**Concordia University**

Department of Computer Science and Software Engineering

Advanced Programming Practices
SOEN 6441 – Winter 2020

# Software Architecture & Methodologies

Team **Java Bean**

| | |
|---|---|
| Jatan Gohel | 4007 8112 |
| Siddhant Arora | 4008 5538 |
| Krunal Jagani | 4005 6939 |
| Harsh Vaghani | 4008 4099 |

**Guided by**

Dr. Rodrigo Morales

# Architectural Design Document

## Introduction

Our purpose is to build the Monopoly game from the game framework we have developed in the first build. Which follows a Model View Controller (MVC) architectural design model. We have studied and made sure we follow the extreme programming approach for the smooth development of software by implementing features like:

- **Planning:** Since from the 1st build, our goal was to create our framework as flexible and rich as possible so that it would become easier for us to do game implementation less hard. First, we decided to create a list of multiple reusable modules and build our architecture of 2nd build around them. We added Strategy Pattern, Load and Save Map functionality in our 3rd build.

- **Testing:** Testing is done by writing unit test cases for each module and determined the veracity of the code we have developed. We have used Junit framework for this purpose.

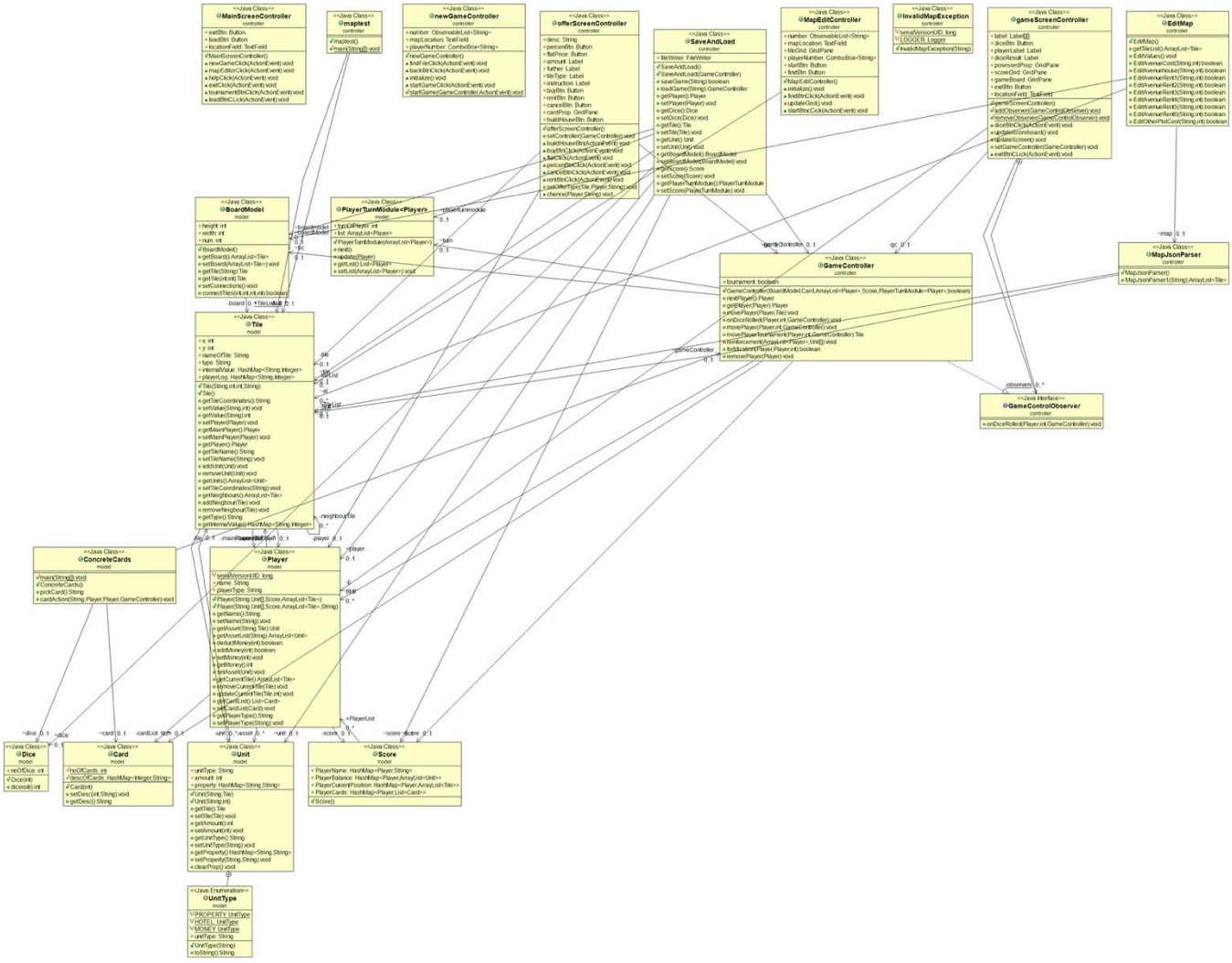
- **Collective Ownership:** Distributed work among group member for different modules, saved changes into different branches and then later on merged all of them into local branch.

- **Simple Design:** Complex designs were avoided in order to make the software more user-friendly, and to also avoid faults. Smooth features were implemented with the help of Simple designs which resulted in quick, structured and clear game flow.

- **Continuous Integration:** Git was used as Revision Version Control which made all the group members up to date. Maintenance of concurrent changes, rollback sequences were maintained through local branches and master branch. Errors and conflicts detected at a time of integration were rectified quickly.

## Architecture:

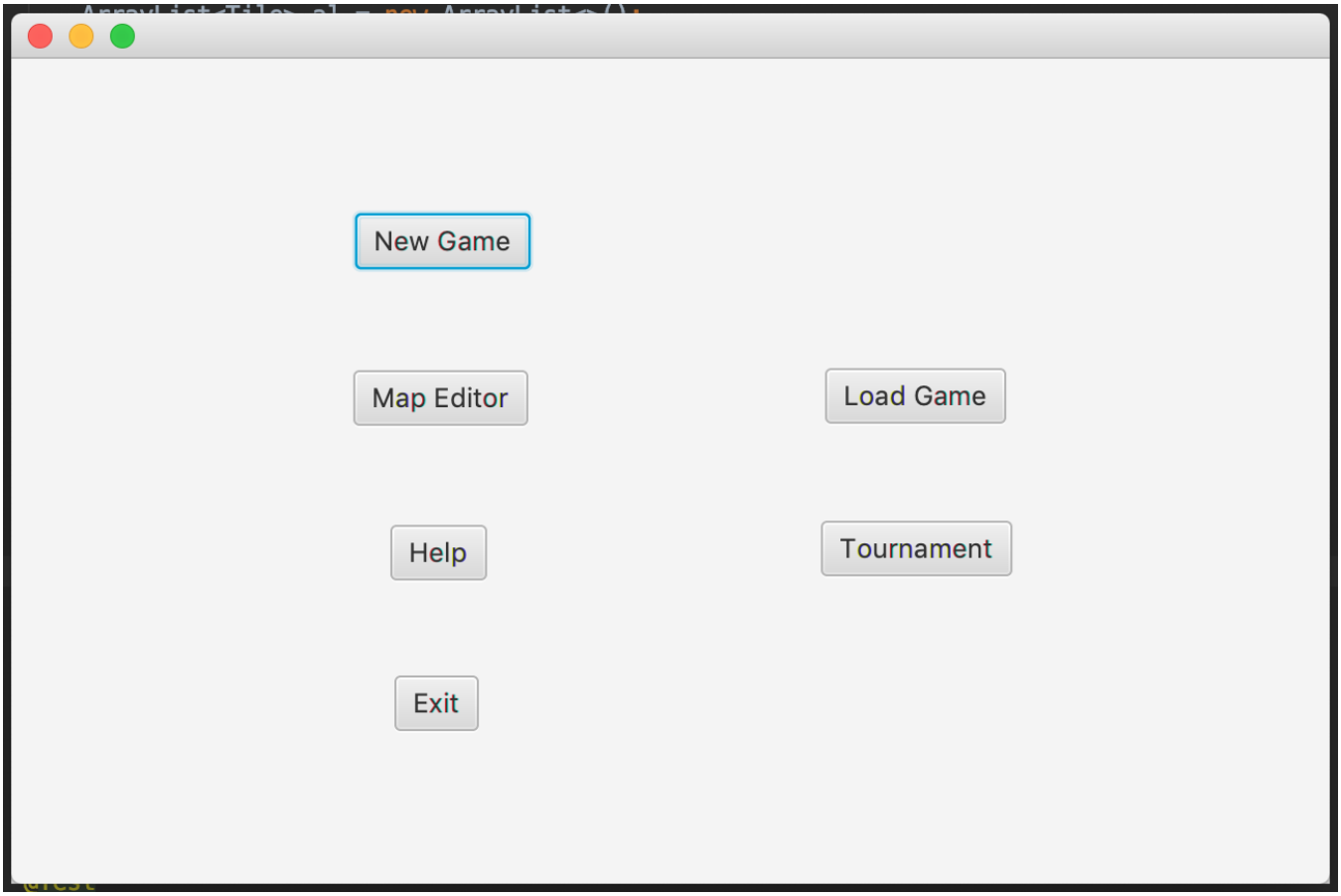
Since our first build was based on MVC architecture, we decided to continue working in the same architecture after doing some amount of refactoring to facilitate future needs of the second build.

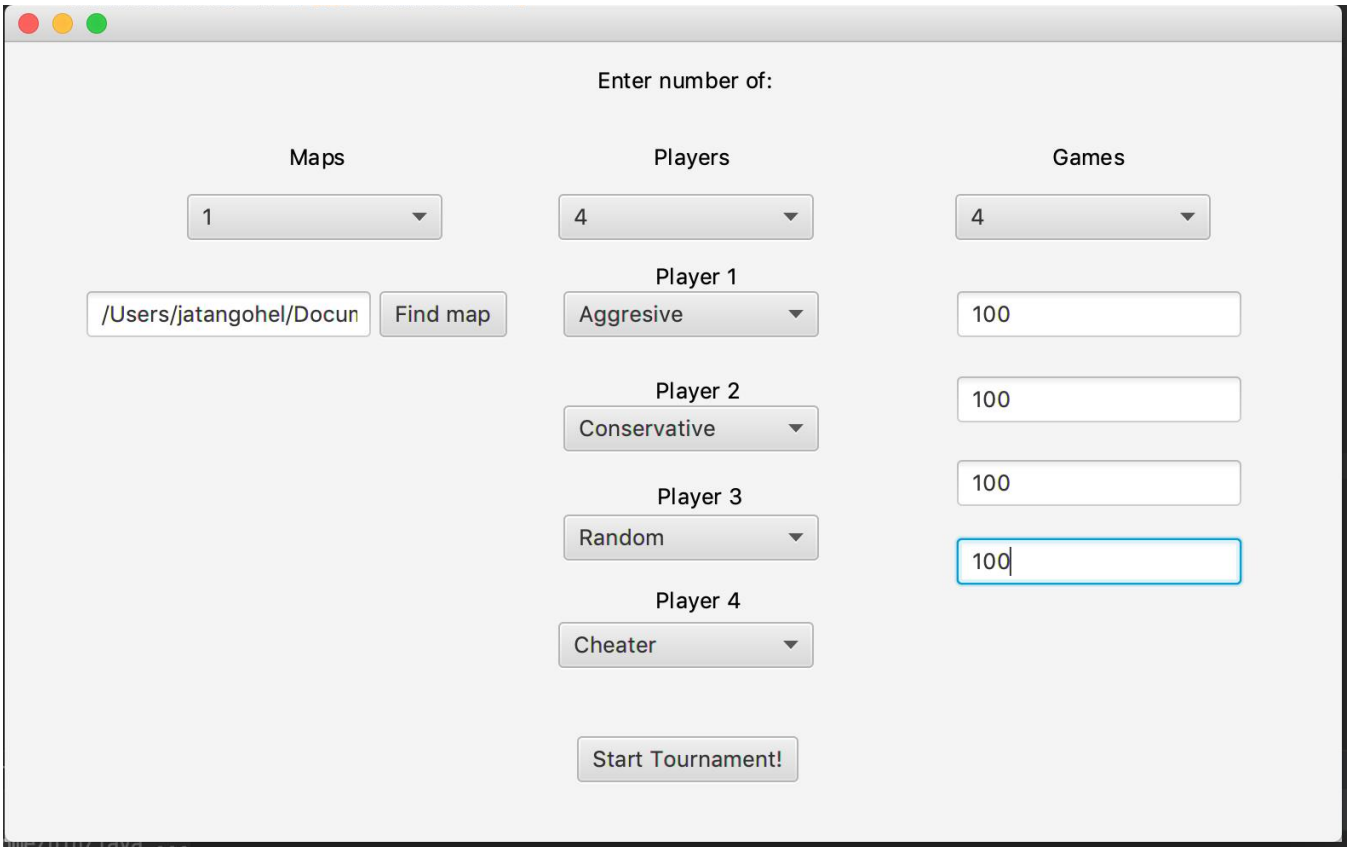



Apart from that observer design pattern was implemented in the GameController where game controller will notify on a successful dice roll.

```
private static ArrayList<GameControlObserver> observers = new ArrayList<>();

private GameController gc;
private Player p;

public static void addObserver(GameControlObserver gameControlObserver) {
        observers.add(gameControlObserver);
}

public static void removeObserver(GameControlObserver gameControlObserver) {
        observers.remove(gameControlObserver);
}
```

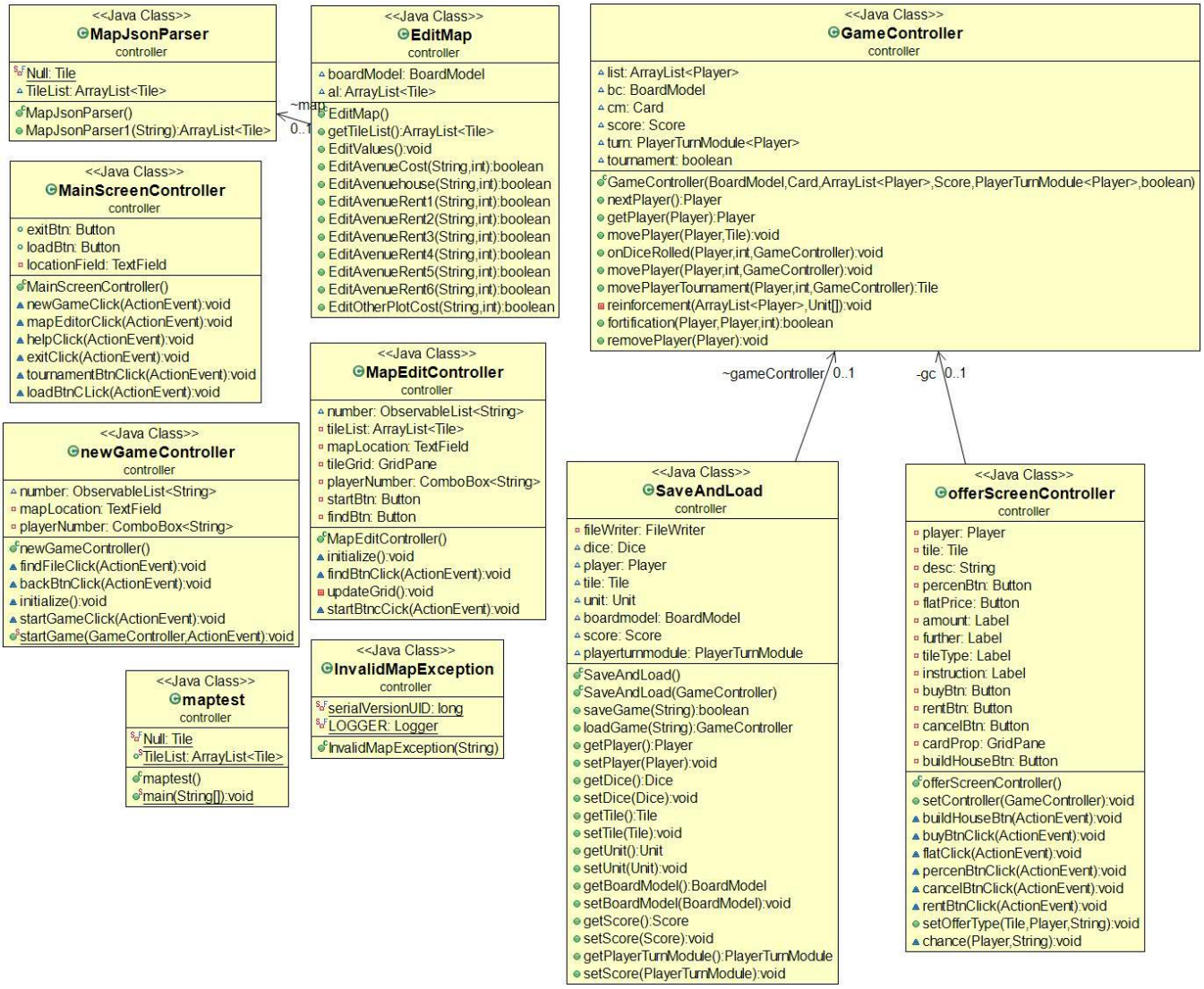
**Main screen View (new files added)**:




Enter number of:
Maps
Players
Games
1
4
4
/Users/jatangohel/Docun
Find map
Player 1
Aggresive
100
Player 2
Conservative
100
100
Player 3
Random
100
Player 4
Cheater
Start Tournament!

```
After :player1******Conservative******3324
--------------------------------------------------------------------
Player before :player2******Conservative******2782
User rolled a 6
you landed on Ventura Avenue
After :player2******Conservative******2760
--------------------------------------------------------------------
Player before :player3******Random******2389
User rolled a 6
you landed on Boardwalk
After :player3******Random******2339
--------------------------------------------------------------------
Player before :player0******Aggresive******1312
User rolled a 5
you landed on Community Chest
You received the following card : Empty
After :player0******Aggresive******1312
--------------------------------------------------------------------
Player before :player1******Conservative******3324
User rolled a 2
you landed on Newyork Avenue
After :player1******Conservative******3308
--------------------------------------------------------------------
Player before :player2******Conservative******2760
User rolled a 6
you landed on Community Chest
You received the following card : Building loan matures. Collect $150 from Bank
After :player2******Conservative******2910
--------------------------------------------------------------------
Player before :player3******Random******2339
User rolled a 3
you landed on Community Chest
You received the following card : Move To Go
After :player3******Random******2739
--------------------------------------------------------------------
Player before :player0******Aggresive******1328
User rolled a 3
you landed on Free Parking
After :player0******Aggresive******1328
--------------------------------------------------------------------
Player before :player1******Conservative******3308
User rolled a 4
you landed on Indiana Avenue
After :player1******Conservative******3290
--------------------------------------------------------------------
Player before :player2******Conservative******2910
User rolled a 2
you landed on Shortline
Conservative player : Won't buy any property
After :player2******Conservative******2910
--------------------------------------------------------------------
Player before :player3******Random******2739
User rolled a 3
you landed on Baltic Avenue
After :player3******Random******2735
--------------------------------------------------------------------
Draw
```

**Controller (new files added)**:




1. Edit map: Edit map has functionality to edit values of different tiles based on their properties. It has some security constraints. Such as end user can not change the names and type of properties because in monopoly each type of tile serves its purpose. If the property is Avenue then, an end user has the permission to edit its rents, house and cost prices.

2. Map file parser: Map file parser parses data in the form of Json format. Every object is handled individually as a matter of the fact that each plot has a different property. For instance, Avenue plots has different property than go and go has different property than railroads. So, each object had to be treated differently.

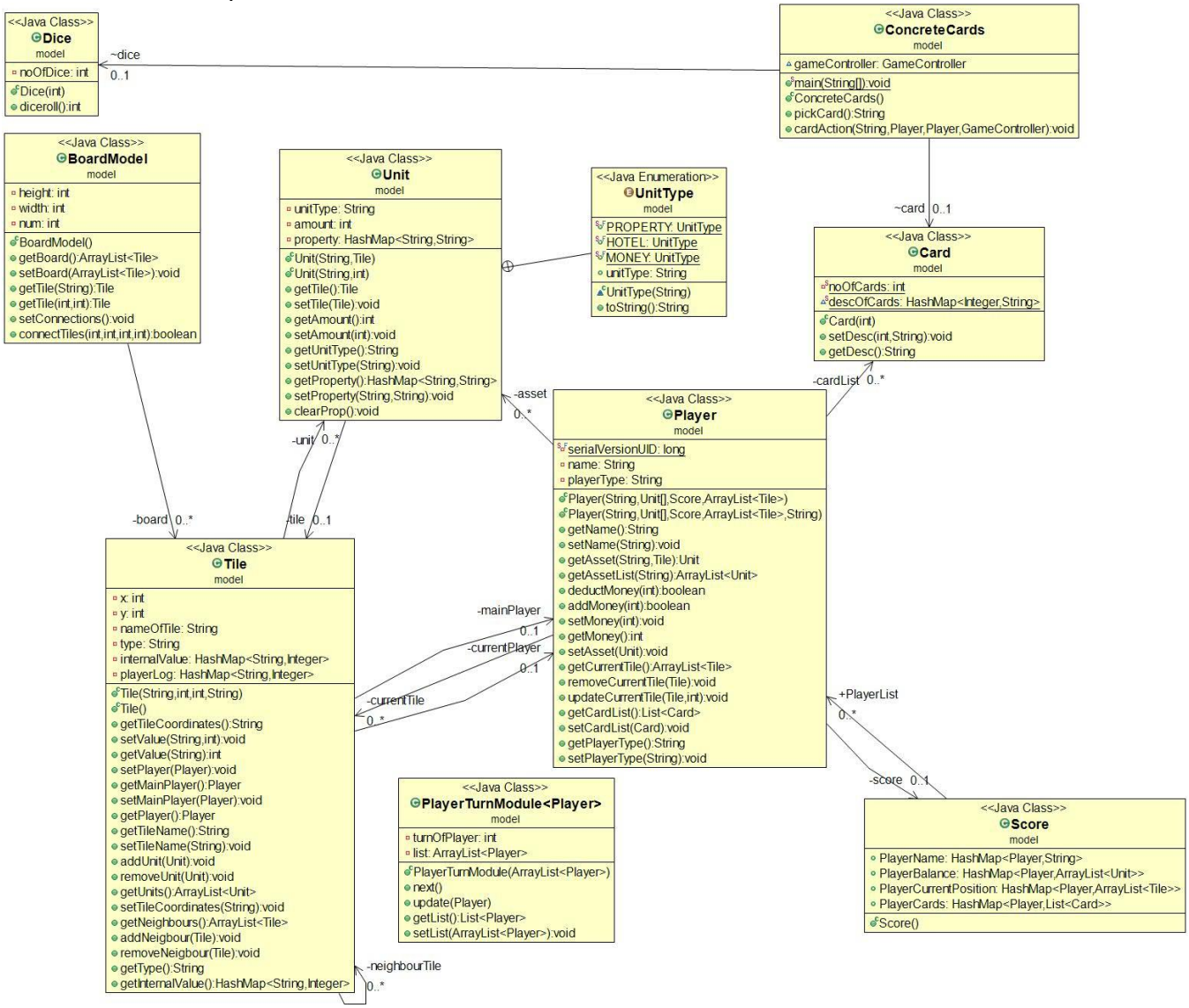
3. Invalid map file exception: It is used to generate exceptions when invalid map is passed as the parameter to the game.

4. Game Screen Controller has methods to control specific actions while interacting with the main screen. It implements observer design pattern to update screen.

5. OfferScreenController: It handles different interaction between objects such as cards, players, turn money transfer. For instance, If a player lands on a card, a card will be picked randomly and method inside it will get executed.

## Models (new files added):

- Concrete cards: Concrete implementation of the cards modules where we defined different community chest cards.

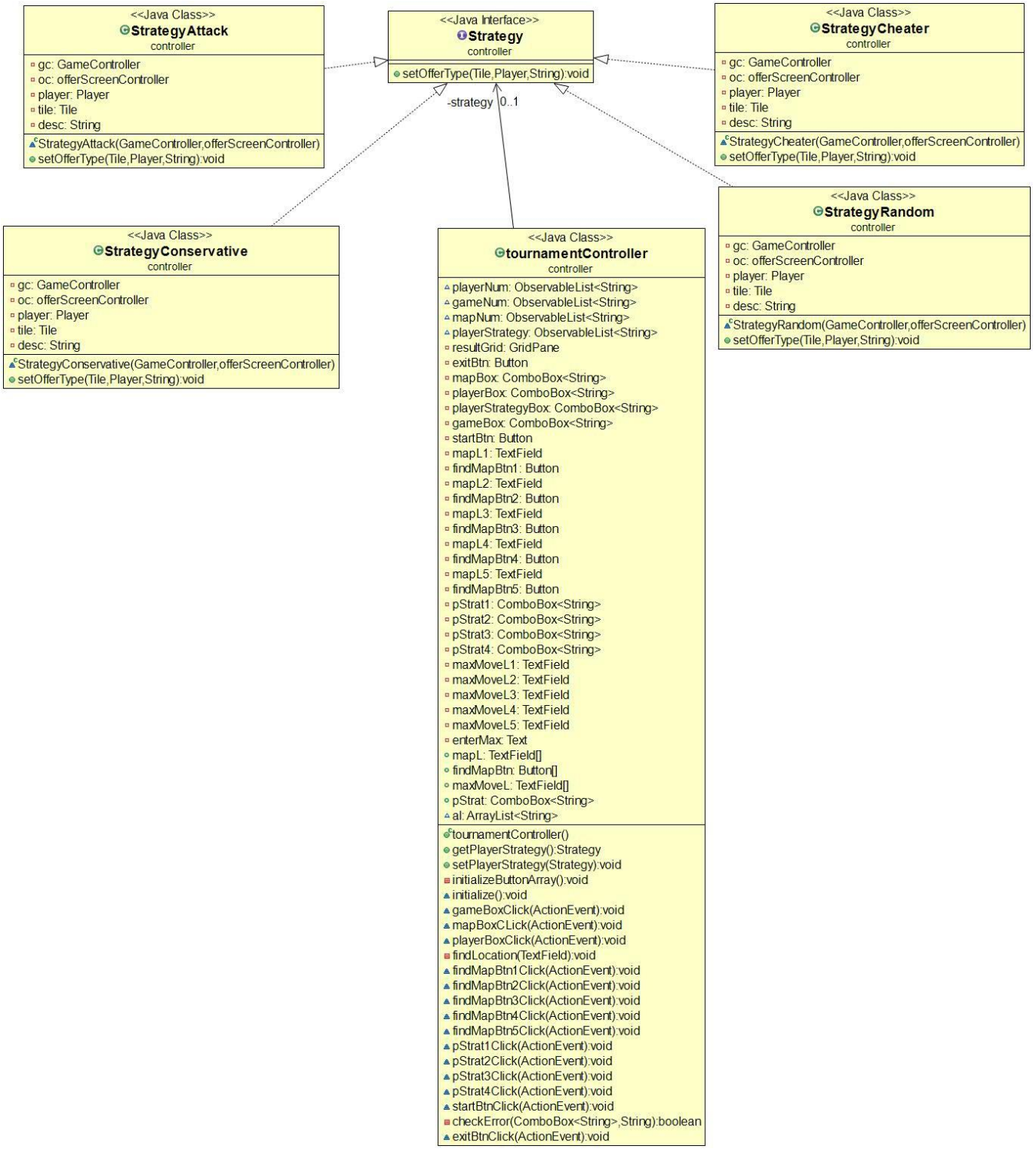



## Strategy Pattern:

Strategy pattern involves removing an algorithm from its host class and putting it in separate class so that in the same programming context there might be different algorithms (i.e. in our case, Aggressive,

Conservative, Random and Cheater), which can be selected in runtime. Strategy pattern enables a client code to choose from a family of related but different algorithms and gives it a simple way to choose any of the algorithm in runtime depending on the client context.

In Build 3, we implemented strategy pattern which can be explained with the following updated class diagram:



## References


- http://www.oracle.com/technetwork/articles/javase/index-142890.html
- https://mdn.mozillademos.org/files/16042/model-view-controller-light-blue