Bilkent University

Department of Computer Engineering

CS319 Term Project

Group: 1I-TM

Spring 2020

Design Report

Team Members

Rafi Çoktalaş

Zeynep Cankara

Kamil Gök

Efe Macit

Arda Gültekin

Instructor: Eray Tüzün

Teaching Assistant(s): Alperen Çetin

Contents

1. Introduction **4**
1.1. Purpose of Terra Mystica 4
1.2. Design Goals 4
1.2.1. Documentation 4
1.2.1.1. Internal Documentation 4
1.2.1.2. External Documentation 5
1.2.2. Usability 5
1.2.3. Robustness 5
1.2.4. Extensibility 6
1.2.5. Performance 6
1.2.5.1. Response Time 6
1.2.5.2. Memory 6
1.2.5.3. Network 6
1.2.6. Maintainability 6

2. High-level Software Architecture **7**
2.1. Subsystem Decomposition 7
2.1.1 ClientInterface 8
2.1.2 GameInterface 8
2.1.3 MainMenuInterface 8
2.1.4 HelpInterface 8
2.1.5 GameLogic 8
2.1.6 GameManager 9
2.1.7 GameFlow 9
2.1.8 GameBoard 9
2.1.9 Player 9
2.1.10 Server 9
2.2. Hardware/Software Mapping 10
2.3. Persistent Data Management 10
2.4. Access Control and Security 11
2.5. Boundary Conditions 11
2.5.1. Initialization (Start-up) 11
2.5.2. Termination (Shutdown) 12
2.5.3. Exceptions 12
2.5.3.1. Loss of network connection: 12

2.5.3.2. Failure of user authentication: 12
2.5.3.3. Inconsistent data: 12
2.5.3.4. Corrupted persistent data: 13

3. Subsystem Services 13

4. Low-level Design 14
4.1. Object Design Trade-offs 14
4.1.1. Documentation vs. Development time 14
4.1.2. Space complexity vs. Time complexity 14
4.1.3. Extensibility vs. Complexity 14
4.1.4. Usability vs. Functionality 14

4.2. Object Diagram 15
4.2.1 MainController Class 16
4.2.1.1 Attributes of MainController Class 16
4.2.1.2 Methods of MainController Class 16
4.2.2. Game Class 18
4.2.2.1 Attributes of Game Class 18
4.2.2.1 Methods of Game Class 19
4.2.3. Player Class 20
4.2.3.1 Attributes of Player Class 21
4.2.3.2 Constructor of Player Class 22
4.2.3.3 Methods of Player Class 22
4.2.4 CultBoard Class 23
4.2.4.1. Attributes Of CultBoard Class 23
4.2.5. CultTrack Class 24
4.2.5.1 Attributes of CultTrack Class 24
4.2.6 Faction Class 24
4.2.6.1 Attributes of Faction Class 24
4.2.6.2 Methods of Faction Class 25
4.2.7 GameBoard Class 25
4.2.7.1 Attributes of GameBoard Class 25
4.2.7.2 Methods of GameBoard Class 25
4.2.8 Terrain Class 26
4.2.8.1 Attributes of Terrain Class 26
4.2.8.2 Methods of Terrain Class 26
4.2.9 Structure Class 27
4.2.9.1 Attributes of Structure Class 27
4.2.9.2 Method of Structure Class 27
4.2.10 GameUtil Class 28

4.2.10.1 Attributes of GameUtil Class 28
4.2.10.2 Methods of GameUtil Class 28
4.2.11 Dwelling Class 28
4.2.11.1 Methods of Dwelling Class 28
4.2.12 Temple Class 29
4.2.12.1 Methods of Temple Class 29
4.2.13 TradingHouse Class 29
4.2.13.1 Methods of TradingHouse Class 29
4.2.14 Sanctuary Class 29
4.2.14.1 Methods of Sanctuary Class 30
4.2.15 Stronghold Class 30
4.2.15.1 Methods of Stronghold Class 30
4.2.16 ScoringTile Class 30
4.2.16.1 Attributes of ScoringTile Class 30
4.2.16.2 Methods of ScoringTile Class 30
4.2.17 ScoringTileList Class 31
4.2.17.1 Attributes of ScoringTileList Class 31
4.2.17.2 Methods of ScroingTileList Class 31
4.2.18 FavorTile Class 31
4.2.19 FavorTileList Class 31
4.2.19.1 Attributes of FavorTileList Class 32
4.2.20 TownTile Class 32
4.2.21 TownTileList Class 32
4.2.21.1 Attributes of TownTileList Class 32
4.2.22 BonusCard Class 32
4.2.23 BonusCardList Class 32
4.2.23.1 Attributes of BonusCardList Class 33
4.2.24 Tile Class 33
4.2.24.1. Attributes of Tile Class 33
4.3. Packages 33
4.4. Class Interfaces 33

5. Glossary & References 34

Appendix 34

Design Report

Terra Mystica

1. Introduction

1.1. Purpose of Terra Mystica

Terra Mystica is designed to accurately and completely digitalize the Terra Mystica boardgame. Terra Mystica will have all the components and game mechanics of his real world counterpart. Being in a digital environment adds flexibility and room for new features to be added.

Terra Mystica will be a multiplayer game and will be played online via Internet connection by two to five people. Terra Mystica will be compatible with Windows and MacOS.

The intended audience for Terra Mystica is only limited by knowing how to use a computer and viable Internet connection. Having a large scope will be kept in mind in external documentations.

1.2. Design Goals

1.2.1. Documentation

1.2.1.1. Internal Documentation

Each class and method will have javadoc comments which can later be used to generate an automatic documentation in HTML format [1]. Moreover, every method that contains an algorithm will have the algorithm in the source code as a pseudocode. This is to increase the readability of the code for the developers. However, this approach will also increase the writability as information in other classes and methods will be easy to understand for the developers all the time.

Internal documentation also includes pre-meeting documents in which the duration, topics and goal of a meeting is depicted and post-meeting documents which

is the summary of a meeting. Post-meeting documents consist of the tools to be used and coding practices to follow.

1.2.1.2. External Documentation

Terra Mystica will have a in game help screen and also a tutorial document for users. Also the taken approaches, list of algorithms used will be documented for fellow developers.

1.2.2. Usability

The user interface will consist of 4 screens: login, main menu, help, and game. Other than the game screen, the screens will have minimal graphical elements to ease the entry of a user to the game. Also these screens will consist of buttons that are accessed directly and will equip users with a go back button thus providing users with simple navigation.

The game screen will inevitably have complex graphical elements to mimic the real board game. To overcome any inconvenience regarding usability several strategies will be employed. Firstly, a status label will always be visible on the game screen. The status label will narrate the last event,e.g., a message saying that the player is trying to take an action that is against the rules or simple messages explaining the change in the gameboard resulting from the last action. Secondly, the components will have distinct images and labels. Thirdly, the turn order and whose turn currently is being played will be shown all the time. Lastly, all components of the game which do not have to be visible at all times will be hidden to make room for those that have to be visible all the time. By doing so, players will have a cleaner graphical interface to see all the time. The hidden components will be accessible via distinct buttons representing them.

1.2.3. Robustness

The program will not act on invalid inputs on the game screen, i.e., a player trying to take action in another player’s turn. Furthermore, players will be notified why the action they have tried to take is invalid.

1.2.4. Extensibility

Even though Terra Mystica is a highly complex board game, there is always room for improvement; therefore the development will proceed in a way that will ease the addition of new functionalities or classes. So as to do so, fewer and cleaner dependencies; therefore, low cohesion and high coupling, will always be aimed for. Moreover, interfaces will be used to have further extensibility.

1.2.5. Performance

1.2.5.1. Response Time

In the game, players will see the outcome of the action they have taken after less than 1 second. Also other players will also be able to see the outcome of the action in a similar time. Moreover, the outcome of the action on the game will be narrated in the status bar. The status bar will be updated within 1 second when an action is happened or failed.

1.2.5.2. Memory

Terra Mystica will require less than 250 MB of space. To achieve this, algorithms will be investigated in terms of their space complexity and image files will be compressed.

1.2.5.3. Network

Users should be able to connect and play the game via a network connection with 1 Mbps bandwidth or higher.

1.2.6. Maintainability

The application will be easy to maintain. In order to achieve this, the source code will go under several reviews after it is written and before it is added to the application. The source code will be tested for maintainability measures with help of tools like Testwell CMTJava [2].

2. High-level Software Architecture

2.1. Subsystem Decomposition

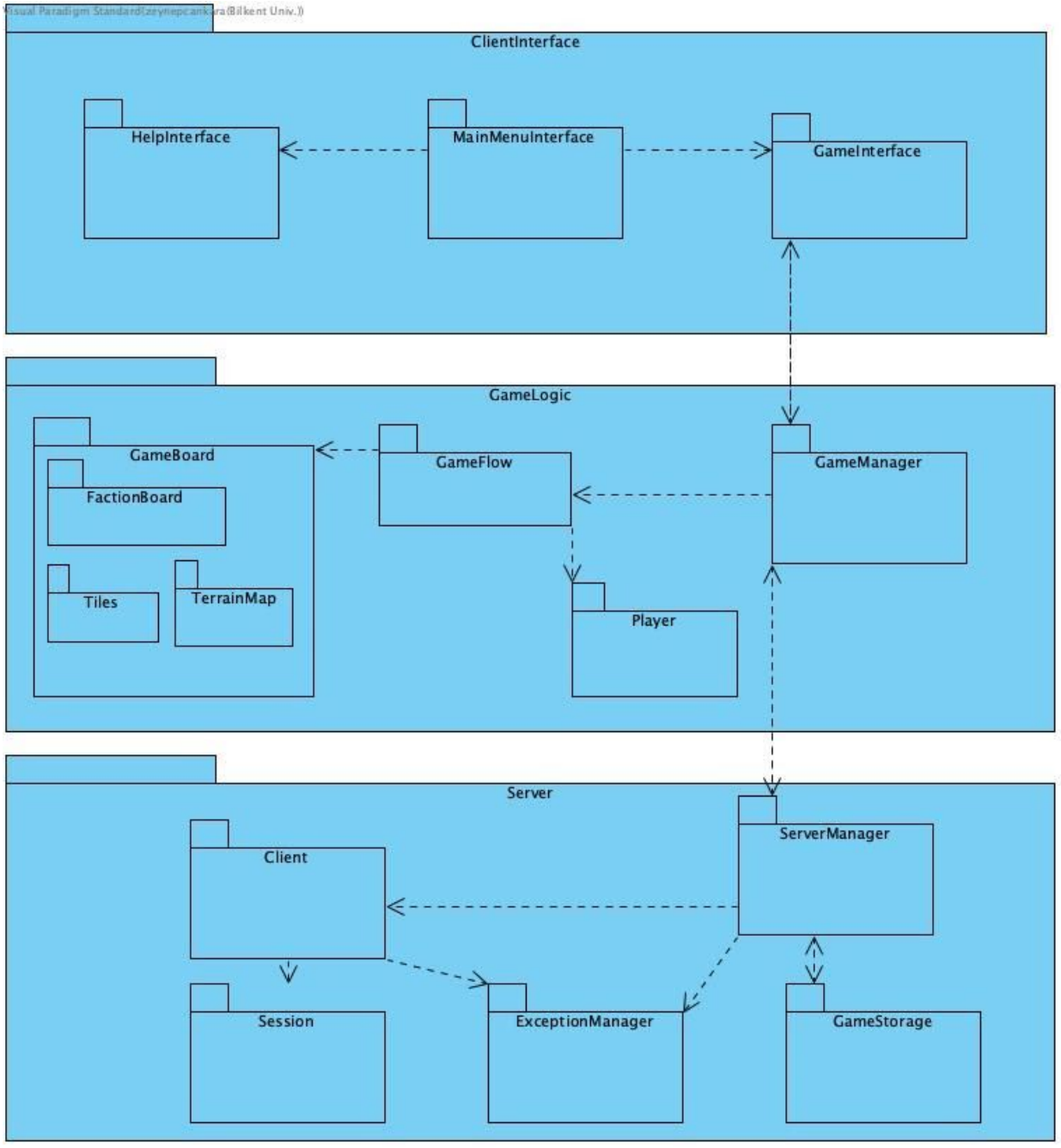

Figure 1: Subsystem Decomposition Diagram

We have decided to use 3-layer architectural style while designing our system. Our presentation layer consists of subsystems related with our User Interface hence named as ClientInterface contains subsystems including HelpInterface, MainMenuInterface, GameInterface. Our Application Logic Layer contains the game controller and application domain entities including subsystems GameManager, GameFlow, GameBoard, Player. Our data layer contains subsystems related with managing user data including ServerManager, GameStorage, ExceptionManager, Client and Session.

2.1.1 ClientInterface

Subsystem responsible of managing the View of the game. Includes subsystems responsible of handling the UI.

2.1.2 GameInterface

Handles UI View of the board game screen.

2.1.3 MainMenuInterface

Handles UI View of Main Menu.

2.1.4 HelpInterface

Handles UI View of the Help Menu.

2.1.5 GameLogic

Subsystem responsible of managing the Model and Controllers of the game. Includes subsystems containing application domain models and game logic controllers which are able to fetch and send information to data layer.

2.1.6 GameManager

Representation of the game logic controller of the game. Bridges the connection between the game model and UI. Also sends/receives information from the server.

2.1.7 GameFlow

Subsystem managing the application domain entities

2.1.8 GameBoard

Subsystem responsible of holding the board data of the game.

2.1.9 Player

Subsystem representing the player of the game.

2.1.10 Server

Our server communications will handled by Java Admin SDK of Firebase Database. The server subsystem will contain subsystems managing client sessions and managing the state of the game by sending/receiving messages from the players. Additionally it will be responsible of sending notifications to users in order to handle exception incidents.

2.2. Hardware/Software Mapping

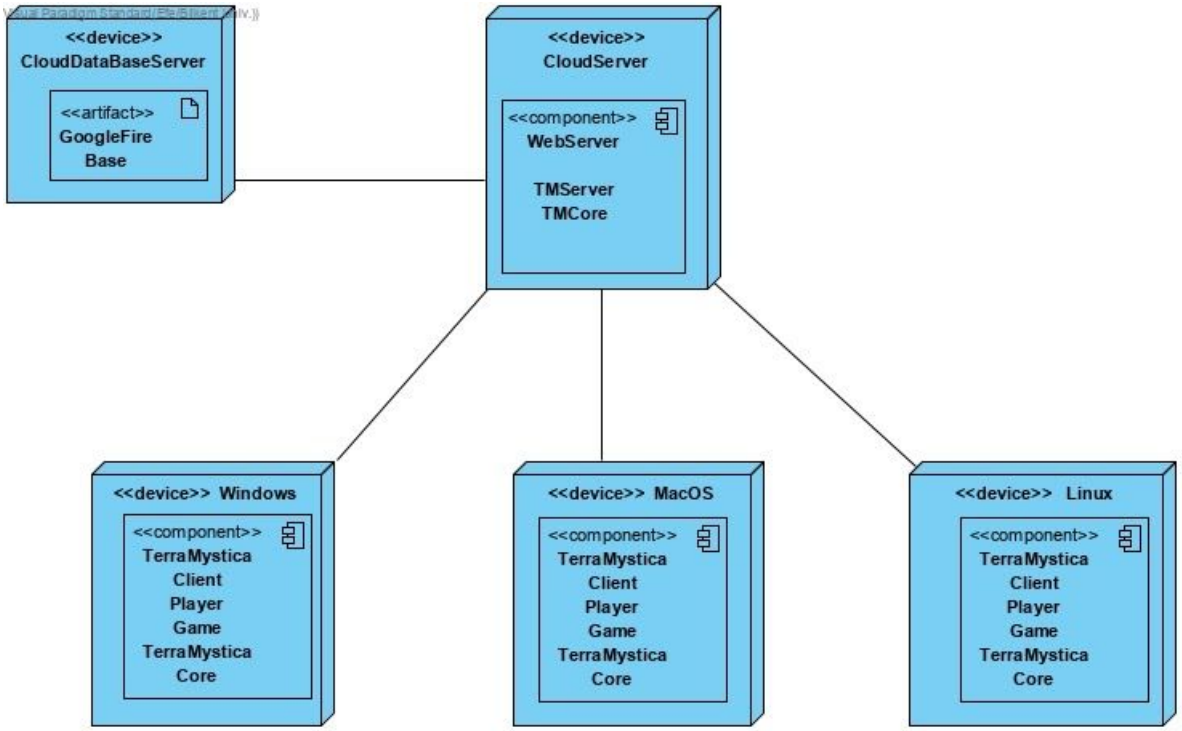

Figure 2: Hardware/Software Mapping Deployment Diagram

2.3. Persistent Data Management

Images will be kept in local hard disks of the users as the part of the client package. This will be prevent unnecessary data flow. Initial map design is holded in json file which is received from server and game maps will set up according to this initial file on the client. Database holds the registered players usernames and passwords. Server manages the all data which affects the multiplayer game environment. Server is independent from local application. There is no saved or load game options in the game. Once the player has disconnect, game cannot be loaded or if any errors occurred in the server, game cannot be loaded in the players.

2.4. Access Control and Security

There is no single player mode, in multiplayer all players have to register and after that must login. There is no authorization in the game concept in the other words there is no administrator or player priority. All players have same roles and there is no privilege between them. All players can take actions in just their turns, they are turned just a basic spectator on the players turns.

2.5. Boundary Conditions

2.5.1. Initialization (Start-up)

In order to initialize the game, all UI componants such as fxml files, images and sound must load to the local system. The game will be in an executable program. When user opens the executable of the game a fxml file associated with main menu window called as mainMenu.fxml will loaded. If user wants to play single player game localGame.fxml will loaded which is the file associated with single player game UI. After the images of the game loaded from the local hardware game will be ready to start reliably and stay stable throughout the game.

Multiplayer mode will require different initialization process which involves verifying the user credentials to enable to store the user information in cloud. The information stored in cloud will involve previous gaming sessions user attended together with the players involved in the gaming session. Once user clicks the “play online” button user will directed to a login page where user being asked to enter credential information which will be user’s gmail address with password. After user credentials verified, player will be directed to a new screen where server matches the player with four random players. After four other players found. The game will initialized from gameView.fxml file together with other game assets. During the initialization stage of the game server will be responsible of random initialization of the game board map and player resources. Afterwards each player’s instances will controlled by the server in realtime.

2.5.2. Termination (Shutdown)

If the user plays the game in single player mode. The user either terminates execution of the game or presses the “Exit” button and returns to the main menu screen of the game. In case of multiplayer game mode, user termination might be result of poor network connection. The user will encounter main menu window when they exit the game due to network failure. Furthermore if the user wants exit the game in the multiplayer mode, user presses the “Exit” button and returns to the main menu screen.

2.5.3. Exceptions

In this section we will address how we will react the exeptions and failures which can occur independent of the game.

2.5.3.1. Loss of network connection:

A user’s network can fail during the any part of the online playing mode (login, player mathcing, online game). When a network connection failure happens player will return to the main menu screen. If the connection failure happens during the online game all players will return to the main menu screen with a notification of which user disconnected from the game.

2.5.3.2. Failure of user authentication:

Users will need to provide their authentication information for playing the game online from a server. The authentication method will b e combination of email-password combination since we are relying on Google Firebase Database for keeping user data in cloud while getting Google’s promise of safety and security of the cloud database system. Authentication exception will occur if user tries to provide inaccurate credential information which does not exist in the system database.

2.5.3.3. Inconsistent data:

This type of system exception might occur when player is at both online play and local play modes. Ideally one player must notified by all the changes happening throughout the game. In case of local play mode the view might fail to update itself resulting displaying data inconsistent with the game state. In case of online play

mode, the exception might occur if not all players modified by the update one player has on the game in real time.

2.5.3.4. Corrupted persistent data:

This type of exception might occur when fxml file fails to load UI assets including images and sounds. One of our most important design goal is Usability since we want out users to perform tasks safely, reliabily and efficiently. Thus, when data corruption noticed by the system the system stops trying to load the game and notifies the user about the corrupted data.

3. Subsystem Services

Since we used 3-Tier Architectural Style in our design, main subsystems are user interface(view subsystem), the game logic and the controller.

- User Interface : UI includes different minor subsystem services. These services are, game interface which provides the main game board view, main menu interface which provides the menu selections for the game such as number and name of players. Also, help interface enables the view for the help menu and login interface consists of login and signup services.
- Game Logic : This layer consist of object-oriented design of the game and includes all of the services that are included in the game flow such as game board, player, tiles, cult tracks.
- Controller : This division includes 2 main services.These services are providing communication of game logic with the remote servers for multiplayer games and also providing methods for connecting databases.

4. Low-level Design

4.1. Object Design Trade-offs

4.1.1. Documentation vs. Development time

Detailed documentation is aimed in this project but the more detailed the documentation the more it requires time. Therefore, documentation may affect the development time negatively. However, detailed internal documentation will serve the project greatly in the long run in terms of maintenance and improvement; external documentation will introduce application to the users better.

4.1.2. Space complexity vs. Time complexity

In order to have the game require less space algorithms will prioritize space over time. The game consists of relatively small number of inputs for its functions so time complexity can be compromised on. Furthermore, data used will be compressed to save space but compression will take time.

4.1.3. Extensibility vs. Complexity

Having extensibility concerns may result in introduction of new interfaces and classes that would not be required otherwise. Thus, complexity of the source code may increase. This increase in complexity will not affect the application greatly yet being extensible may lead to great additions.

4.1.4. Usability vs. Functionality

Other than the game screen, usability will be the greater concern since these screen will not house much functionality whereas on the game screen user interactions may become unpleasant to some extent in order to capture all the functionality of Terra Mystica. This will be minimized and heavily explained in the external documents and in game help screen.

4.2. Object Diagram

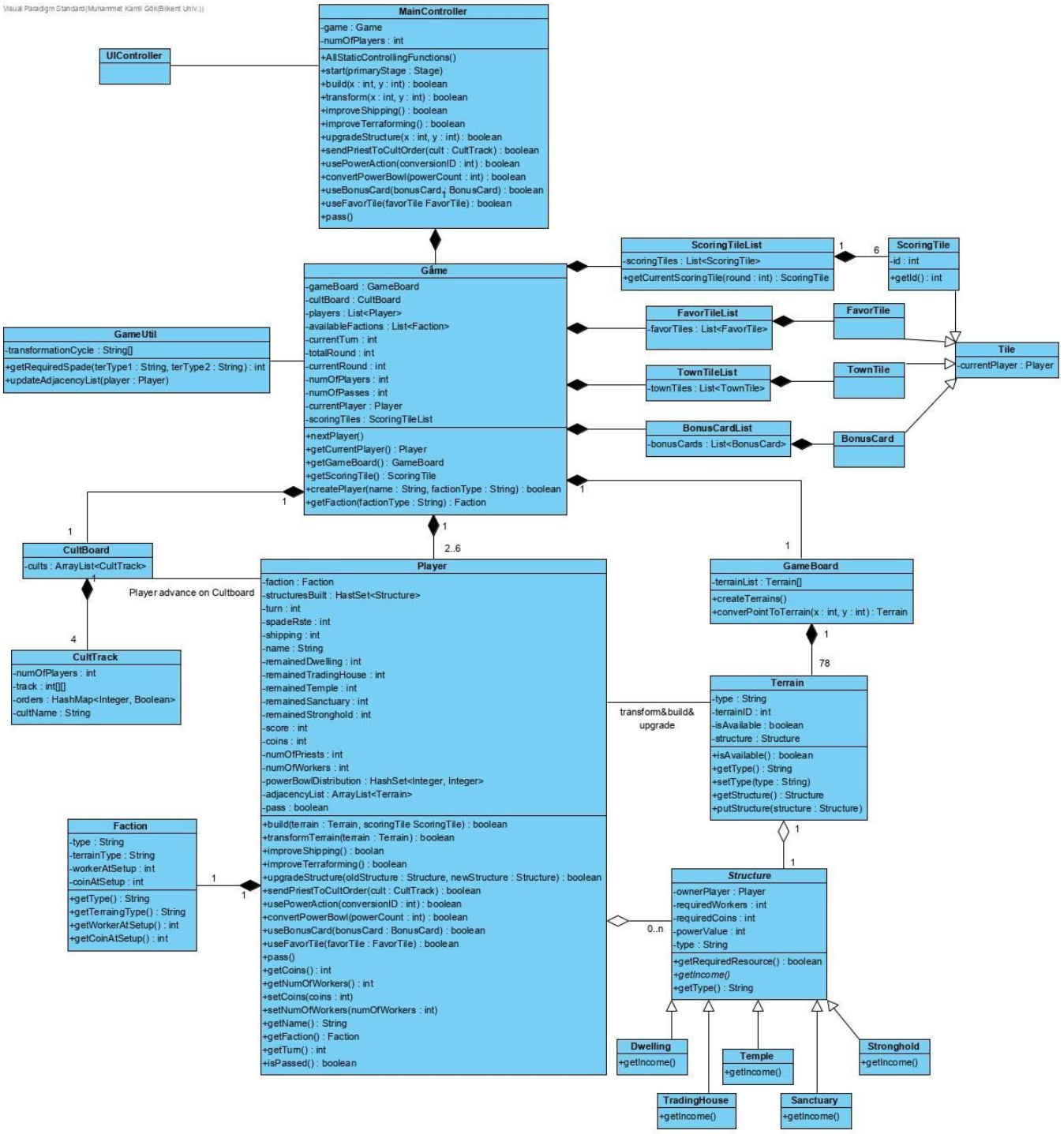

Figure 3: Object Diagram(see Appendix)

4.2.1 MainController Class

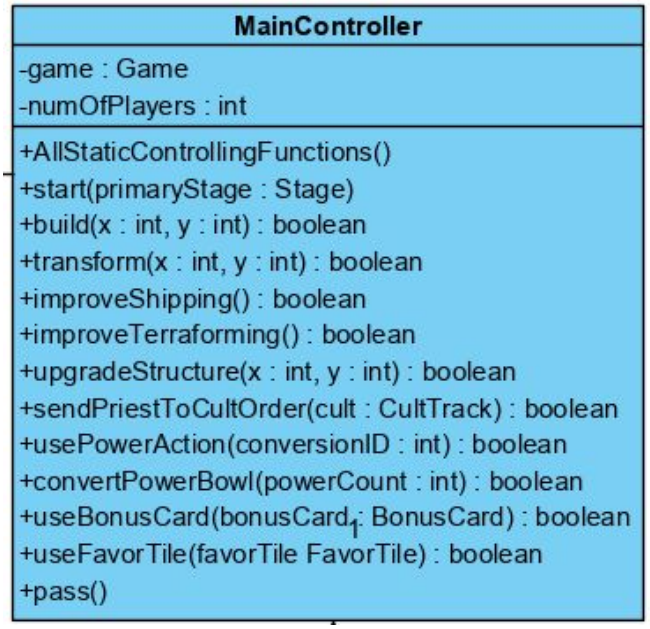

Figure 4: MainController Class Diagram

This class provides connection between user interface package and object model. The instance of Game class is created in MainController class. Object model is reached through the the instance of Game class.

4.2.1.1 Attributes of MainController Class

private static Game *game*: It is used to keep the game object.

private static int numOfPlayers: It is used to store number of players in the game.

4.2.1.2 Methods of MainController Class

public static boolean build(int x, int y): Converts chosen location to the equivalent terrain and calls build function of the current player of the game.

public static boolean transform(int x, int y): Converts chosen location to the equivalent terrain and calls transform function of the current player of the game.

public static boolean improveShipping(): Calls improveShipping function of the current player of the game.

public static boolean improveTerraforming(): Calls improveTerraforming function of the current player of the game.

public static boolean upgradeStructure(int x, int y): Calls upgradeStructure function of the current player of the game.

public static boolean sendPriestToCultOrder(CultTrack cult): Calls sendPriestToCultOrder function of the current player of the game.

public static boolean usePowerAction(int conversionID): Calls usePowerAction function of the current player of the game.

public static boolean convertPowerBowl(int powerCount): Calls convertPowerBowl function of the current player of the game.

public static boolean useBonusCard(BonusCard bonusCard): Calls useBonusCard function of the current player of the game.

public static boolean useFavorTile(FavorTile favorTile): Calls useFavorTile function of the current player of the game.

public static void pass(): Calls pass function of the current player of the game.

4.2.2. Game Class

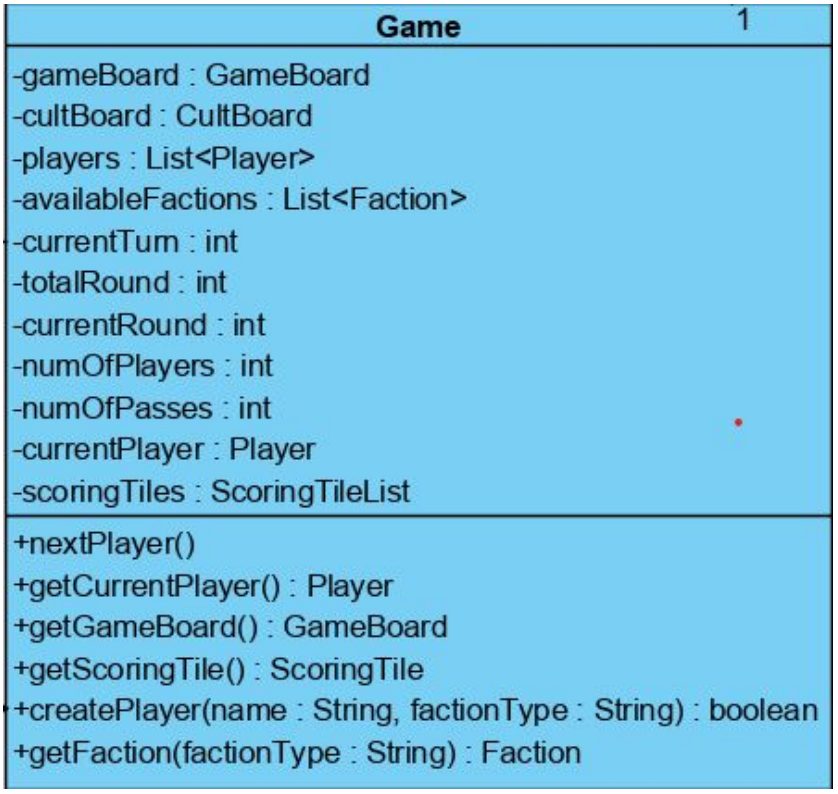

Figure 5: Game Class Diagram

Everything related to the game is connected to the Game Class, it keeps track of the game progress with its attributes and functions.

4.2.2.1 Attributes of Game Class

private Player currentPlayer : It stores the player object whose round is now.

private ArrayList<Player> players: It is used to store all the players playing the game.

private int totalRound : It is used to store the total rounds in the game.

private int currentRound : It is used to store the current round number in the game.

private int currentTurn : It is used to store the current turns number.

private int numOfPlayers: It is used to store the total number of players playing the game.
private ArrayList<Faction> availableFactions : It is used to store all the available factions in the game.
private CultBoard cultBoard : It is used to store the cult board progress.
private GameBoard gameBoard: It is used to store the changes in game board / game.
private int numOfPasses : It is used to store the number of passes in the round.
private ScoringTileList scoringTiles: It is used to store the scoring tile list.

4.2.2.1 Methods of Game Class

public void nextPlayer() : When the currentPlayer is done with the turn and used actions, the nextPlayer() function is called to get the next player .
public Player getCurrentPlayer() : It is used to get the currentPlayer object.
public GameBoard getGameBoard(): It is used to get the Game Board object.
public ScoringTile getScoringTile() : It gets the current scoring tile of the round.
public void createPlayer(String name,String factionType): It is called when the game starts. Initially game class calls it to create the desired number of players one by one.
public Faction getFaction():

4.2.3. Player Class

Figure 6: Player Class Diagram

Player class includes all attributes that belongs to a players and includes functions for the player actions. Also, the class has getter and setter functions for the attributes. This class' functionalities constitute the main flow and actions of the game.

4.2.3.1 Attributes of Player Class

private String name: It is used to store the name of the player.

private Faction faction: It is used to store the chosen faction of the player.

private int turn: It is used to indicate the turn number of the player.

private int remainedDwelling: It is used to store remained usable dwelling count.

private int remainedTradingHouse: It is used to store remained usable trading house count.

private int remainedTemple: It is used to store remained usable temple count.

private int remainedSanctuary: It is used to store remained usable sanctuary count.

private int remainedStronghold: It is used to store remained usable Stronghold count.

private HashSet<Structure> structuresBuilt: It is used to store the structures built on the terrains.

private int score: It is used to store the score -victory points-.

private int coins: It is used to store the total number of coins owned.

private int numOfPriests: It is used to store the total number of priests owned.

private int numOfWorkers: It is used to store the total number of workers owned.

private HashMap<Integer,Integer> powerBowlDistribution: It is used to keep the number of power tokens in each of 3 power bowls.

private ArrayList<Terrain> adjacencyList: It is used to keep the adjacent terrains list.

private int shipping: It is used to store shipping skill.

private int spadeRate: It is used to store the conversion rate from workers to spades.

private boolean pass: It is used to indicate whether the player passed or not.

4.2.3.2 Constructor of Player Class

public Player(String name, Faction faction, int turn): Initializes player object with the given name, chosen faction and given turn number.

4.2.3.3 Methods of Player Class

public boolean build(Terrain terrain, ScoringTile scoringTile): If the terrain is empty and the player has enough resources, it builds dwelling on the given terrain. Then, gets the income both from the structure and the scoring tile if it is possible.

public boolean transformTerrain(Terrain terrain): Transforms the terrain if it is empty and the player has enough resources.

public boolean improveShipping(): Improves shipping skill by using required resources.

public boolean improveTerraforming(): Improves terraforming skill by using required resources.

public boolean upgradeStructure(Structure oldStructure, String newStructure): Upgrades old structure to the new structure by using required resources. Then, it gets the income from the new structure.

public boolean sendPriestToCult(int cultID): Sends priest to one of the cults' order in order to advance on that cult track.

public boolean usePowerAction(int powerActionID): Uses power tokens in bowl 3 and move those tokens to bowl 1 in order to have coins, workers, priests, spades or bridges.

public boolean convertPowerBowl(int powerCount): Moves power tokens from bowl 2 to bowl 3 by sacrificing power tokens.

public boolean useBonusCard(BonusCard bonusCard): Uses given bonus card accordingly.

public boolean useFavorTile(FavorTile favorTile): Uses given favor tile accordingly.

public void pass(): Passes and waits for the other players to pass.

public int getCoins(): Gets the total number of coins.

public int getNumOfWorkers(): Gets the total number of workers.

public void setCoins(int coins): Sets the number of coins.

public void setNumOfWorkers(int numOfWorkers): Sets the total number of workers.

public String getName(): Gets the name of the player.

public Faction getFaction(): Gets the faction of the player.

public int getTurn(): Returns the turn number of the player.

public boolean isPassed(): Checks if the player passes or not.

4.2.4 CultBoard Class

Figure 7: CultBoard Class Diagram

This class is used to represent the cult board status in the game. It has a single attribute which is list of CultTrack. Which will provide four different cult tracks in our game.

4.2.4.1. Attributes Of CultBoard Class

private <CultTrack> cults : The CultBoard class has a single attribute which stores 4 different cult tracks. And it is all stored in this array list.

4.2.5. CultTrack Class

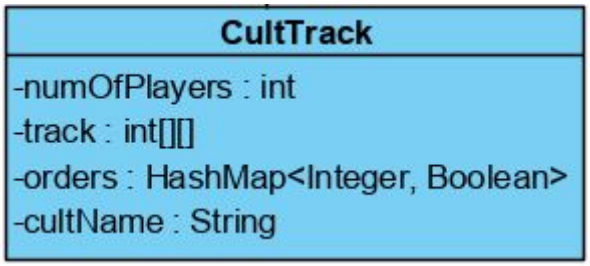

Figure 8: CultTrack Class Diagram

This class is used to keep track of a single cult track in the cult board. It stores the current situation of the individual cult track.

4.2.5.1 Attributes of CultTrack Class

private int numOfPlayers : It is used to store the number of players.

private int[][] track : It is used to store cult's track status.

private HashMap <Integer,Boolean> orders :

private String cultName : It is used to store the name of the cult.

4.2.6 Faction Class

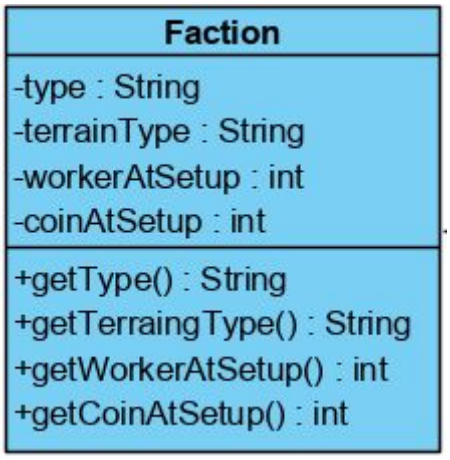

Figure 9: Faction Class Diagram

It is used to represent the Faction types in our game.

4.2.6.1 Attributes of Faction Class

private String type : It stores the factions type.

private String terrainType: It stores the factions corresponding terrain type in string.
private int workerAtSetup: It stores the factions number of workers initially.
private int coinAtSetup: It stores the factions number of coins initially.

4.2.6.2 Methods of Faction Class

public String getType(): It returns the type of the faction in string.
public String getTerrainType(): It returns the terrain type in string.
public int getWorkerAtSetup(): It returns the number of workers in setup.
public int getCoinAtSetup(): It returns the number of coins at setup.

4.2.7 GameBoard Class

Figure 10: GameBoard Class Diagram

GameBoard class is used in our implementation to create the tiles in our game. It is also used to transform the terrains while playing the game.

4.2.7.1 Attributes of GameBoard Class

private Terrain[] terrainList : It is used to store all the terrains in the actual game board.

4.2.7.2 Methods of GameBoard Class

private void crateTerrains(): It creates all the terrains in the game and initialize to the terrainList.
private Terrain convertPointToTerrain(int x,int y): When player invokes transform method it comes here and changes the terrain.

4.2.8 Terrain Class

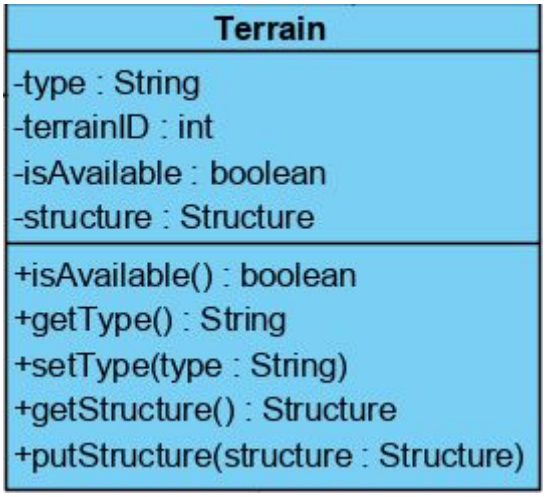

Figure 11: Terrain Class Diagram

This class is essence of the game since buildings done on the gameboard is a concern of the Terrain class.

4.2.8.1 Attributes of Terrain Class

private String type : It is used to represent the type of the terrain.
private int terrainID: Every different type terrain has different terrain ID. It is used to keep track of it.
private boolean isAvailable: It is used to check if the terrain is available.
private Structure structure : It is used to see the Structure which built on the terrain.

4.2.8.2 Methods of Terrain Class

public boolean isAvailable(): It returns the isAvailable variable.
public String getType(): It returns the type variable.
public void setType(String type): It changes the type variable of the class to the desired type.
public Structure getStructure(): It returns the structure variable.
public void putStructure(Structure structure): If the terrain is empty , or the building is being upgraded. This method is called to change the structure or put it first time.

4.2.9 Structure Class

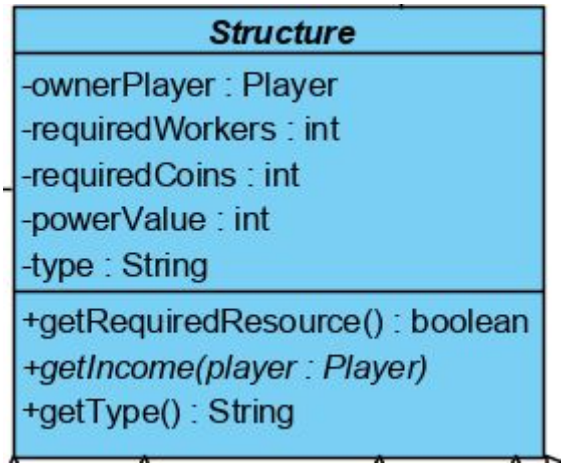

Figure 12: Structure Class Diagram

The structure class represents all the buildings in the game. And they're bounded to the player.

4.2.9.1 Attributes of Structure Class

private Player ownedPlayer: This variable shows who owns the Structure.
private int requiredWorkers : This variable shows the required worker number to be built.
private int requiredCoins: This variable shows the required coin number to be built.
private int powerValue : This variable represents the power value of the structure.
private String type: Since,there are different types of structures it represents which one is selected.

4.2.9.2 Method of Structure Class

public boolean getRequiredResource(): It checks whether player can build the structure or not and gets the resource from the player if player can build.
public void getIncome(Player player): It adds the structures income to players income.
public void getType(): It returns the type of the structure.

4.2.10 GameUtil Class

Figure 13: GameUtil Class Diagram

4.2.10.1 Attributes of GameUtil Class

private static String[] transformationCycle: It is used to keep the conversion cycle of different terrain types.

4.2.10.2 Methods of GameUtil Class

public static int getRequiredSpade(String terType1,String terType2) : Returns required number of spades for tranformation from one terrain type to another one.
public static void updateAdjacency(Player player): It updates the adjacency list of player which is called after building a dwelling or increasing shipping or building bridge.

4.2.11 Dwelling Class

Figure 14: Dwelling Class Diagram

It is the child class of the structure and it's one of the structures in the game.

4.2.11.1 Methods of Dwelling Class

public getIncome(Player player) : It adds income to the players income where every structure have different income number.

4.2.12 Temple Class

Figure 15: Temple Class Diagram

It is the child class of the structure and it's one of the structures in the game.

4.2.12.1 Methods of Temple Class

public getIncome(Player player) : It adds income to the players income where every structure have different income number.

4.2.13 TradingHouse Class

Figure 16: TradingHouse Class Diagram

It is the child class of the structure and it's one of the structures in the game.

4.2.13.1 Methods of TradingHouse Class

public getIncome(Player player) : It adds income to the players income where every structure have different income number.

4.2.14 Sanctuary Class

Figure 17: Sanctuary Class Diagram

It is the child class of the structure and it's one of the structures in the game.

4.2.14.1 Methods of Sanctuary Class

public getIncome(Player player) : It adds income to the players income where every structure have different income number.

4.2.15 Stronghold Class

Figure 18: Stronghold Class Diagram

It is the child class of the structure and it's one of the structures in the game.

4.2.15.1 Methods of Stronghold Class

public getIncome(Player player) : It adds income to the players income where every structure have different income number.

4.2.16 ScoringTile Class

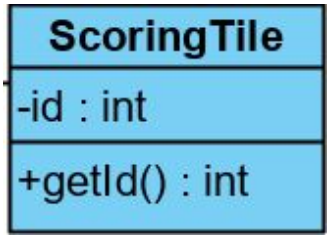

Figure 19: ScoringTile Class Diagram

4.2.16.1 Attributes of ScoringTile Class

private int id : It is used to store the scoring tile's id.

4.2.16.2 Methods of ScoringTile Class

public int getId(): It returns the id number.

4.2.17 ScoringTileList Class

Figure 20: ScoringTileList Class Diagram

It is used to get all the scoringTile classes into an ArrayList.

4.2.17.1 Attributes of ScoringTileList Class

private List<ScoringTile> scoringTiles: It is used to store all the scoring tiles to an array list.

4.2.17.2 Methods of ScroingTileList Class

public ScoringTile getCurrentScoringTile(int round): It returns the scoring tile depending on the round number.

4.2.18 FavorTile Class

Figure 21: FavorTile Class Diagram

It represents the favor tiles in the game.

4.2.19 FavorTileList Class

Figure 22: FavorTileList Class Diagram

It is used to store all the favor tiles in the game.

4.2.19.1 Attributes of FavorTileList Class

private List<FavorTile> favorTiles : It is used to store all the favor tiles into an array list.

4.2.20 TownTile Class

Figure 23: TownTile Class Diagram

4.2.21 TownTileList Class

Figure 24: TownTileList Class Diagram

4.2.21.1 Attributes of TownTileList Class

private List<TownTile> townTiles: It is used to store all the town tiles into an array list.

4.2.22 BonusCard Class

Figure 25: BonusCard Class Diagram

4.2.23 BonusCardList Class

Figure 26: BonusCardList Class Diagram

4.2.23.1 Attributes of BonusCardList Class

private List<BonusCard> bonusCards: It is used to store all the bonus cards into an array list.

4.2.24 Tile Class

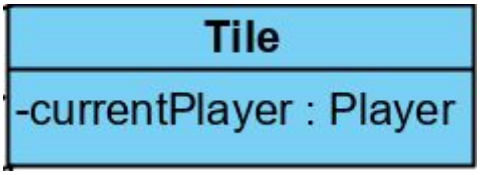

Figure 27: Tile Class Diagram

4.2.24.1. Attributes of Tile Class

private Player currentPlayer: It keeps track of the current player(player whose turn it is) in the game.

4.3. Packages

As packages we decided to use different layers of the program as packages . We decided to use 3 different packages as one UI(User Interface) package, one model package(where the game logic designs are stored) and as the third package we decided to use a server package(where database is stored).

4.4. Class Interfaces

We are planning to have 2 different interfaces, profitable for game logic and draggable for user interface design.

Profitable interface will be implemented by structures, all kinds of tiles and bonus cards since they all give some kind of resource -profit- to the player.

Draggable interface will cover the draggable objects for user interface interaction such as structures and tiles.

5. Glossary & References

1. https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html
2. http://www.testwell.fi/cmtjdesc.html
3. https://ase.in.tum.de/arena02/doc/sdd.pdf
4. https://drive.google.com/file/d/0B9ApNnKlfgcHYlIRbEZEYjFRRzg/view
5. https://www.visual-paradigm.com/VPGallery/diagrams/Package.html

Appendix

For cleaner figure please go to this link.