be play in two ways (see Figure 2). On the left there is the panel with all the buttons / tabs / information of the room (room name, room description, available exits button to change map, player items, button for dropping these items, items in the room, button for taking these items, a tab with the characters in the room, and finally a button to give an item to one of this character). And on the right there is a playable game (with the arrow keys).

The player (Deadpool) can move up, down, left, right, changing room using arrows and taking object by going on them (red bubbles). Other characters in the room are displayed too (as a Pikachu).

Another interesting feature is that we can modify the rooms as we want (Figure 3):

- all the room and their informations are displayed in a tab.
- room can be delete
- items can be add to room
- items can be add to all room without exits
- we can remove all room without exits / items
- We can set player name and first position
- We can add as many character that we want to any room

And all the error management is done, for example if we delete a room, the starting position will be updated too, etc.

All the code is documented.

4. Ressources:

Figure 1: Menu view

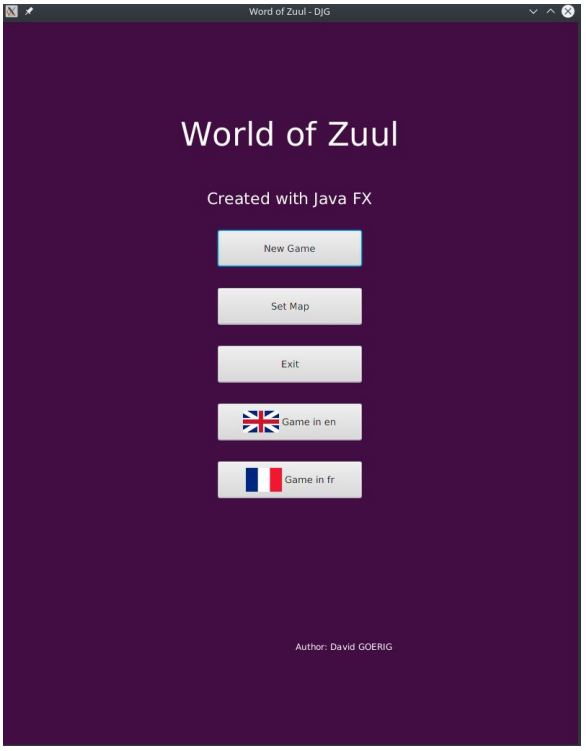


Figure 2: Game view

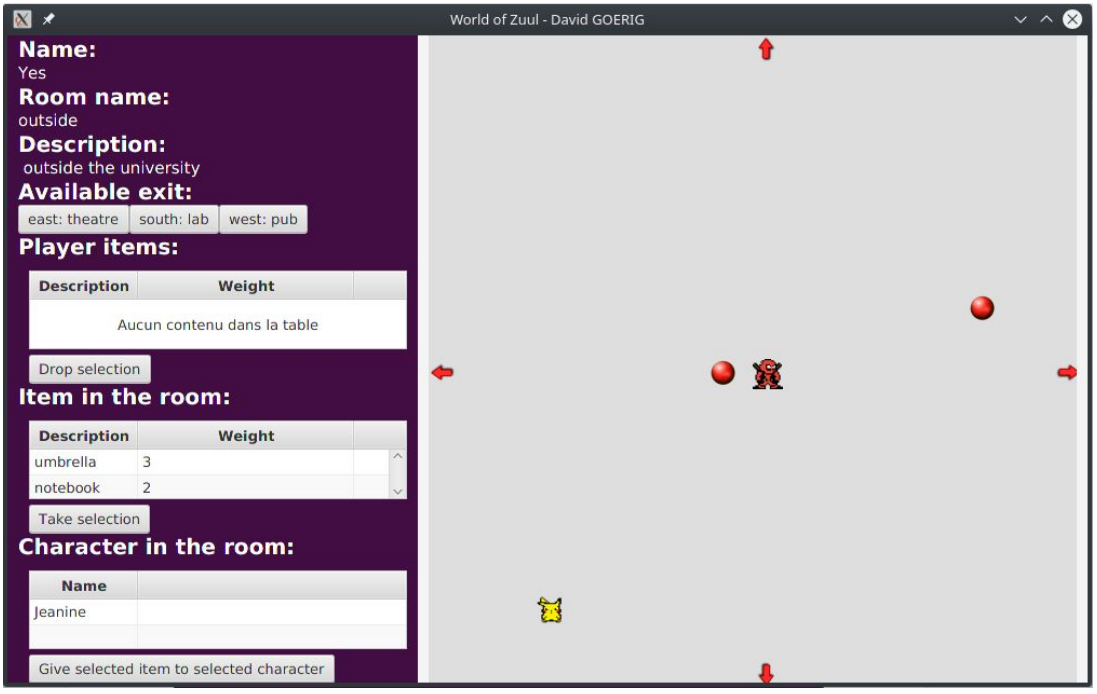


Figure 3: Setting view

Figure 4: Map setter view