Monopoly Game Simulator

Software Requirements Specification

v1.0

Lead Software Engineers

Muhammed Bera KOÇ

Salih ÖZYURT

Mücahit TANACIOĞLU

Ahmet LEKESİZ

CUSTOMERS

Murat Can GANİZ

Serap KORKMAZ

Content

1. Introduction

1.1 Purpose

The purpose of project provides customers to observer famous Monopoly Game in simulation.

1.2 Scope

Monopoly Game is played in simulation by simulated players(SP) that the customers define count of. Monopoly Game will teach customers how to play the game if he/she doesn't know play game. And it gives a manifest chance to create new strategies about the game. The game heavily depends on random events. Actually all choice mechanisms in the game created based on random values generated by the computer itself. Game has some limits such as number of players must be between 2 and 8. There are 3 different types of squares. Squares are blocks which creates the game board. Tax square takes money from the SP whenever SP lands on it.
Go square is a square which has no interaction yet whenever SP start a new turn on the board -by that it is meant that SP must traverse all the board.- go square(starting square) gives the user a specified amount of money by the customer to the SP. The regular square on the other hand has nothing to do. There is a bankruptcy case in the game which occurs when SP balance is less than zero. Game ends when other than the one player, all players are bankrupt.

1.3 Definitions, Acronyms, and Abbreviations

SP(Simulated Player): A simulated player is a player which is controlled by the system - or program-. A customer has no influence and control above it. It is only directed by the system by random c

1.4 References

- Head First Design Patterns: A Brain-Friendly Guide (1st Edition) by Eric Freeman, Bert Bates, Kathy Sierra, Elisabeth Robson
- Head First Object-Oriented Analysis and Design (1st Edition) by Brett D. McLaughlin, Gary Pollice, Dave West
- Object-Oriented Analysis and Design with Applications (3rd Edition) by Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Boobi J. Young, Jim Conallen, Kelli A. Houston
- Effective Java 3rd Edition by Joshua Bloch

2. General Description

2.1 Product Perspective

The product should be similar to Monopoly Game. It requires to have some players and squares to simulate the game.

2.2 Product Functions

The product will simulate Monopoly Game by using the given information. After each step of the players, the program should display the current information of the players in the console. At the end of the game, the program must display the information of the winning player and the loser players on the console.

2.3 User Characteristics

The user should be able to change JSON file to give initial information to the program. Java should be installed in the user's computer. Also, the user should be able to run the game via terminal.

2.4 General Constraints

First constraint is simulation. Product can't be interrupted any process because it is meant to be run by the system itself. A second constraint is that the product cannot contain GUI.

2.5 Assumptions and Dependencies

The product can be run any operating system since it programmed in Java. During the development of JUnit and JSON Simple are used as dependencies.

3.2 Functional Requirements

3.2.1 Monopoly Game

3.2.1.1 Introduction

This is a classic monopoly game simulation.

3.2.1.2 Inputs

Required a JSon file to build game variables.

3.2.1.3 Processing

The data obtained from JSon file will process to build game.

3.2.1.4 Outputs

When a player's turn comes, we print all the details.

3.2.1.5 Error Handling

There is no error handling.

3.3 Use Cases

3.3.1 Use Case #1

First, program reads instructions to start the game from a specific JSON File. After reading those infos, it prints them and starts the game. Then it initialises the players, giving them a unique piece and given balance.Then it starts the first round. In each turn it demands every Simulated Player(SP) to roll a dice. Before that it prints SP type, turn counter, cycle counter, SP location and current SP balance. After rolling two dice program controls the square that matches the total dice value. Program moves the piece of current SP to the given square and perform whatever the specific square requires and prints SP location and SP balance. Then it gives the turn to the other player this goes like this until the round is over. When round is over program starts the round again. The only thing ceases this process is to detect that there is only one player remained. If an SP has no balance, it will be eliminated. When this occurs it prints the winner then the eliminated players.

3.4 Classes / Objects

3.4.1 Player

3.4.1.1 Attributes

```
int balance,int cycleCounter, int totalDiceValue, int
roundValue, int currentDiceVal
boolean bankruptFlag
Dice playerDice
Piece.PieceType pieceType
```

3.4.1.2 Functions

```
void checkAndUpdatePlayer(int currentDiceValue, Square
currentSquare)

int rollDice()

boolean isPlayerCrossTheGoSquare()
```

```
boolean isPlayerBankrupted()

DPlayer getDPlayer()
```

3.4.2 Board

3.4.2.1 Attributes

```
SQUARE_NUMBER = 40

Square[] squares

static Board boardInstance

Instruction instructionInstance

int countOfTaxSquares
```

3.4.2.2 Functions

```
void initSquares()

Square[] getSquares()

static Board getInstance()
```

3.4.3 Square

3.4.3.1 Attributes

```
String SQUARE_TYPE = "TAX_SQUARE";

Instruction DInstructionInstance =
Instruction.getInstance();
```

3.4.3.2 Functions

```
void performOnLand(Player player)

String getSQUARE_TYPE()
```

3.4.4 Dice

3.4.4.1 Attributes

Dice class includes no attributes.

3.4.4.2 Functions

```
int[] rollDice()
```

3.4.5 Observer

3.4.5.1 Attributes

Observer interface class includes no attributes.

3.4.5.2 Functions

```
void checkAndUpdatePlayer(int currentDiceValue, Square
currentSquare)
```

```
void listen()
```

3.4.6 Terminal

3.4.6.1 Attributes

Terminal class includes no attributes.

3.4.6.2 Functions

```
void printCurrentJSONFile()
```

```
void printBeforeRollDice(Player bPlayer)
```

```
void printDicesFaces(int[] diceValues)
```

```
void printLocationType(String squareLocation)
```

```
void printAfterRollDice(Player bPlayer)
```

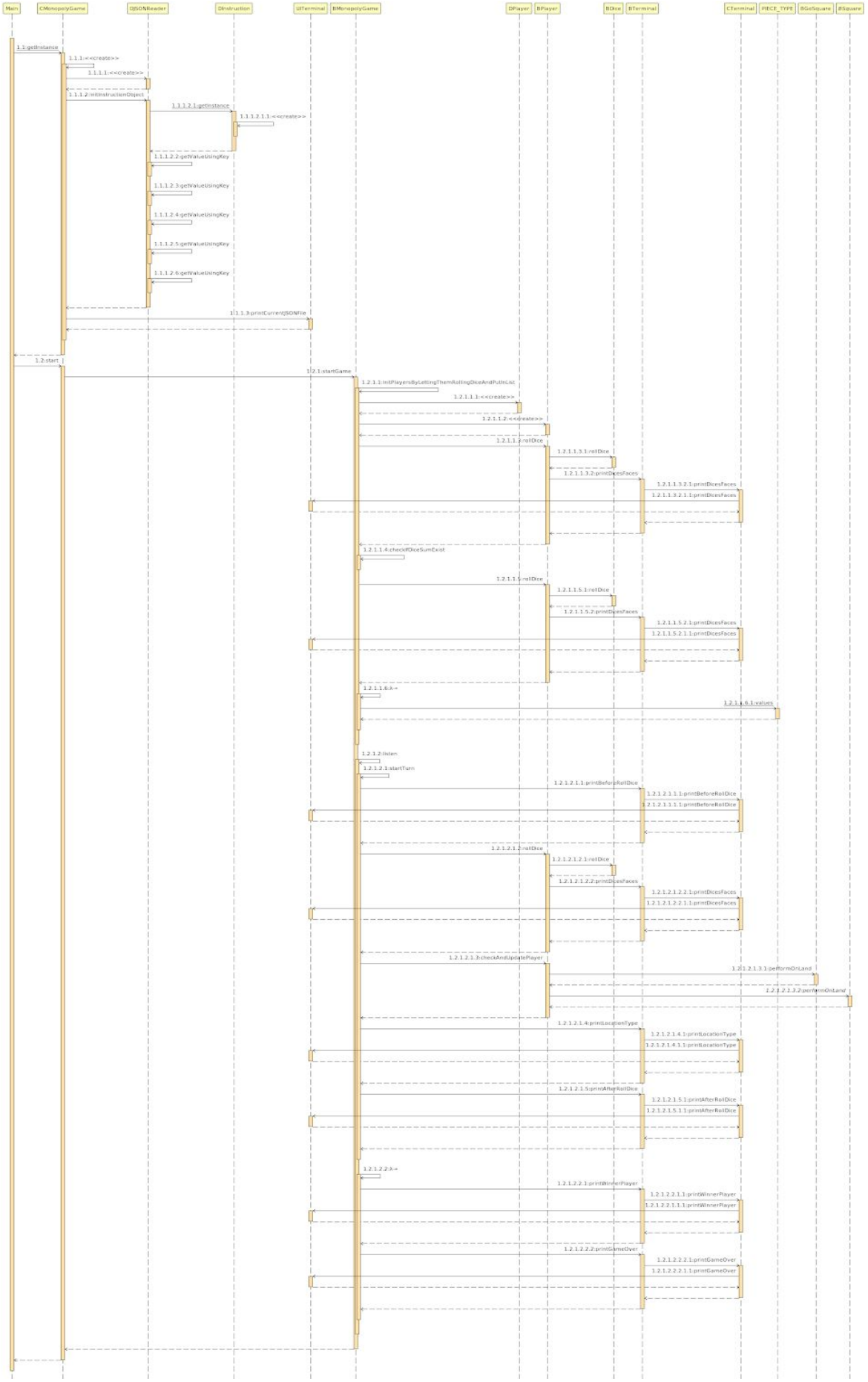
```
void printWinnerPlayer(Player bPlayer)
```

```
void printGameOver(ArrayList<Player> eliminatedList)
```

4. Analysis Models

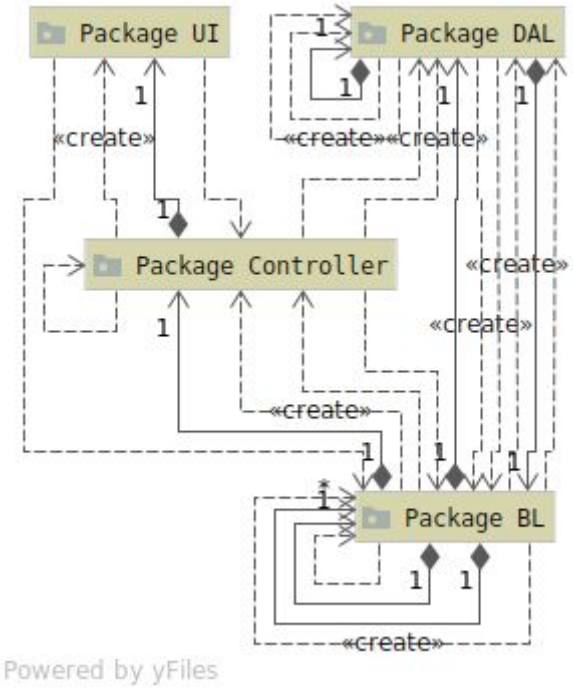
You can access all models using the link: **Models in**

4.1 Sequence Diagram

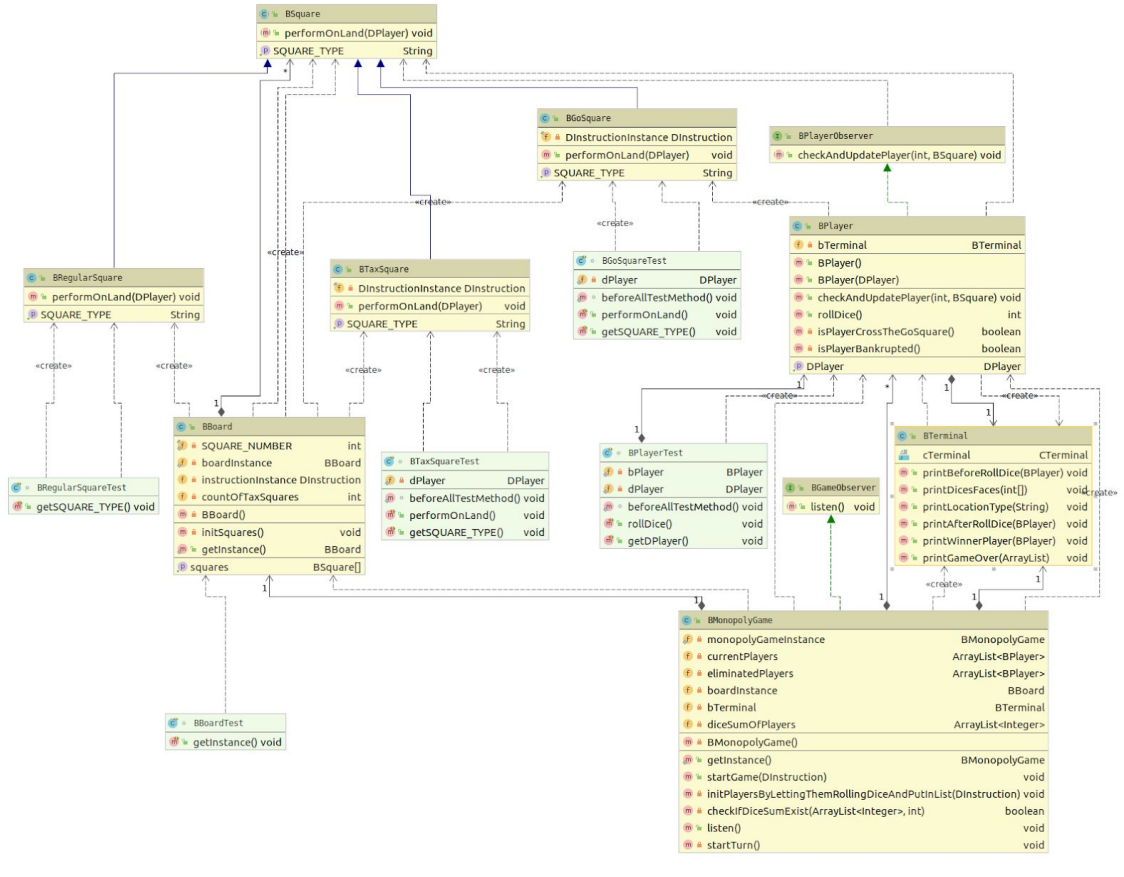


4.3 UML Diagrams

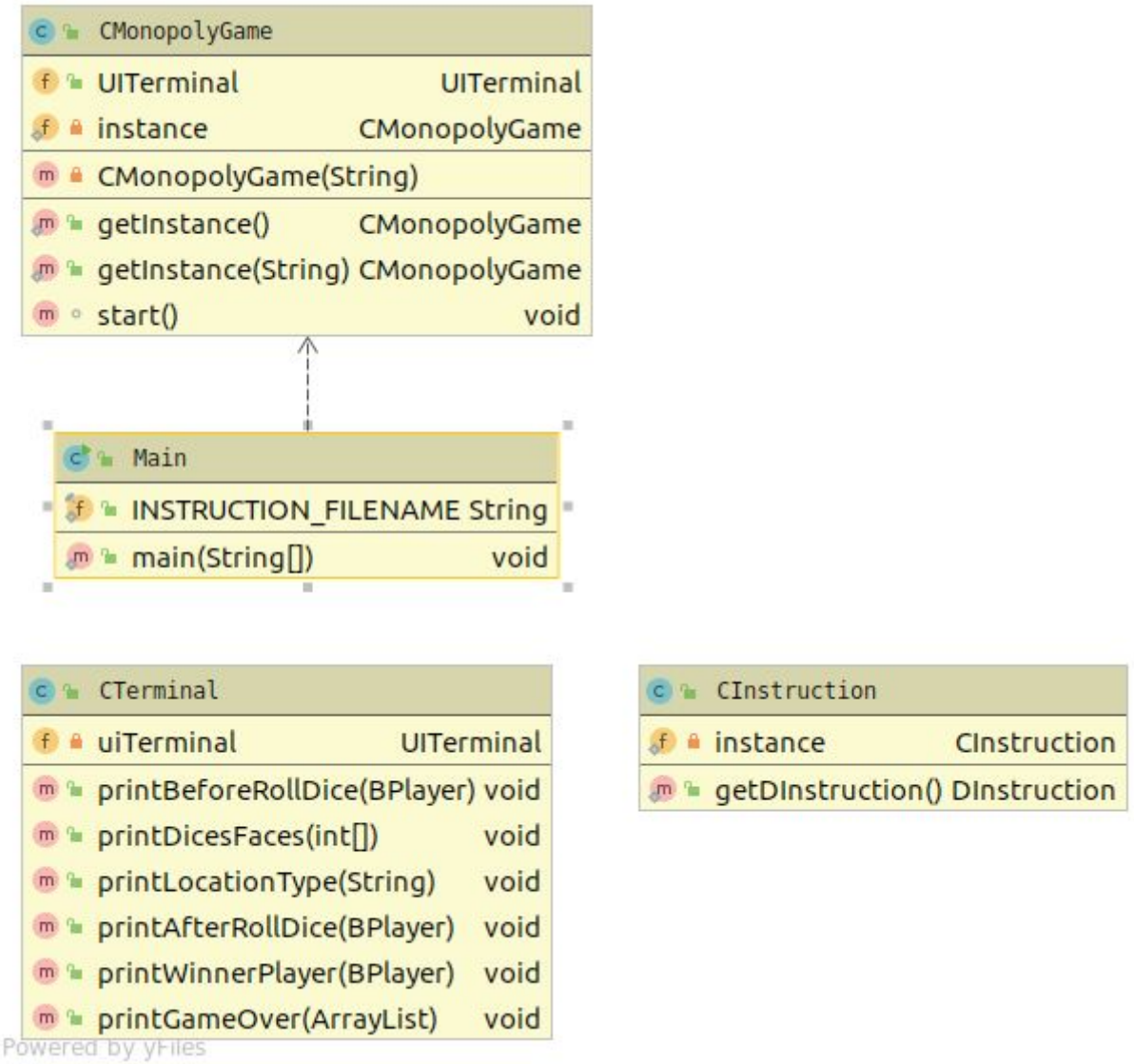
Package Dependencies



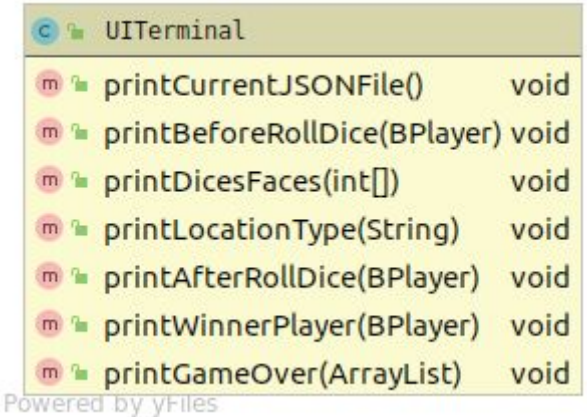
BL package UML Diagram



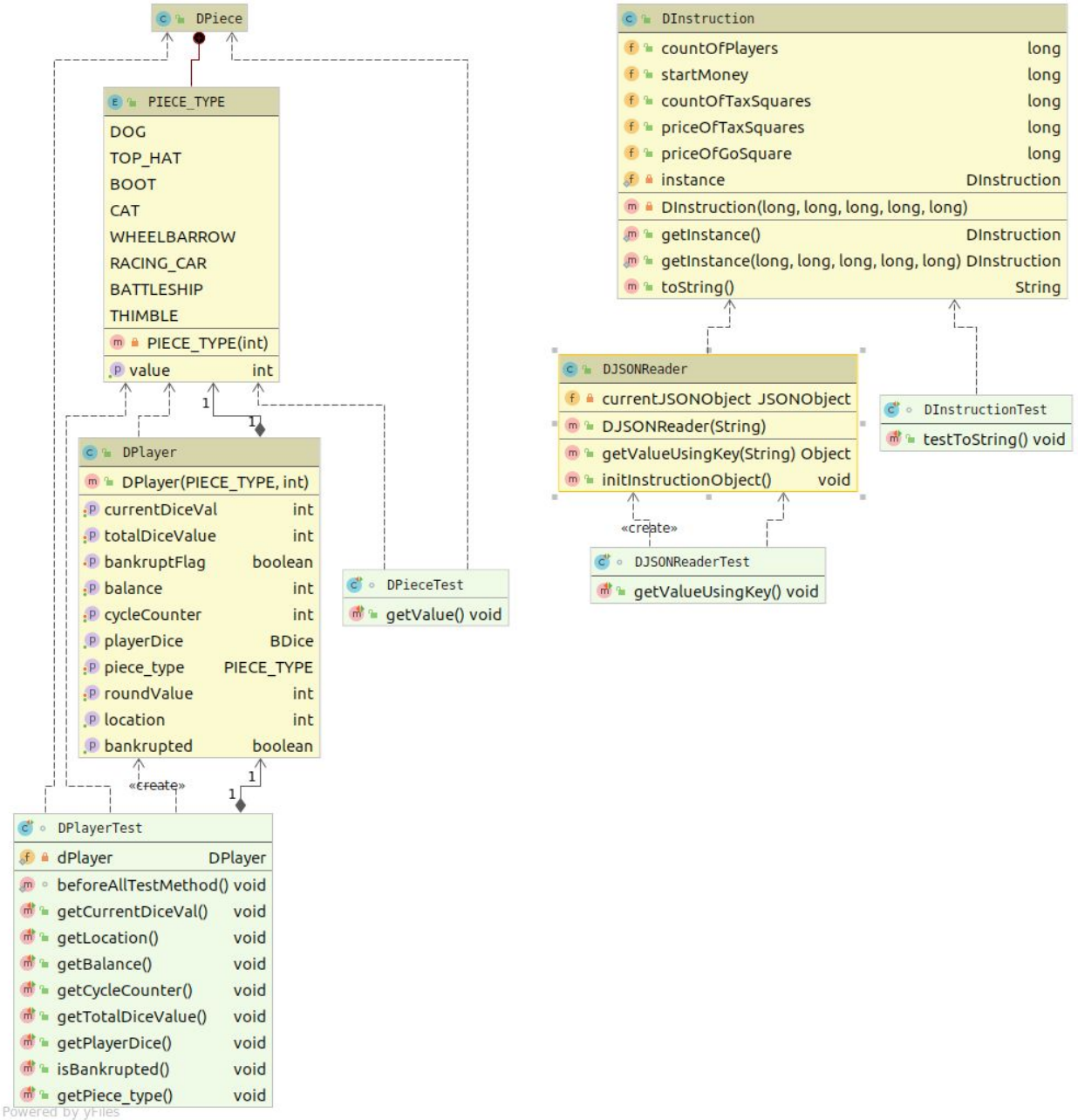
Controller package UML Diagram



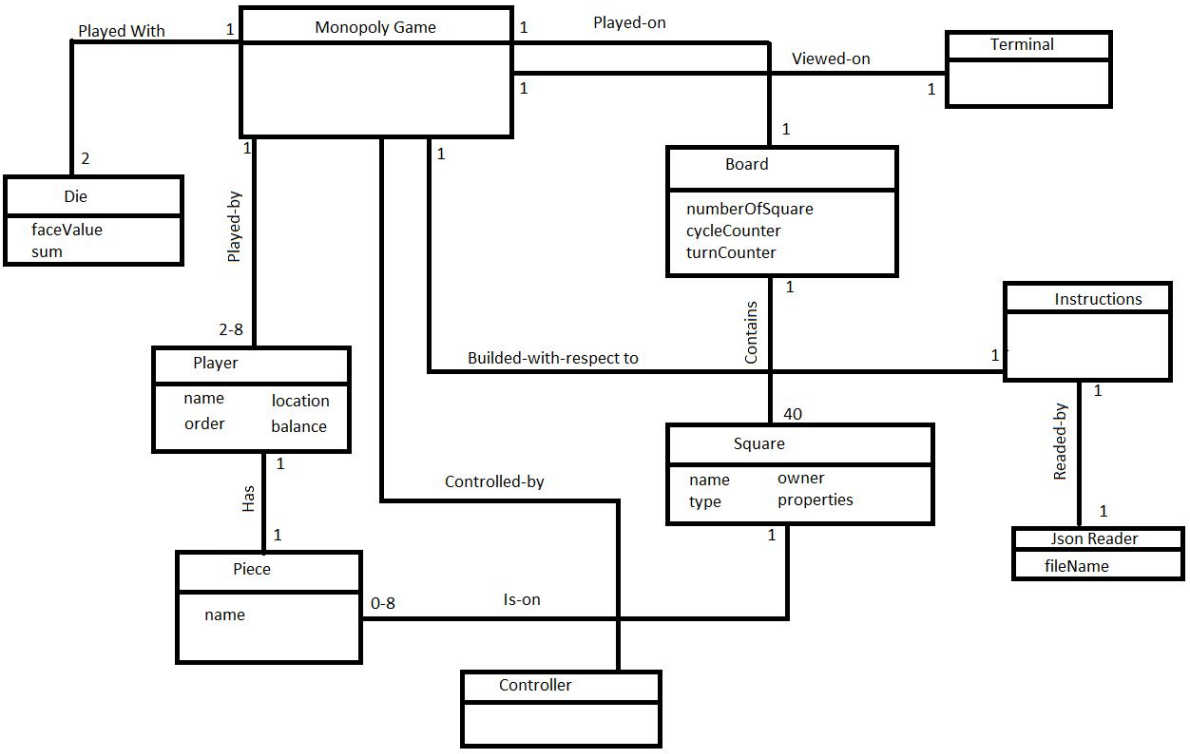
UI package UML Diagram



DAL package UML package


DPiece
PIECE_TYPE
DOG
TOP_HAT
BOOT
CAT
WHEELBARROW
RACING_CAR
BATTLESHIP
THIMBLE
PIECE_TYPE(int)
value int
DPlayer
DPlayer(PIECE_TYPE, int)
currentDiceVal int
totalDiceValue int
bankruptFlag boolean
balance int
cycleCounter int
playerDice BDice
piece_type PIECE_TYPE
roundValue int
location int
bankrupted boolean
DPieceTest
getValue() void
«create»
DPlayerTest
dPlayer DPlayer
beforeAllTestMethod() void
getCurrentDiceVal() void
getLocation() void
getBalance() void
getCycleCounter() void
getTotalDiceValue() void
getPlayerDice() void
isBankrupted() void
getPiece_type() void
DInstruction
countOfPlayers long
startMoney long
countOfTaxSquares long
priceOfTaxSquares long
priceOfGoSquare long
instance DInstruction
DInstruction(long, long, long, long, long)
getInstance() DInstruction
getInstance(long, long, long, long, long) DInstruction
toString() String
DJSONReader
currentJSONObject JSONObject
DJSONReader(String)
getValueUsingKey(String) Object
initInstructionObject() void
DInstructionTest
testToString() void
«create»
DJSONReaderTest
getValueUsingKey() void

4.2 Analysis Domain Model



Stakeholders

- Murat Can GANİZ (Customer)
- Serap KORKMAZ (Customer)
- Ahmet LEKESİZ (Programmer)
- Mücahit TANACIOĞLU (Programmer)
- Muhammed Bera KOÇ (Programmer)
- Salih ÖZYURT (Programmer)