INTERNATIONAL HELLENIC UNIVERSITY
SCHOOL OF ENGINEERING
DEPARTMENT OF INFORMATICS, COMPUTER AND TELECOMMUNICATIONS ENGINEERING

BOARD GAME GENERATOR

FINAL REPORT

Eighth Week

Project Team:

Anastasiades Alkinoos (20003)

Zina Eleni (20046)

Lagiokapas Dimitrios (20079)

Makri Styliani (20060)

Supervisors:

Koureas Argyrios

Lantzos Theodoros

SERRES, 29 MAY to 2 JUNE 2023

Contents

Summary

Board Game Generator is an application designed to automate the creation of electronic board games to some extent. The project aims to provide users with a way to build their own game with their own parameters, as long as they know how to handle JSON files, as well as to provide two ready-made template-games for the user to play. The present project has gone through many stages of changes and in its final form it is expandable, with the help of services and specific design patterns, and easy to manage in case the user wants to change the parameters of the game.

Introduction

The application reads the description of a game in JSON format and then creates the game. The execution of the game is done in text form, there are no graphics. The procedure followed by the team was as follows:

Week 1: The “base”" of the application and a simple game were implemented where whichever player reaches the end of the board first, wins.

Week 2: Enrichment of the code and extension of the program architecture were made so that the game elements are derived from JSON files.

Week 3: Design of the dashboard of the endless type of game and implementation of a simple service aimed at understanding the operation of the services, as it was necessary to integrate them into the program architecture.

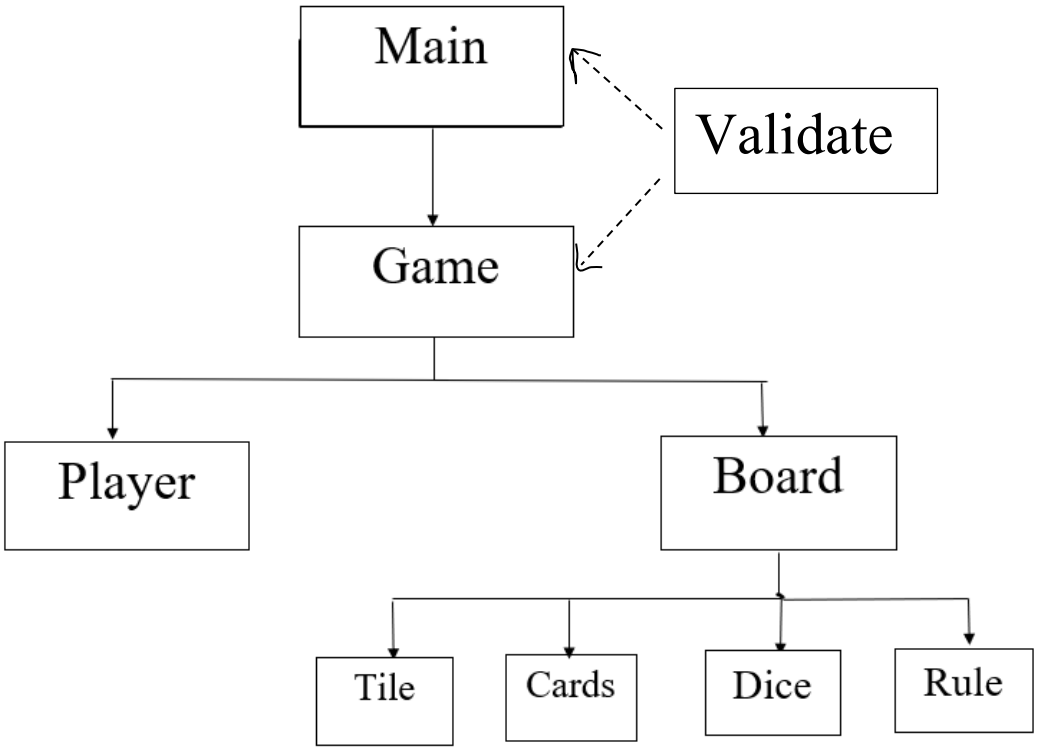
Week 4: Implementation of separate classes for each parameter of the games such as limited and endless board, tiles for limited and endless game type, etc. (This logic did not continue and changed in the following weeks because the program was too complicated.)

Week 5: Removed the unnecessary classes mentioned above, completed the design of all interfaces and developed a Factory Pattern for tiles and cards.

Weeks 6 – 7: Code improvements and added ability for users to import their own JSON file and play the created game.

1. Methodology

Program structure and design:



3

Analysis of the main classes of the program:

The Main class is the entry point of the application. Asks the user to choose a game type. It then delegates the user's selection to the appropriate methods to run. This class acts as the main organizer for the initial creation of the game.

The Game class includes the logic of creating and running the various types of games, i.e. the functions for limited and endless games that contain cards and tiles with special properties.

The new Validate class is somewhat helpful for Main and Game, which simply has a function to check whether the user's choice is an integer.

```java
public class Validate {

    2 usages
    public static int input() {
        Scanner sc = new Scanner(System.in);
        int choice;
        do {
            try {
                choice = Integer.parseInt(sc.next());
                break;
            } catch (NumberFormatException e) {
                System.out.println("Not an integer. Try again.");
            }
        }while(true);
        return choice;
    }
}
```

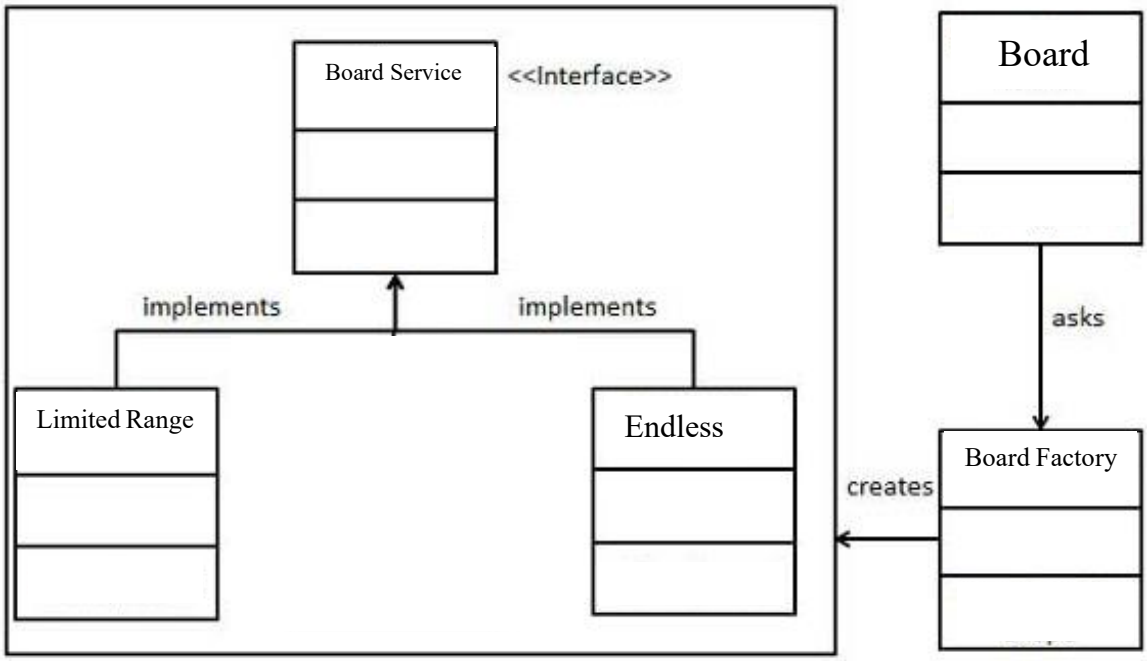
The TilesFactory class is responsible for the dynamic creation of different types of tiles based on JSON specifications. It uses the TileOperations package, which defines the different types of tiles, such as simple tiles, card tiles, point tiles, etc. By using Factory Pattern we can easily add new types of tiles, without affecting the rest of the code. It is one of the features that makes the application extensible.

Another implementation of Factory Pattern is in BoardOperations:

```java
public class BoardFactory {

    1 usage
    public static BoardService getInstance(JSONObject gameObj) throws Exception {
        String boardType= gameObj.get("board_type").toString();
        switch(boardType){
            case "limited_range":
                return new LimitedRangeBoard();
            case "endless":
                return new EndlessBoard();
            default:
                throw new Exception();
        }
    }
}
```

Factory Pattern Model for our case:



2. Implementation

Presentation of the program execution:

We see the following once the program is running:

```
Welcome to the Game Generator.
Choose game from the Game Generator:
1. Limited range board game of progress and setbacks.
2. Endless board game containing cards and special abilities.
3. Initialize your own game with one of these board types above and play.
0. Exit.
```

Here the user will choose what type of game they want to play. The Limited Range game, Endless or its own game that will be loaded from the JSON file provided by the user. Suppose the user selects the Limited game type:

```
Give the name of player 1.
P1
Give the name of player 2.
P2
Give the name of player 3.
P3
Give the name of player 4.
P4
Game rules:
Rule 1: There are some positions that move you back and some others that get you closer to the finishing line.
Rule 2: If you're out of bounds, you'll be sent to the position of the remaining amount of dice's roll.
Rule 3: The first who gets to the exact end of the board is the winner.
P1 is playing now.
Choose from the following options:
1. Roll the dice(s).
2. Display rules.
3. Exit.
```

The generator asks for the names for each player and displays the rules of the game.

Then the game starts and each player plays in turn.

```
Choose from the following options:
1. Roll the dice(s).
2. Display rules.
3. Exit.
1
Value of dice: 4
P1 is on tile 4 which moved them by 3 to tile 7.
P2 is playing now.
Choose from the following options:
1. Roll the dice(s).
2. Display rules.
3. Exit.
```

At the end of each round, the scoreboard is presented, where each player is:

```
State of board:
Id | Player's name |  Tile  |
 1 | P1            |      7 |
 2 | P2            |      8 |
 3 | P3            |      3 |
 4 | P4            |     13 |
```

When a player reaches the end of the board first, the game stops and the message appears:

```
P2, you have reached the end of the board.
P2 is the winner.
```

Below we see the same implementation but in the endless game:

```
Welcome to the Game Generator.
Choose game from the Game Generator:
1. Limited range board game of progress and setbacks.
2. Endless board game containing cards and special abilities.
3. Initialize your own game with one of these board types above and play.
0. Exit.
2
Give the name of player 1.
P1
Give the name of player 2.
P2
Give the name of player 3.
P3
Give the name of player 4.
P4
Game rules:
Rule 1: All players start with 200 points.
Rule 2: Gameplay ends after 30 rounds.
Rule 3: The winner is whoever has maximum amount of points out of all the players.
Rule 4: The starting tile gives +200 points whenever the players passes it.
Rule 5: If a player loses all their points, they get eliminated from the game.
Rule 6: If a player lands on a tile after an action, they don't have the new tile's action applied on them.
```

In the Endless game type, tiles have specific properties, as in this case, the player draws a card:

```
P3 is playing now.
Choose from the following options:
1. Roll the dice(s).
2. Display rules.
3. Exit.
1
Value of dice: 2
Type of tile: card
P3 is on tile 2 and picked a random card.
Action of card: Go to the start.
P3 moved to the beginning tile and got 200 so now they have 400 points.
```

Another example with a green tile:

```
P1 is playing now.
Choose from the following options:
1. Roll the dice(s).
2. Display rules.
3. Exit.
1
Value of dice: 6
Type of tile: green
P1 is on tile 9 which moved them by -7 to tile 2.
P2 is playing now.
```

Once a round is over, players see what position they are in and how many points each has:

```
State of board:
Id | Player's name |  Tile  |  Points  |
 1 | P1            |     10 |      300 |
 2 | P2            |      8 |      200 |
 3 | P3            |      0 |      400 |
 4 | P4            |      5 |      400 |
```

When the 30 rounds are completed, the players' ranking is displayed and the program ends:

```
State of board:
Id | Player's name |  Tile  |  Points  |
 2 | P2            |     26 |     2210 |
 3 | P3            |     18 |     1770 |
 1 | P1            |     31 |     1520 |
 4 | P4            |     26 |     1485 |
P2 is the winner.
```

3. Timetable

29/5-1/6:
Team: Preparing for the presentation and drafting of the report.

4. Results-Conclusion

The results of the project are as follows:

- Development of a Board Game Generator that allows users to choose between different types of board games and play them.
- Implementation of a limited type of game and an endless game, each with its respective characteristics.
- Create an extensible design with the help of the principles of Object Oriented Programming, Services and Design Patterns.
- Integration of JSON files to define game settings, tiles, cards, number of players, board size, game type and rules.

In conclusion, by working on the Board Game Generator, the team has successfully developed an application for creating and running board games. The project demonstrates the ability of the program to produce games with various parameters, and the ease of personalization in almost all aspects of games. While there were changes in the rationale and logic of the program during the eight weeks of project, the team was able to work together efficiently to develop the program and complete the project, as well as the members to collect useful knowledge and experience in a practical way.