Bilkent University

Department of Computer Engineering

Object-Oriented Software Engineering

CS 319 Project: Terra Historica

Design Report

Can Aybalık, Çağlar Çankaya, Muhammed Naci Dalkıran, Oğuz Kaan İmamoğlu, Sena Korkut, Tolga Çatalpınar

Instructor: Eray Tüzün

Teaching Assistant(s): Hamdi Alperen Çetin

1. Introduction

1.1. Purpose Of The System

Terra Mystica is a round-based board game where players choose a faction from 14 different factions, trying to have the highest point until the end of the last round. Every faction has its own specific abilities and a corresponding terrain tile.

Players aim to have most victory points among all players, at the end of the game. They can earn victory points in several ways.

Our system converts mystical characters in the original game to some historical figures as an extension. Religions have been adapted to actual ones. Also, GUI elements and graphical designs are remastered.

1.2. Design Goals

Our aim in pinpointing our design goals is to make them as clear and understandable as possible. Non-functional requirements in our analysis report were our main concern during the design process. Also, we have added some other non-functional requirements for the design report.

1.2.1. End-User Criteria

1.2.1.1. Usability

Since Terra Historica is a game and needs to entertain users, we decided to design a simple and understandable system. This is a desktop version of a board game so, it is not a deep and intense game. So, we do not predict that players will have long play-times. For this reason, we do not want players to spend most of their time learning the game. To prevent this; we tried to design quite plain UI, also a help button will be added to the main menu that orients players to an online manual of the game.

Also, players will have the possibility to play Terra Historica with only their mouses. The keyboard is only needed when players want to use some extra features, like typing a nickname for themselves or viewing the in-game menu by using the keyboard shortcuts instead of the mouse.

1.2.1.2. Performance

Performance is an important issue, users must be able to play the game without interruptions and lags. In that stage, Terra Historica does not seem to have an online feature. This means that lags are not our concern for now. Also, there won't be animations in graphics of Terra Historica, since it is desktop version of a board game. This means even low system computers can run Tera Historica. We expect that our system to run between 30 and 60 frame per second.

1.2.1.3. Reliability

Game crashes should not cause data loss. We do not want our players' efforts and times to be wasted. So, they can save the game whenever they want (we are planning that game will be multiplayer but will be played on same computer) and the current game's data will be stored on our database.

1.2.2. Maintenance Criteria

1.2.2.1. Extendibility

The project can be further extended according to users' feedbacks. Thus, the project must be extendable; in order to new updates can be implemented without hardships. We designed a system that new classes can be added easily. For example if a new class is added to view package, all other classes in the view package will not be affected by this situation; no new changes are needed for these classes.

1.2.2.2. Modifiability

Game can need some changes even after it's release. For instance a faction might be so overpowered that can unbalance the meta of game. In a situation like that new changes must

be made in this faction's model class. Our design does not require changes in other classes if one of them is changed.

1.2.2.3. Portability

Terra Historica will be developed with Java and it will run on JVM. So that, playing the game on different operating systems like Windows and MacOs is going to be possible.

1.2.2.4. Reusability

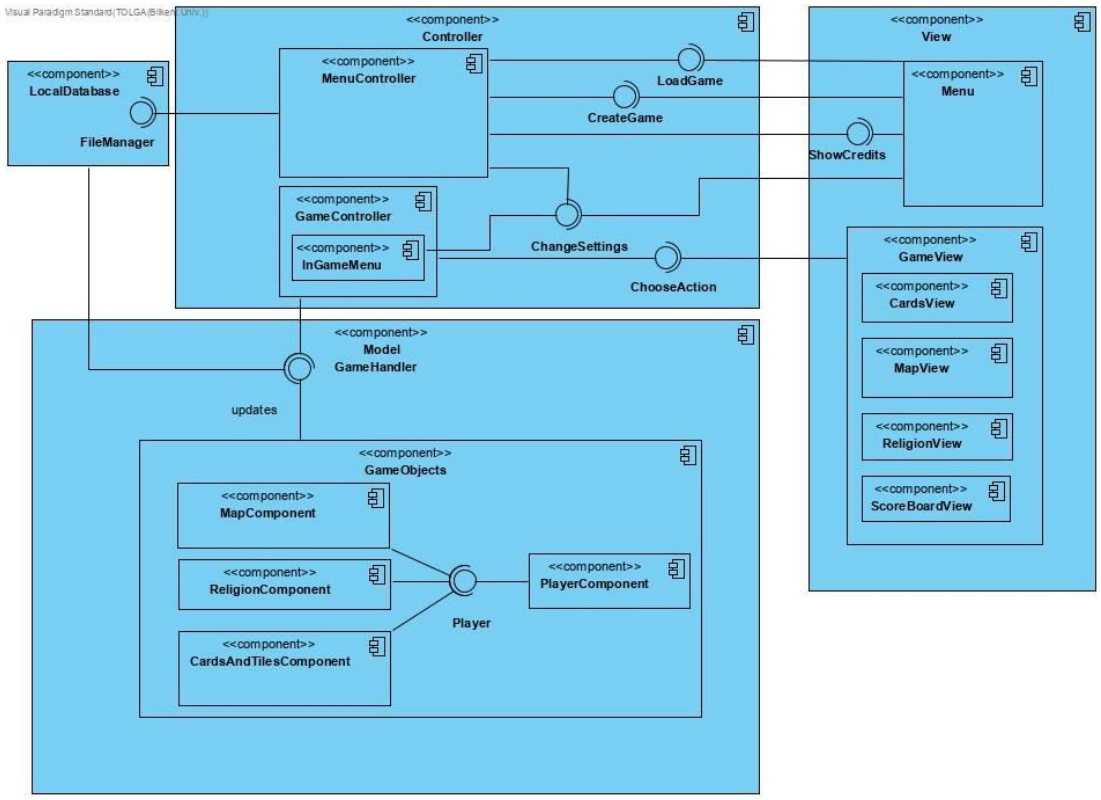
Some classes in Terra Historica can be used in other games without requiring any changes. For instance, main menu class in GUI.

1.2.2.5. Readability

MVC design pattern will be used in this project, related classes and interfaces will be on same package. Also, every single one of the team use comments in their codes and push their commits to VCS with additional comments.

2. High-level Software Architecture

2.1. Subsystem Decomposition



Model 1

In this section, to analyze the design in a detailed and easier manner, system is decomposed into subsystems. It is decided to apply Model, View, Controller (MVC) design pattern to the system. The diagram for subsystem decomposition can be seen from Figure X. Subsystems will be explained in a detailed manner later.

Following the patterns of MVC design, the system divided into three different subsystems:

Model: This subsystem is called Game Entities. It contains components of game such as map, religion, cards and tiles and player.

View: This subsystem is called User Interface. It contains components to provide a n interface for user.

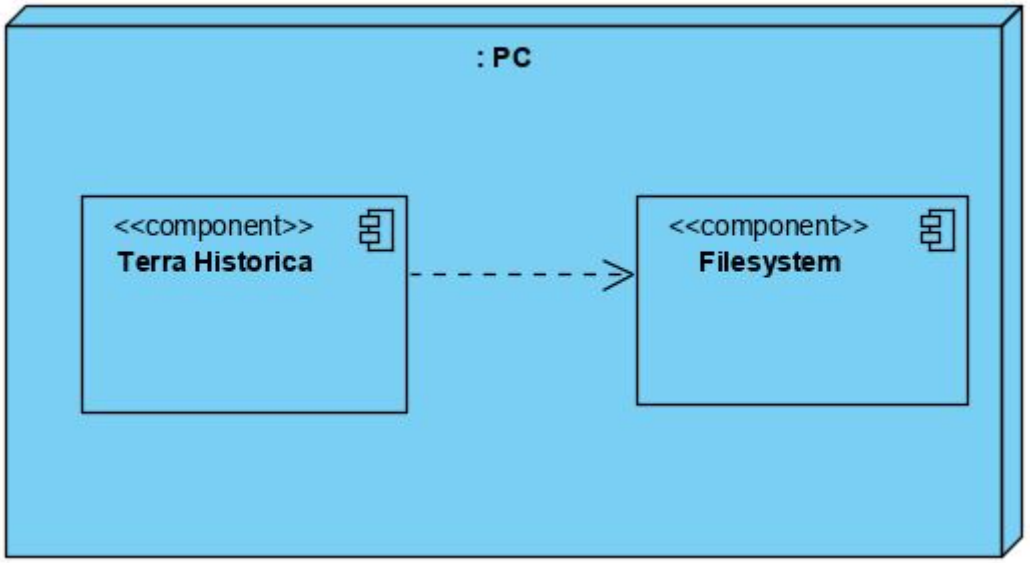
Controller: This subsystem is called Manager. It contains controller components for the game that handles inputs and updates model and view accordingly.

2.2. Hardware/software decomposition

Considering software decomposition of Terra Historica, which is a new version for board game Terra Mystica, it will be written in Java language. The requirement to execute the game is to have minimum Java Development Kit 8 because mainly in the program, JavaFx library will be used.

Considering that our plan does not include online multiplayer option for the game, an online database system for game seems unnecessary. So; at least for current stage, Terra Mystica is going to use local file systems as database and going to store game information into text files. So, when players save the game the current game data will be sent to the files and when they load the game the game data will be pulled from these files.

In the light of previous information, it can be said that Terra Mystica will run on single hardware; this hardware will be PC’s of players.



Model 2

2.3. Persistent data management

As explained in the previous part; filesystem will be used to manage the data. Because, the game will run on single device and it means data are used by multiple users (all players in the game) and single writer (if only one of the players save the game, that is enough for all players).

Since the data does not require to be accessed by multiple platforms and does not require a lot of infrastructure database does not look like ideal option, considering that it may cause some issues like late response time, storage space and high cost. To expand high cost, we have to mention other data management options. Using MySql for AWS RDS was the main option if we decided to manage the data by using online database. Since AWS may cause high bills sometimes, we did not continue with that way.

Another reason for using filesystem is to provide our users to ability that play Terra Historica without needing internet connection.

2.4. Access control and security

In Terra Historica, only actor is player. Player actor can only create game, change game settings, see the credits, load game, exit the game, reach help page, play the game and save the game.

Since Terra Historica is a simple game that does not record it's users' data, security is not a concern for us. Also the system will not use a database to operate, that means system will be safe against SQL Injection attacks.

2.5. Boundary conditions

2.5.1. Initialization of the System

The executable of the game will be a .jar file. Program will load images, FXML files and other files from local filesystem. If they do not exist program will not be initialized. Since there are no database connection required, there will not be any delay related to database connection.

We expect that our system's size will be approximately 100 mb and we do not expect loading time more than 3 seconds for any loading screen.

2.5.2. Termination Of System

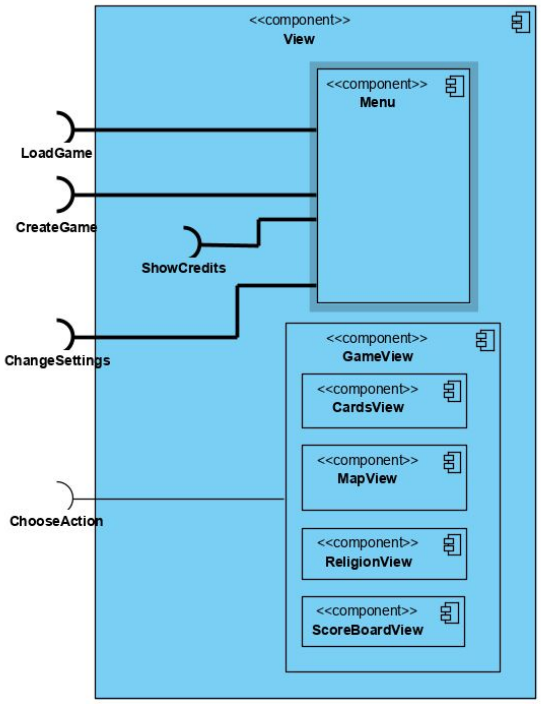
System will be terminated if user chooses to exit in the main menu or in-game menu.

2.5.3. Exception and Failures

In the state of program crash, system will be terminated and game data may be loss unless the game is saved manually by a player.

3. Subsystem Services

3.1. View Subsystem



Model 3

View Subsystem is responsible for providing user interface of Terra Historica. MVC design pattern is used for this subsystem. View Subsystem has 2 main components which are Menu and GameView. Menu component is responsible for representing Main Menu Views such as CreateGame, LoadGame, ShowCredits and ChangeSettings. GameView component represents in-game views such as CardsView, MapView, ReligionView, ScoreBoardView.

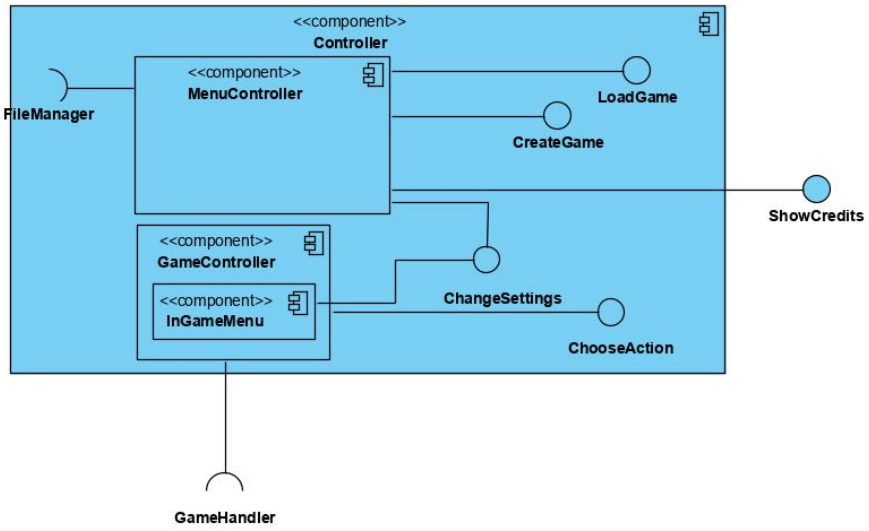
CardsView: This component provides interface of a tab that includes cards and tiles buttons

MapView: This component provides the map interface of the game

ReligionView: This component represents the religion track interface of each players

ScoreBoardView: This component shows the score table interface of each players

3.2. Controller Subsystem



Model 4

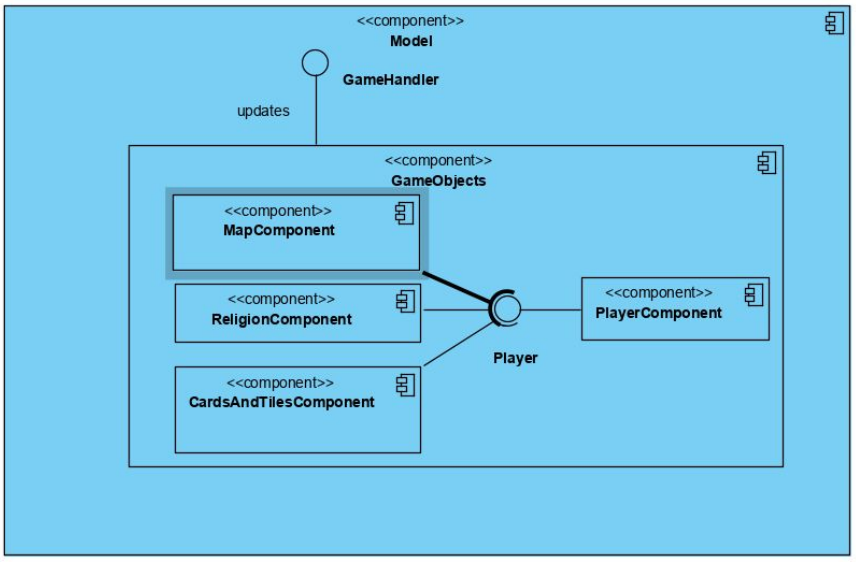
Controller Subsystem provides the data transformation between entity and the boundary objects. There are two main components in controller subsystem such as MenuController and the GameController.

MenuController is responsible for controlling data transition according to selections of user in menu. In order to provide this functionality Menu controller provides three interfaces such as LoadGame, CreateGame, ChangeSettings and ShowCredits. LoadGame interface provides the selection of previously saved games to be selected and sets the game according to selected data by using FileManager interface of LocalDatabase. CreateGame interface handles the setup process of a new game such as gathering data of player count, factions of players etc. Another interface that MenuController component provides is ShowCredits which changes view of main menu to credits scene. Finally, ChangeSettings interface changes view such that user can set the volume or extra settings and game continues in these settings.

GameController is responsible for controlling data transition between users selections in game and model objects. Game controller has InGameMenu. In order to provide this functionality GameController provides two interfaces such as ChooseAction and ChangeSettings. ChooseAction gathers the change data of player made in his/her round and

passes it to model by using GameHandler interface. InGameMenu provides ,as it mentioned before, ChangeSettings interface in order to give chance to change settings during game.

3.3. Model Subsystem



Model 5

As mentioned before, model subsystem represents the model in design pattern MVC. Model subsystem has a main component called GameHandler. This component handles the updates of the game accordingly. GameHandler is a class that is like a controller, however, it does not belong to Controller subsystem of MVC design at it does not take information from view and updates the game accordingly. GameHandler has its methods to be used in controller classes. It only updates game objects that are part of model subsystem.

The model contains a component called Game Objects. Game objects represents all of the sub components of the game such as Map, Religion, Cards, Tiles and Player.

Map Component: This component includes map object with spaces and different structures that can be built on map.

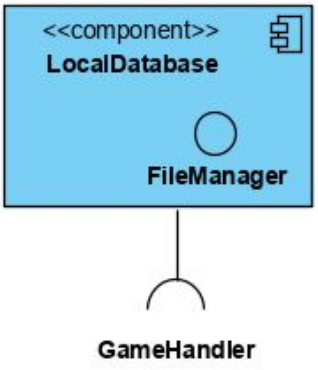
Religion Component: This component includes religion with its four subclasses, Christianity, Islam, Hinduism and Judaism.

Cards and Tiles Component: This component includes both cards and tiles of the game. Cards and tiles can be used for performing different actions and earning new resources for a player during the game. These are bonus cards, favor tiles and town tiles.

Player Component: This component represents each player in the game and keeps the information of each player throughout the game.

Model subsystem gets information from controller or database and GameHandler updates game objects by using relevant methods of corresponding classes.

3.4. Local Database Subsystem



Model 6

Local Database Subsystem saves the data of games into local files or loads data from the files. It operates on the commands that comes from **Game Handler**. **File Manager** has the required methods for saving or loading games.

4. Low-level Design

4.1. Object Design Trade-offs

4.1.1. Performance vs Maintainability

Terra Historica is designed to make the game as efficient as possible. The game is not flexible towards further improvements or updates, since it simulates the board game version of it. Therefore, maintenance is not our first concern. For that reason, we avoided using various controllers in the game. The aim is to decrease data transactions between controllers, which then promotes efficiency of the game.

4.1.2. Functionality vs Usability

We tried to design a simple and usable system as much as possible. However Terra Mystica is a complex board game, even it's manual is 20 pages long. We did not want to remove features of the original game. So much features and options may cause that a decrement in understandability of the game.

4.1.3. Rapid development vs Functionality

We had to this project in less than 13 week. In this restricted time, our priority was not add some extra features like online multiplayer or log-in system for players but to present a whole and working game. To make this real we focused to main features of the game rather than adding some extra features.

4.2. Design Patterns

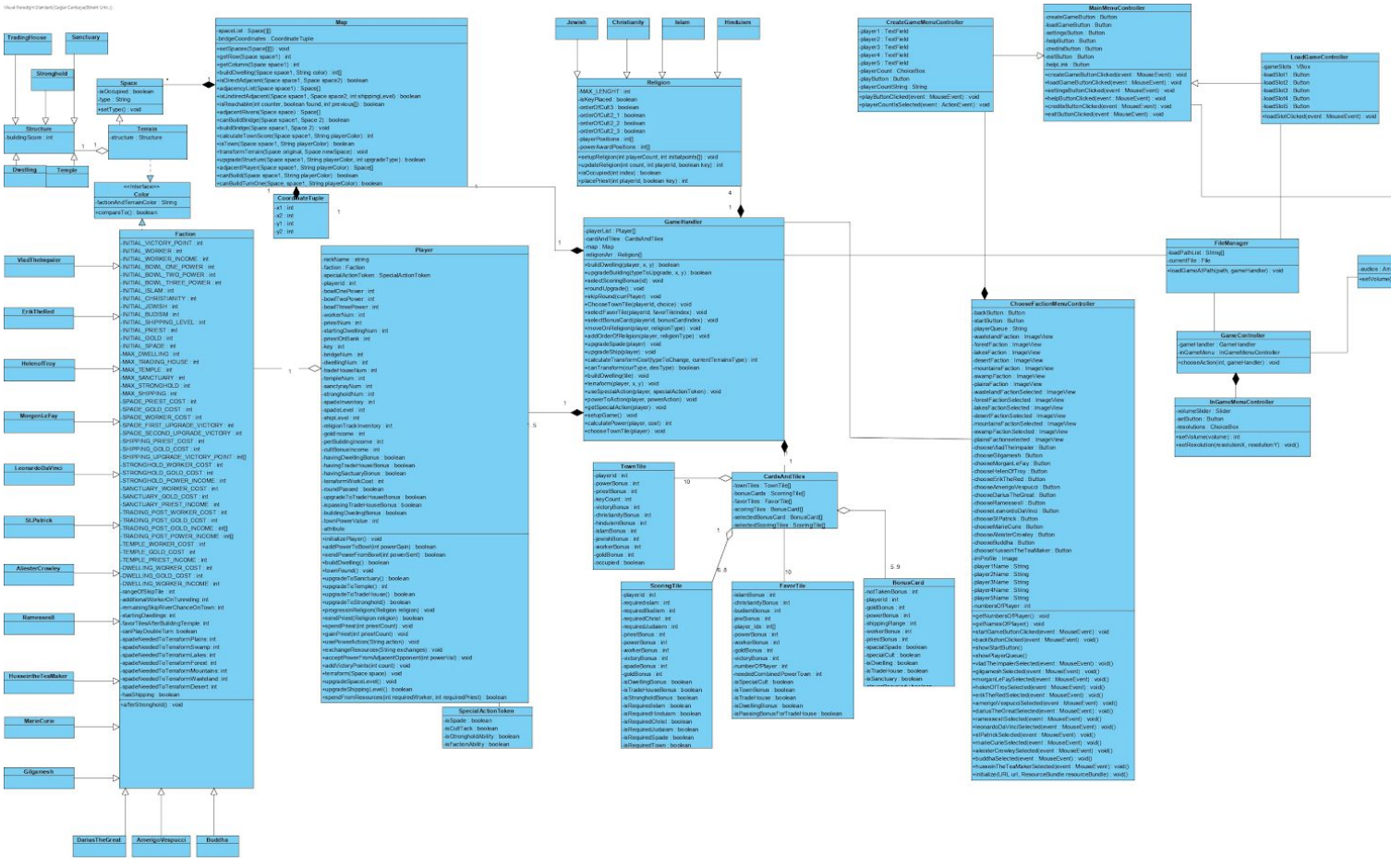
4.2.1. Facade Design Pattern

We used Facade Design Pattern in our code. Because there are so many subsystems in the game that must be unified. For instance, a universal controller had to handle all of the actions that players may perform, as well as updating scores, resources and the map. The subsystems implement the related code in their classes, and then Game Controller calls all of the methods that are predefined.

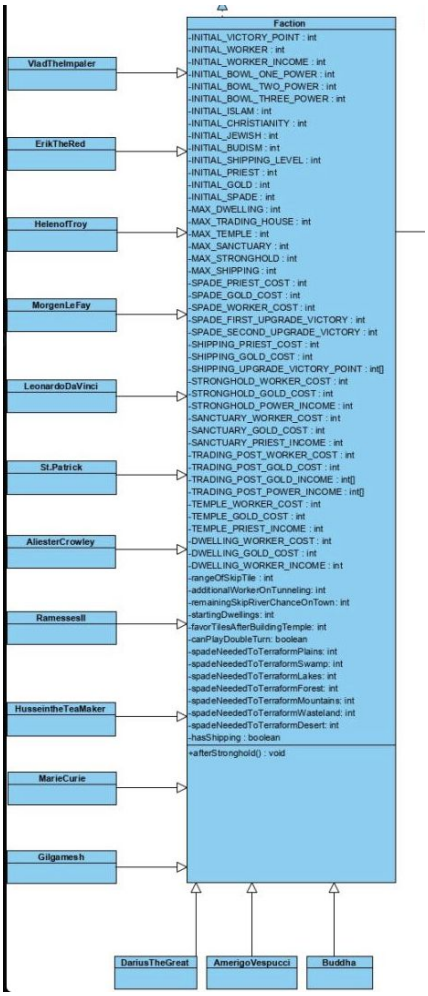
4.2.2. Builder Design Pattern

Terra Historica also implements builder design pattern. In the game, there are various building types which share same attributes and can transform each other. Their methods also are similar. Moreover, each terrain in the map can be transformed to another by the same method. Therefore, our design fits builder design pattern.

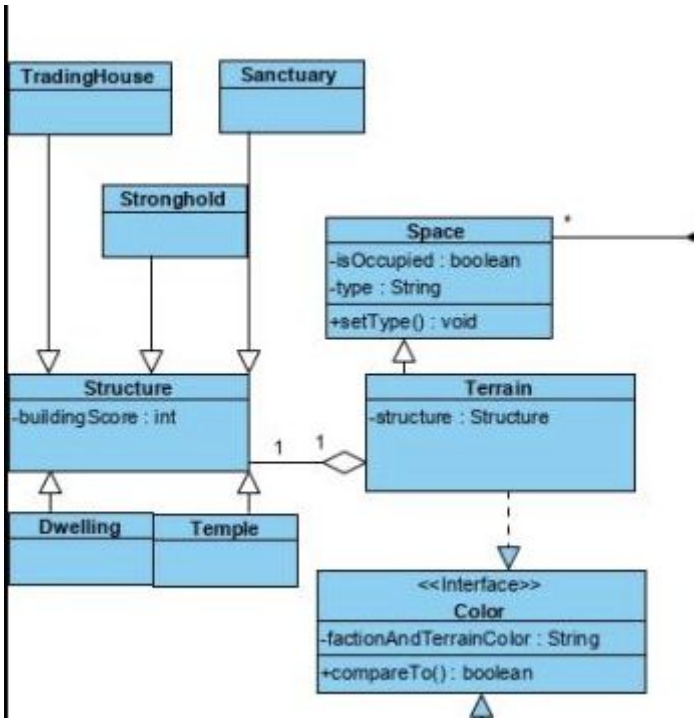
4.3. Final Object Design



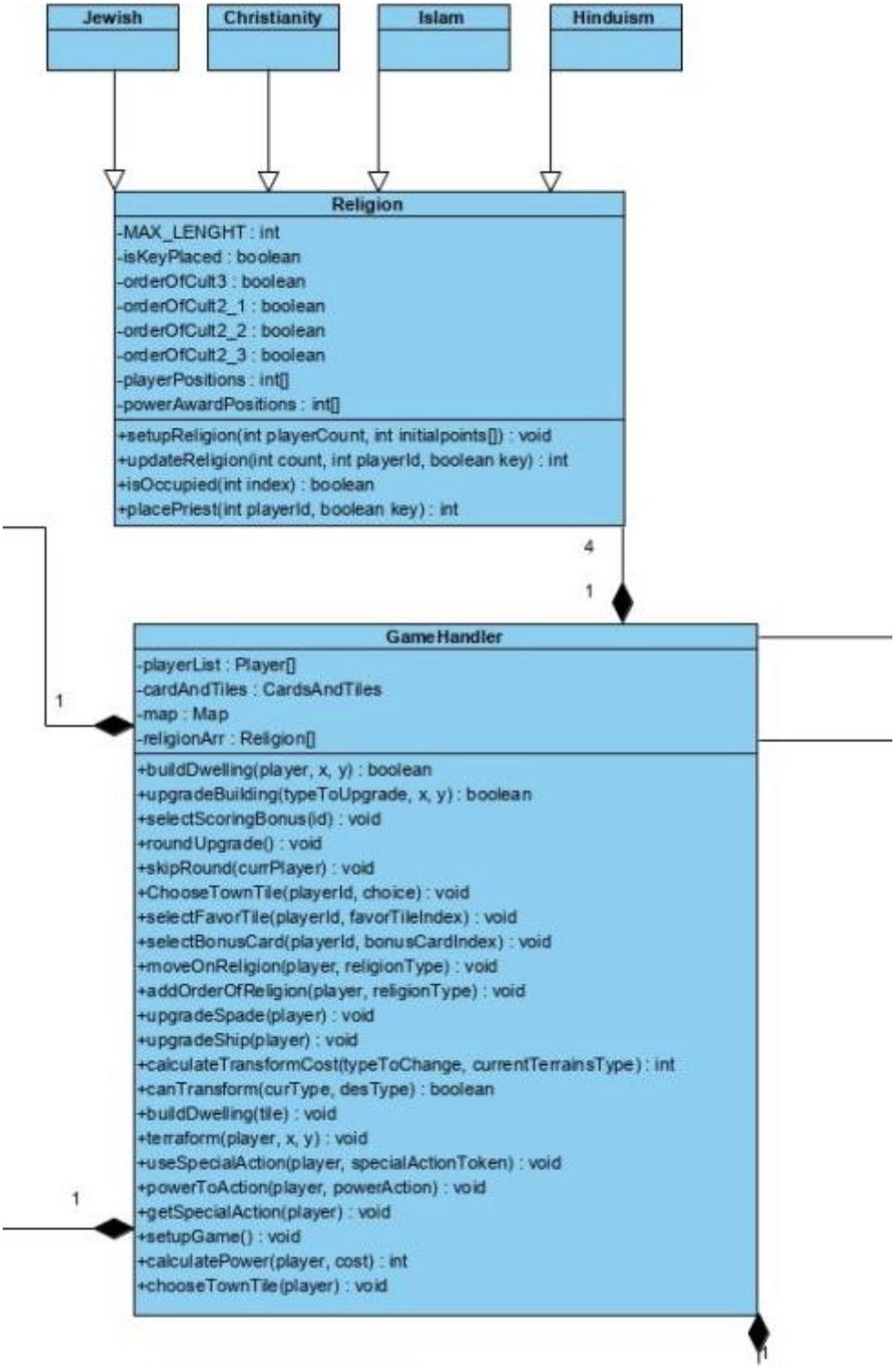
Model 7


Faction
-INITIAL_VICTORY_POINT : int
-INITIAL_WORKER : int
-INITIAL_WORKER_INCOME : int
-INITIAL_BOWL_ONE_POWER : int
-INITIAL_BOWL_TWO_POWER : int
-INITIAL_BOWL_THREE_POWER : int
-INITIAL_ISLAM : int
-INITIAL_CHRISTIANITY : int
-INITIAL_JEWISH : int
-INITIAL_BUDISM : int
-INITIAL_SHIPPING_LEVEL : int
-INITIAL_PRIEST : int
-INITIAL_GOLD : int
-INITIAL_SPADE : int
-MAX_DWELLING : int
-MAX_TRADING_HOUSE : int
-MAX_TEMPLE : int
-MAX_SANCTUARY : int
-MAX_STRONGHOLD : int
-MAX_SHIPPING : int
-SPADE_PRIEST_COST : int
-SPADE_GOLD_COST : int
-SPADE_WORKER_COST : int
-SPADE_FIRST_UPGRADE_VICTORY : int
-SPADE_SECOND_UPGRADE_VICTORY : int
-SHIPPING_PRIEST_COST : int
-SHIPPING_GOLD_COST : int
-SHIPPING_UPGRADE_VICTORY_POINT : int[]
-STRONGHOLD_WORKER_COST : int
-STRONGHOLD_GOLD_COST : int
-STRONGHOLD_POWER_INCOME : int
-SANCTUARY_WORKER_COST : int
-SANCTUARY_GOLD_COST : int
-SANCTUARY_PRIEST_INCOME : int
-TRADING_POST_WORKER_COST : int
-TRADING_POST_GOLD_COST : int
-TRADING_POST_GOLD_INCOME : int[]
-TRADING_POST_POWER_INCOME : int[]
-TEMPLE_WORKER_COST : int
-TEMPLE_GOLD_COST : int
-TEMPLE_PRIEST_INCOME : int
-DWELLING_WORKER_COST : int
-DWELLING_GOLD_COST : int
-DWELLING_WORKER_INCOME : int
-rangeOfSkipTile : int
-additionalWorkerOnTunneling: int
-remainingSkipRiverChanceOnTown: int
-startingDwellings: int
-favorTilesAfterBuildingTemple: int
-canPlayDoubleTurn: boolean
-spadeNeededToTerraformPlains: int
-spadeNeededToTerraformSwamp: int
-spadeNeededToTerraformLakes: int
-spadeNeededToTerraformForest: int
-spadeNeededToTerraformMountains: int
-spadeNeededToTerraformWasteland: int
-spadeNeededToTerraformDesert: int
-hasShipping : boolean
+afterStronghold() : void
VladTheImpaler
ErikTheRed
HelenofTroy
MorgenLeFay
LeonardoDaVinci
St.Patrick
AliesterCrowley
RamesesII
HusseintheTeaMaker
MarieCurie
Gilgamesh
DariusTheGreat
AmerigoVespucci
Buddha

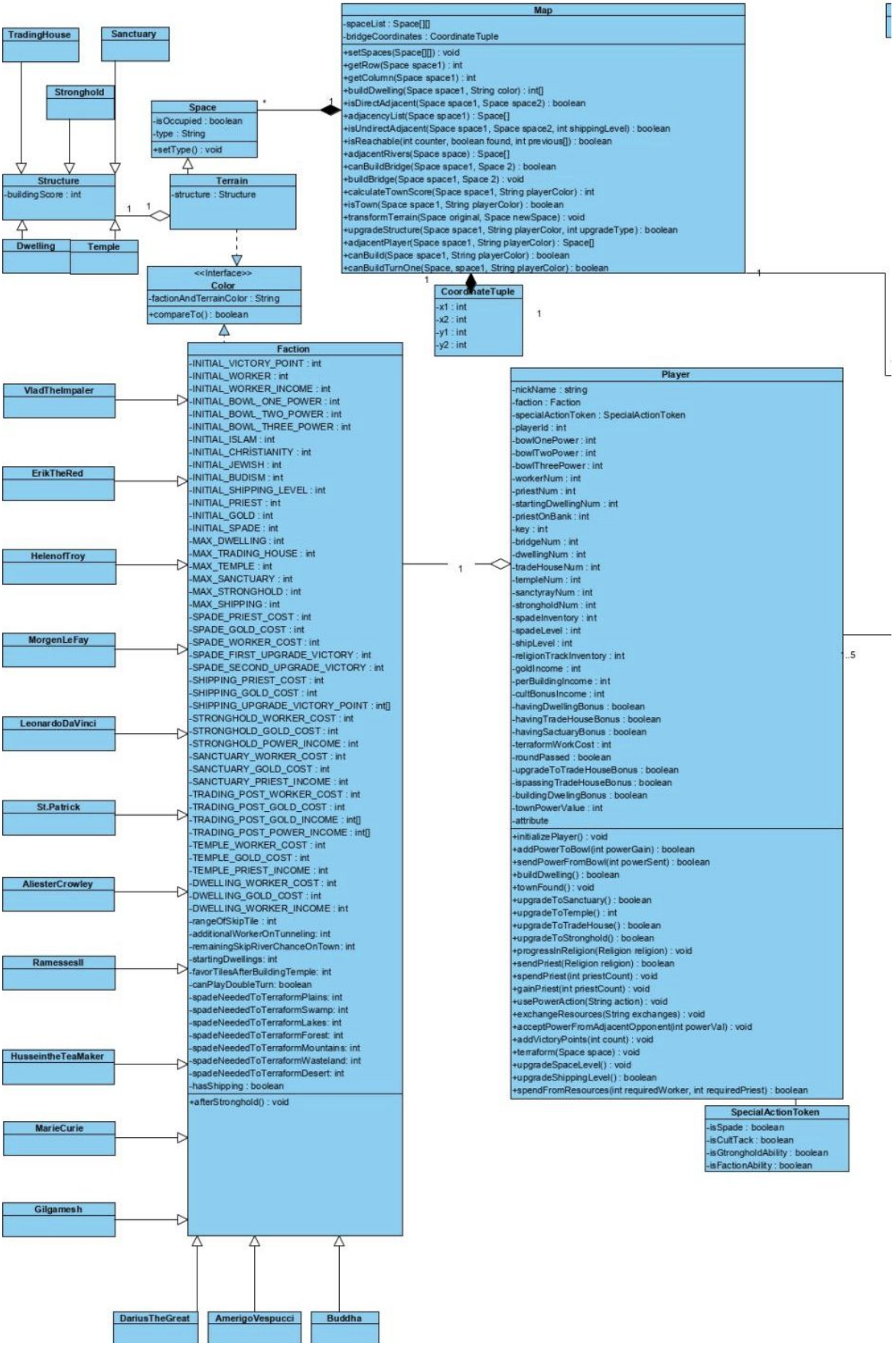
Model 8


TradingHouse
Sanctuary
Stronghold
Space
-isOccupied : boolean
-type : String
+setType() : void
*
Structure
-buildingScore : int
Terrain
-structure : Structure
1
1
Dwelling
Temple
<<Interface>>
Color
-factionAndTerrainColor : String
+compareTo() : boolean

Model 9


Jewish
Christianity
Islam
Hinduism
Religion
-MAX_LENGHT : int
-isKeyPlaced : boolean
-orderOfCult3 : boolean
-orderOfCult2_1 : boolean
-orderOfCult2_2 : boolean
-orderOfCult2_3 : boolean
-playerPositions : int[]
-powerAwardPositions : int[]
+setupReligion(int playerCount, int initialpoints[]) : void
+updateReligion(int count, int playerId, boolean key) : int
+isOccupied(int index) : boolean
+placePriest(int playerId, boolean key) : int
4
1
GameHandler
-playerList : Player[]
-cardAndTiles : CardsAndTiles
-map : Map
-religionArr : Religion[]
1
+buildDwelling(player, x, y) : boolean
+upgradeBuilding(typeToUpgrade, x, y) : boolean
+selectScoringBonus(id) : void
+roundUpgrade() : void
+skipRound(currPlayer) : void
+ChooseTownTile(playerId, choice) : void
+selectFavorTile(playerId, favorTileIndex) : void
+selectBonusCard(playerId, bonusCardIndex) : void
+moveOnReligion(player, religionType) : void
+addOrderOfReligion(player, religionType) : void
+upgradeSpade(player) : void
+upgradeShip(player) : void
+calculateTransformCost(typeToChange, currentTerrainsType) : int
+canTransform(curType, desType) : boolean
+buildDwelling(tile) : void
+terraform(player, x, y) : void
+useSpecialAction(player, specialActionToken) : void
+powerToAction(player, powerAction) : void
1
+getSpecialAction(player) : void
+setupGame() : void
+calculatePower(player, cost) : int
+chooseTownTile(player) : void

Model 10


TradingHouse
Sanctuary
Stronghold
Space
-isOccupied : boolean
-type : String
+setType() : void
Map
-spaceList : Space[][]
-bridgeCoordinates : CoordinateTuple
+setSpaces(Space[][]) : void
+getRow(Space space1) : int
+getColumn(Space space1) : int
+buildDwelling(Space space1, String color) : int[]
+isDirectAdjacent(Space space1, Space space2) : boolean
+adjacencyList(Space space1) : Space[]
+isUndirectAdjacent(Space space1, Space space2, int shippingLevel) : boolean
+isReachable(int counter, boolean found, int previous[]) : boolean
+adjacentRivers(Space space) : Space[]
+canBuildBridge(Space space1, Space 2) : boolean
+buildBridge(Space space1, Space 2) : void
+calculateTownScore(Space space1, String playerColor) : int
+isTown(Space space1, String playerColor) : boolean
+transformTerrain(Space original, Space newSpace) : void
+upgradeStructure(Space space1, String playerColor, int upgradeType) : boolean
+adjacentPlayer(Space space1, String playerColor) : Space[]
+canBuild(Space space1, String playerColor) : boolean
+canBuildTurnOne(Space, space1, String playerColor) : boolean
Structure
-buildingScore : int
Terrain
-structure : Structure
Dwelling
Temple
<<Interface>>
Color
-factionAndTerrainColor : String
+compareTo() : boolean
CoordinateTuple
-x1 : int
-x2 : int
-y1 : int
-y2 : int
Faction
-INITIAL_VICTORY_POINT : int
-INITIAL_WORKER : int
-INITIAL_WORKER_INCOME : int
-INITIAL_BOWL_ONE_POWER : int
-INITIAL_BOWL_TWO_POWER : int
-INITIAL_BOWL_THREE_POWER : int
-INITIAL_ISLAM : int
-INITIAL_CHRISTIANITY : int
-INITIAL_JEWISH : int
-INITIAL_BUDISM : int
-INITIAL_SHIPPING_LEVEL : int
-INITIAL_PRIEST : int
-INITIAL_GOLD : int
-INITIAL_SPADE : int
-MAX_DWELLING : int
-MAX_TRADING_HOUSE : int
-MAX_TEMPLE : int
-MAX_SANCTUARY : int
-MAX_STRONGHOLD : int
-MAX_SHIPPING : int
-SPADE_PRIEST_COST : int
-SPADE_GOLD_COST : int
-SPADE_WORKER_COST : int
-SPADE_FIRST_UPGRADE_VICTORY : int
-SPADE_SECOND_UPGRADE_VICTORY : int
-SHIPPING_PRIEST_COST : int
-SHIPPING_GOLD_COST : int
-SHIPPING_UPGRADE_VICTORY_POINT : int[]
-STRONGHOLD_WORKER_COST : int
-STRONGHOLD_GOLD_COST : int
-STRONGHOLD_POWER_INCOME : int
-SANCTUARY_WORKER_COST : int
-SANCTUARY_GOLD_COST : int
-SANCTUARY_PRIEST_INCOME : int
-TRADING_POST_WORKER_COST : int
-TRADING_POST_GOLD_COST : int
-TRADING_POST_GOLD_INCOME : int[]
-TRADING_POST_POWER_INCOME : int[]
-TEMPLE_WORKER_COST : int
-TEMPLE_GOLD_COST : int
-TEMPLE_PRIEST_INCOME : int
-DWELLING_WORKER_COST : int
-DWELLING_GOLD_COST : int
-DWELLING_WORKER_INCOME : int
-rangeOfSkipTile : int
-additionalWorkerOnTunneling: int
-remainingSkipRiverChanceOnTown: int
-startingDwellings: int
-favorTilesAfterBuildingTemple: int
-canPlayDoubleTurn: boolean
-spadeNeededToTerraformPlains: int
-spadeNeededToTerraformSwamp: int
-spadeNeededToTerraformLakes: int
-spadeNeededToTerraformForest: int
-spadeNeededToTerraformMountains: int
-spadeNeededToTerraformWasteland: int
-spadeNeededToTerraformDesert: int
-hasShipping : boolean
+afterStronghold() : void
Player
-nickName : string
-faction : Faction
-specialActionToken : SpecialActionToken
-playerId : int
-bowlOnePower : int
-bowlTwoPower : int
-bowlThreePower : int
-workerNum : int
-priestNum : int
-startingDwellingNum : int
-priestOnBank : int
-key : int
-bridgeNum : int
-dwellingNum : int
-tradeHouseNum : int
-templeNum : int
-sanctyrayNum : int
-strongholdNum : int
-spadeInventory : int
-spadeLevel : int
-shipLevel : int
-religionTrackInventory : int
-goldIncome : int
-perBuildingIncome : int
-cultBonusIncome : int
-havingDwellingBonus : boolean
-havingTradeHouseBonus : boolean
-havingSactuaryBonus : boolean
-terraformWorkCost : int
-roundPassed : boolean
-upgradeToTradeHouseBonus : boolean
-ispassingTradeHouseBonus : boolean
-buildingDwelingBonus : boolean
-townPowerValue : int
-attribute
+initializePlayer() : void
+addPowerToBowl(int powerGain) : boolean
+sendPowerFromBowl(int powerSent) : boolean
+buildDwelling() : boolean
+townFound() : void
+upgradeToSanctuary() : boolean
+upgradeToTemple() : int
+upgradeToTradeHouse() : boolean
+upgradeToStronghold() : boolean
+progressInReligion(Religion religion) : void
+sendPriest(Religion religion) : boolean
+spendPriest(int priestCount) : void
+gainPriest(int priestCount) : void
+usePowerAction(String action) : void
+exchangeResources(String exchanges) : void
+acceptPowerFromAdjacentOpponent(int powerVal) : void
+addVictoryPoints(int count) : void
+terraform(Space space) : void
+upgradeSpaceLevel() : void
+upgradeShippingLevel() : boolean
+spendFromResources(int requiredWorker, int requiredPriest) : boolean
SpecialActionToken
-isSpade : boolean
-isCultTack : boolean
-isGtrongholdAbility : boolean
-isFactionAbility : boolean
ViadTheImpaler
ErikTheRed
HelenofTroy
MorgenLeFay
LeonardoDaVinci
St.Patrick
AliesterCrowley
RamessesII
HusseintheTeaMaker
MarieCurie
Gilgamesh
DariusTheGreat
AmerigoVespucci
Buddha

Model 11


CreateGameMenuController
player1 : TextField
player2 : TextField
player3 : TextField
player4 : TextField
player5 : TextField
playerCount : ChoiceBox
playButton : Button
playerCountString : String
+playButtonClicked(event : MouseEvent) : void
+playerCountIsSelected(event : ActionEvent) : void
MainMenuController
-createGameButton : Button
-loadGameButton : Button
-settingsButton : Button
-helpButton : Button
-creditsButton : Button
-exitButton : Button
-helpLink : Button
+createGameButtonClicked(event : MouseEvent) : void
+loadGameButtonClicked(event : MouseEvent) : void
+settingsButtonClicked(event : MouseEvent) : void
+helpButtonClicked(event : MouseEvent) : void
+creditsButtonClicked(event : MouseEvent) : void
+exitButtonClicked(event : MouseEvent) : void
LoadGameController
-gameSlots : VBox
-loadSlot1 : Button
-loadSlot2 : Button
-loadSlot3 : Button
-loadSlot4 : Button
-loadSlot5 : Button
+loadSlotClicked(event : MouseEvent) : void
FileManager
-loadPathList : String[]
-currentFile : File
+loadGameAtPath(path, gameHandler) : void
SoundController
-audios : ArrayList<Media>
+setVolume(int)
ChooseFactionMenuController
-backButton : Button
-startButton : Button
-playerQueue : String
-wastelandFaction : ImageView
-forestFaction : ImageView
-lakesFaction : ImageView
-desertFaction : ImageView
-mountainsFaction : ImageView
-swampFaction : ImageView
-plainsFaction : ImageView
-wastelandFactionSelected : ImageView
-forestFactionSelected : ImageView
-lakesFactionSelected : ImageView
-desertFactionSelected : ImageView
-mountainsFactionSelected : ImageView
-swampFactionSelected : ImageView
-plainsFactionselected : ImageView
-chooseVladTheImpaler : Button
-chooseGilgamesh : Button
-chooseMorganLeFay : Button
-chooseHelenOfTroy : Button
-chooseErikTheRed : Button
-chooseAmerigoVespucci : Button
-chooseDariusTheGreat : Button
-chooseRamsesseII : Button
-chooseLeanordoDaVinci : Button
-chooseStPatrick : Button
-chooseMarieCurie : Button
-chooseAleisterCrowley : Button
-chooseBuddha : Button
-chooseHusseinTheTeaMaker : Button
-imProfile : Image
-player1Name : String
-player2Name : String
-player3Name : String
-player4Name : String
-player5Name : String
-numbersOfPlayer : int
+getNumbersOfPlayer() : void
+getNamesOfPlayer() : void
+startGameButtonClicked(event : MouseEvent) : void
+backButtonClicked(event : MouseEvent) : void()
+showStartButton()
+showPlayerQueue()
+vladTheImpalerSelected(event : MouseEvent) : void()
+gilgameshSelected(event : MouseEvent) : void()
+morganLeFaySelected(event : MouseEvent) : void()
+helenOfTroySelected(event : MouseEvent) : void()
+erikTheRedSelected(event : MouseEvent) : void()
+amerigoVespucciSelected(event : MouseEvent) : void()
+dariusTheGreatSelected(event : MouseEvent) : void()
+ramsesseIISelected(event : MouseEvent) : void()
+leonardoDaVinciSelected(event : MouseEvent) : void()
+stPatrickSelected(event : MouseEvent) : void()
+marieCurieSelected(event : MouseEvent) : void()
+aleisterCrowleySelected(event : MouseEvent) : void()
+buddhaSelected(event : MouseEvent) : void()
+husseinTheTeaMakerSelected(event : MouseEvent) : void()
+initialize(URL url, ResourceBundle resourceBundle) : void()
GameController
-gameHandler : GameHandler
-inGameMenu : InGameMenuController
+chooseAction(int, gameHandler) : void
InGameMenuController
-volumeSlider : Slider
-setButton : Button
-resolutions : ChoiceBox
+setVolume(volume) : int
+setResolution(resolutionX, resolutionY) : void()
5..9
BonusCard
-notTakenBonus : int
-playerId : int
-goldBonus : int
-powerBonus : int
-shippingRange : int
-workerBonus : int
-priestBonus : int
-spacialSpade : boolean
-specialCult : boolean
-isDwelling : boolean
-isTradeHouse : boolean
-isSanctuary : boolean

Model 12

4.4. Packages

4.4.1. javafx.scene.*

This package is used for creating scenes and transferring between scenes

4.4.2. javafx.stage.*

This package is used for initializing a stage at the beginning of program execution

4.4.3. javafx.FXML.*

This package is used for editing scenes via “Scene Builder”

4.4.4. javafx.event.*

This package is used for initializing MouseEvent and ActionEvent into the scene objects

4.4.5. javafx.Application.*

This package is used for defining user interface container by means of a stage and a scene

4.4.6. java.util.*

This package is used for features such as ArrayList and ResourceBundle

4.5. Class Interfaces

4.5.1. MainMenuController Class

Attributes:

private Button createGameButton: Button instance of create game button

private Button loadGameButton: Button instance of load game button

private Button settingsButton: Button instance of settings button

private Button helpButton: Button instance of help button

private Button creditsButton: Button instance of credits button

private Button exitsButton: Button instance of exit button

private Button helpLink: Button instance of online manual link

Methods:

public void createGameButtonClicked(MouseEvent event): Listener method that changes scene to the create game menu view

public void loadGameButtonClicked(MouseEvent event): Listener method that changes scene to the load game menu view

public void settingsButtonClicked(MouseEvent event): Listener method that changes scene to the settings menu view

public void helpButtonClicked(MouseEvent event): Listener method that changes scene to the help menu view

public void creditsButtonClicked(MouseEvent event): Listener method that changes scene to the credits menu view

public void exitButtonClicked(MouseEvent event): Listener method that end s all the scenes and exits the game

4.5.2. CreateGameMenuController class

Attributes:

private TextField player1: TextField instance of the first player's name

private TextField player2: TextField instance of the second player's name

private TextField player3: TextField instance of the third player's name

private TextField player4: TextField instance of the forth player's name

private TextField player5: TextField instance of the fifth player's name

private ChoiceBox<String> playerCount: ChoiceBox instance of player counts

private Button playButton: Button instance that starts the game

private String playerCountString: String instance of player counts

Methods:

public void playButtonClicked(MouseEvent event): Listener method that changes scene to the game menu view

public void playerCountIsSelected(ActionEvent event): Listener method that enables play button if the player count is seelected

4.5.3. ChooseFactionMenuController class

Attributes:

private Button backButton: Button instance of back button

private Button startButton: Button instance of start button

private String playerQueue: String instance that hold player queue

private ImageView wastelandFaction: ImageView instance of wasteland faction

private ImageView forestFaction: ImageView instance of forest faction

private ImageView lakesFaction: ImageView instance of lakes faction

private ImageView desertFaction: ImageView instance of desert faction

private ImageView mountainsFaction: ImageView instance of mountains faction

private ImageView swampFaction: ImageView instance of swamp faction

private ImageView plainsFaction: ImageView instance of plains faction

private ImageView wastelandFactionSelected: ImageView instance of when user selected the wasteland faction

private ImageView forestFactionSelected: ImageView instance of when user selected the forest faction

private ImageView lakesFactionSelected: ImageView instance of when user selected the lakesfaction

private ImageView desertFactionSelected: ImageView instance of when user selected the desert faction

private ImageView mountainsFactionSelected: ImageView instance of when user selected the mountains faction

private ImageView swampFactionSelected: ImageView instance of when user selected the swamp faction

private ImageView plainsFactionSelected: ImageView instance of when user selected the plains faction

private Button chooseGilgamesh: Button instance that indicates user selected the Gilgamesh

private Button chooseVladTheImpaler: Button instance that indicates user selected the Vlad The Impaler

private Button chooseMorganLeFay: Button instance that indicates user selected the Morgan Le Fay

private Button chooseHelenOfTroy: Button instance that indicates user selected the Helen of Troy

private Button chooseErikTheRed: Button instance that indicates user selected the Erik The Red

private Button chooseAmerigoVespucci: Button instance that indicates user selected the Amerigo Vespucci

private Button chooseDariusTheGreat: Button instance that indicates user selected the Darius The Great

private Button chooseRamessesII: Button instance that indicates user selected the Ramesses II

private Button chooseLeonardoDaVinci: Button instance that indicates user selected the Leonardo Da Vinci

private Button chooseStPatrick: Button instance that indicates user selected the St. Patrick

private Button chooseMarieCurie: Button instance that indicates user selected the Marie Curie

private Button chooseAleisterCrowley: Button instance that indicates user selected the Aleister Crowley

private Button chooseBuddha: Button instance that indicates user selected the Buddha

private Button chooseHusseinTheTeaMaker: Button instance that indicates user selected the Hussein The Teamaker

private Image imProfile: Image instance that shows the faction cards of characters

private String player1Name: String instance that shows the first player's name

private String player2Name: String instance that shows the second player's name

private String player3Name: String instance that shows the third player's name

private String player4Name: String instance that shows the forth player's name

private String player5Name: String instance that shows the fifth player's name

private int numberOfPlayers: Integer instance that holds number of players

Methods:

public void getNumbersOfPlayer(String str): gets number of players as string, then convert it to integer

public void getNamesOfPlayers(String player1, String player2, String player3, String player4, String player5): gets name of players as string

public void startGameButtonClicked(MouseEvent event): Listener methods that changes scene to the game view

public void backButtonClicked(MouseEvent event): Listener methods that changes scene to create game view

public void showStartButton(): shows start button if all the players selected their faction

public void showPlayerQueue(): shows which player selecting the faction

public void gilgameshSelected(MouseEvent event): Listener method that assign player faction to the Gilgamesh

public void vladTheImpalerSelected(MouseEvent event): Listener method that assign player faction to the Vlad The Impaler

public void morganLeFaySelected(MouseEvent event): Listener method that assign player faction to the Morgan Le Fay

public void helenOfTroySelected(MouseEvent event): Listener method that assign player faction to the Helen Of Troy

public void erikTheRedSelected(MouseEvent event): Listener method that assign player faction to the Erik The Red

public void amerigoVespucciSelected(MouseEvent event): Listener method that assign player faction to the Amerigo Vespucci

public void dariusTheGreatSelected(MouseEvent event): Listener method that assign player faction to the Darius The Great

public void ramessesIISelected(MouseEvent event): Listener method that assign player faction to the Ramesses II

public void leonardoDaVinciSelected(MouseEvent event): Listener method that assign player faction to the Leonardo Da Vinci

public void stPatrickSelected(MouseEvent event): Listener method that assign player faction to the St. Patrick

public void marieCurieSelected(MouseEvent event): Listener method that assign player faction to the Marie Curie

public void aleisterCrowleySelected(MouseEvent event): Listener method that assign player faction to the Aleister Crowley

public void buddhaSelected(MouseEvent event): Listener method that assign player faction to the Buddha

public void husseinTheTeaMakerSelected(MouseEvent event): Listener method that assign player faction to the Hussein The Teamaker

public void Initialize(URL url, ResourceBundle resourceBundle): Initializes choose faction menu images and imageviews

4.5.4. Map Class

Attributes:

public Space [][] spaces: This attribute represents the map. All terrains are stored in this 2d array.

Methods:

public Map(): This is the constructor method. Creates the 2d array and fills it with terrains.

public void setSpaces(Space[][] spaces): This initializes spaces array.

public int getRow(Space space1): This method returns the row index of specified terrain.

public int getColumn(Space space1): This method returns the column index of specified terrain.

public int[] buildDwelling(Space space1, String color): This method constructs a dwelling in the specified terrain for specified player.

public boolean isDirectAdjacent(Space space1, Space space2): This method enables us to learn whether the two terrains are adjacent or not.

public Space[] adjacencyList(Space space1): This method returns all of adjacents of a specified terrain.

public boolean isUndirectAdjacent(Space space1, Space space2, int shippingLevel): This method enables us to learn whether two terrains are indirectly adjacent, which means only one space of river separates them.

public boolean isReachable(Space space1, Space space2, int shippingLevel, int counter,boolean found, ArrayList<Space> previous): This method says that can we reach to a specified terrain from another specified terrain with river transportation.

public ArrayList<Space> adjacentRivers(Space space1): This method returns all adjacent rivers of a specified terrain.

public boolean canBuildBridge(Space space1, Space space2): This method controls can we build bridge between two terrains.

public void buildBridge(Space space1, Space space2): This method builds bridges between two terrain.

public int calculateTownScore(Space space1, String playerColor): This method calculates town score of a specified player when this player build a new building. This method helps the following one.

public boolean isTown(Space space1, String playerColor): This method determines if the player founds a town.

public void transformTerrain(Space original, Space newSpace): This method transforms a terrain to another type of terrain.

public boolean upgradeStructure(Space space1, String playerColor, int upgradeType): This method upgrades the specified building of a specified player.

public ArrayList<Space> adjacentPlayer(Space space1, String playerColor): This method returns the list of all adjacent players of the specified player.

public boolean canBuild(Space space1, String playerColor): This method controls whether the specified player can build on the specified terrain.

public boolean canBuildTurnOne(Space space1, String playerColor): This method controls whether the specified player can build on specified terrain in turn one.

4.5.5. Player Class

Attributes:

String nickName: This attribute represents the name the player enters at the beginning.

Faction faction: This attribute represents the chosen faction by player.

SpecialActionToken specialActionToken: This attribute represents special action token.

private int playerId: This attribute represents integer number for player.

private int bowlOnePower: This attribute represents the number of powers in bowl one.

private int bowlTwoPower: This attribute represents the number of powers in bowl two.

private int bowlThreePower: This attribute represents the number of powers in bowl three.

private int workerNum: This attribute represents the number of workers player has.

private int priestNum: This attribute represents the number of priests player has.

private int goldNum: This attribute represents the number of gold player has.

private int victoryPointNum: This attribute represents the number of victory points player has.

int townScore : This attribute represents the progress of player for founding a town.

private int powerIncome: This attribute represents the number of power income for next round that player has.

private int workerIncome: This attribute represents the number of worker income for next round that player has.

private int priestIncome: This attribute represents the number of priest income for next round that player has.

private int startingDwellingNum: This attribute represents the number of dwellings player can build at the beginning of the game.

private int priestOnBank: This attribute represents the number of priests player is able to use.

private int bridgeNum: This attribute represents the number of bridges player currently have.

private int dwellingNum: This attribute represents the number of dwellings player currently have.

private int tradingPostNum: This attribute represents the number of trading posts player currently have.

private boolean key: This attribute represents if player has a key.

private int templeNum: This attribute represents the number of temples player currently have.

private int sanctuaryNum: This attribute represents the number of sanctuaries player currently have.

private int strongholdNum: This attribute represents the number of strongholds player currently have.

private int spadeInventory: This attribute represents the number of spades inventory.

private int spadeLevel: This attribute represents the level of spade player currently have.

private int shipLevel: This attribute represents the level of shipping player currently have.

private int religionTrackInventory: This attribute represents the number of religion track inventory.

private int goldIncome: This attribute represents the number of gold income for next round that player has.

private int perBuildingIncome: This attribute represents the income value per building.

private int cultBonusIncome: This attribute represents the bonus income for religion.

private boolean havingDwellingBonus: This attribute represents if player has a dwelling bonus.

private boolean havingTradingPostBonus: This attribute represents if player has a trading post bonus.

private boolean havingSanctuaryBonus: This attribute represents if player has a sanctuary bonus .

private int terraformWorkerCost: This attribute represents the cost of workers to terraform a place.

private boolean roundPassed: This attribute represents if player has passed the round.

private boolean upgradeToTradingPostBonus: This attribute represents if player has a bonus for upgrading to trading post.

private boolean isPassingTradingPostBonus: This attribute represents if player has a bonus for passing trading post.

private boolean buildingDwellingBonus: This attribute represents if player has a bonus for building a dwelling.

private int townPowerValue: This attribute represents the power value to build a town.

Methods:

public void setFaction(Faction faction): This method is to set a faction to player.

public Faction getFaction(): This method is to get the faction of player.

public void initializePlayer(): This method is to initialize properties of player at the beginning of the game.

public boolean addPowerToBowl(int powerGain): This method is to add power to bowl according to the rules.

public boolean spendPowerFromBowl(int powerSpent): This method is to spend power from bowl according to the rules.

public boolean sacrificePower(): This method is to sacrifice a power for the rest of the game.

public boolean buildDwelling(): This method is to build a dwelling and update properties of player.

public void townFound(): This method updates properties of player after a town found.

public boolean upgradeToSanctuary(): This method is to upgrade a dwelling to sanctuary and update properties of player.

public int upgradeToTemple(): This method is to upgrade a trading post and update properties of player.

public boolean upgradeToTradingPost(boolean isThereAdjacentOpponent): This method is to upgrade a dwellingto trading post and update properties of player.

public boolean upgradeToStronghold(): This method is to upgrade a trading post to stronghold and update properties of player.

public void progressInReligion(Religion religion): This method is to make a progress for player in religion.

public boolean sendPriest(Religion religion): This method is to send a priest.

public void spendPriest(int priestCount): This method is to spend priest for an action.

public void gainPriest(int priestCount): This method is to gain action.

public void usePowerAction(String action): This method is to use a power action and update properties accordingly.

public void exchangeResources(String exchanges): This method is to exchange resources and update properties accordingly.

public void acceptPowerFromAdjacentOpponent(int powerVal): This method is for accepting gaining power from adjacency and update properties accordingly.

public void addVictoryPoints(int count): This method is to increase victory poin number.

public void terraform(Space space): This method is to terraform a landscape.

public void upgradeSpadeLevel(): This method is to upgrade spade level.

public boolean upgradeShippingLevel(): This method is to upgrade shipping level.

public boolean spendFromResources(int requiredWorker , int requiredGold , int requiredPriest): This method is to spend resources for various actions and update properties accordingly.

public void passRound(): This method is to pass a round.

public void useBonusFromFavorTile(): This method is to use the bonus of favor tile chosen and update properties accordingly.

public void returnBonusCard(): This method is to return the bonus card at the end of the round.

public void useFavorTile(): This method is to use the favor tile and update properties accordingly.

4.5.6. Faction Class

Faction has 14 different subclasses for each historical figure. Each faction has its own value for properties explained below.

Attributes:

public final int MAX_DWELLING: This attribute represents maximum number of a dwelling that can be built.

public final int MAX_TRADING_POST: This attribute represents maximum number of a trading post that can be built.

public final int MAX_TEMPLE: This attribute represents maximum number of a temple that can be built.

public final int MAX_SANCTUARY: This attribute represents maximum number of a sanctuary that can be built.

public final int MAX_STRONGHOLD: This attribute represents maximum number of a stronghold that can be built.

public int MAX_SHIPPING: This attribute represents maximum number of shipping level that can be reached.

public int INITIAL_VICTORY_POINT: This attribute represents the starting victory point for the faction.

public int INITIAL_WORKER: This attribute represents the starting worker number for the faction.

public int INITIAL_WORKER_INCOME: This attribute represents the starting worker income for the faction.

public int INITIAL_BOWL_ONE_POWER: This attribute represents the starting power value for bowl one for the faction.

public int INITIAL_BOWL_TWO_POWER: This attribute represents the starting power value for bowl two for the faction.

public int INITIAL_BOWL_THREE_POWER: This attribute represents the starting power value for bowl three for the faction.

public int INITIAL_ISLAM: This attribute represents the starting islam level for the faction.

public int INITIAL_CHRISTIANITY: This attribute represents the starting christianity level for the faction.

public int INITIAL_JUDAISM: This attribute represents the starting judaism level for the faction.

public int INITIAL_HINDUISM: This attribute represents the starting hinduism level for the faction.

public int INITIAL_SHIPPING_LEVEL: This attribute represents the starting shipping level for the faction.**;**

public int INITIAL_PRIEST: This attribute represents the starting priest number for the faction.

public int INITIAL_GOLD: This attribute represents the starting gold number for the faction.

public int INITIAL_SPADE: This attribute represents the starting spade number for the faction.

public int SPADE_PRIEST_COST: This attribute represents the cost of priest for spade according to the faction.

public int SPADE_GOLD_COST: This attribute represents the cost of gold for spade according to the faction.

public int SPADE_WORKER_COST: This attribute represents the cost of worker for spade according to the faction.

public int SPADE_FIRST_UPGRADE_VICTORY: This attribute represents the gained victory point for first spade upgrading.

public int SPADE_SECOND_UPGRADE_VICTORY: This attribute represents the gained victory point for second spade upgrading.

public int SHIPPING_PRIEST_COST: This attribute represents the cost of priest for shipping according to the faction.

public int SHIPPING_GOLD_COST: This attribute represents the cost of gold for shipping according to the faction.

public int SHIPPING_UPGRADE_VICTORY_POINTS[] = {2,3,4,0};

public int STRONGHOLD_WORKER_COST: This attribute represents the cost of worker for stronghold according to the faction.

public int STRONGHOLD_GOLD_COST: This attribute represents the cost of priest for spade according to the faction.

public int STRONGHOLD_POWER_INCOME: This attribute represents the income of power for stronghold according to the faction.

public int STRONGHOLD_PRIEST_INCOME: This attribute represents the income of priest for stronghold according to the faction.

public int SANCTUARY_WORKER_COST: This attribute represents the cost of worker for sanctuary according to the faction.

public int SANCTUARY_GOLD_COST: This attribute represents the cost of gold for sanctuary according to the faction.

public int SANCTUARY_PRIEST_INCOME: This attribute represents the income of priest for sanctuary according to the faction.

public int TRADING_POST_WORKER_COST: This attribute represents the cost of worker for trading post according to the faction.

public int TRADING_POST_GOLD_COST: This attribute represents the cost of gold for trading post according to the faction.

public int tradingPostGoldIncome[]: This attribute represents the income of gold for trading post according to the faction.

public int tradingPostPowerIncome[]: This attribute represents the income of power for trading post according to the faction.

public int TEMPLE_WORKER_COST: This attribute represents the cost of worker for temple according to the faction.

public int TEMPLE_GOLD_COST: This attribute represents the cost of gold for temple according to the faction.

public int TEMPLE_PRIEST_INCOME : This attribute represents the income of priest for temple according to the faction.

public int DWELLING_WORKER_COST: This attribute represents the cost of worker for dwelling according to the faction.

public int DWELLING_GOLD_COST: This attribute represents the cost of gold for dwelling according to the faction.

public int DWELLING_WORKER_INCOME: This attribute represents the income of worker for dwelling according to the faction.

public int TERRAFORM_WORKER_COST: This attribute represents the cost of worker for terraforming according to the faction.

public int rangeOfSkipTile: This attribute represents the range of skipping a tile.

public int additionalWorkerOnTunneling: This attribute represents the additional worker for tunneling.

public int remainingSkipRiverChanceOnTown: This attribute represents the remaining skip river chance on town.

public int startingDwellingNum: This attribute represents the dwelling number for the beginning of the game.

public int favorTilesAfterBuildingTemple: This attribute represents the number of favor tiles after building a temple.

public boolean canPlayDoubleTurn: This attribute represents if faction has a feature to take double turn.

public int spadeNeededToTerraformPlains: This attribute represents the cost of spade to terraform a landscape to plains.

public int spadeNeededToTerraformSwamp: This attribute represents the cost of spade to terraform a landscape to swamp.

public int spadeNeededToTerraformLakes: This attribute represents the cost of spade to terraform a landscape to lakes.

public int spadeNeededToTerraformForest: This attribute represents the cost of spade to terraform a landscape to forest.

public int spadeNeededToTerraformMountains: This attribute represents the cost of spade to terraform a landscape to mountains.

public int spadeNeededToTerraformWasteland: This attribute represents the cost of spade to terraform a landscape to wasteland.

public int spadeNeededToTerraformDesert: This attribute represents the cost of spade to terraform a landscape to desert.

public int freeSpadesToTerraformIntoHome: This attribute represents the number of free spades to terraform a landscape a home terrain.

public int priestNeededToSkipTile: This attribute represents the number of priests needed to skip tile.

public boolean freeTerraformOnSpecialAction: This attribute represents if faction has a feature to terraform for free on special action.

public boolean gainFavorTileAfterStronghold: This attribute represents if faction has a feature to gain favor tile after building stronghold.

public boolean gainActionTokenAfterStronghold: This attribute represents if faction has a feature to gain action token after building stronghold.

public boolean hasShipping: This attribute represents if faction has a shipping feature.

4.5.7. Religion Class

Attributes:

private final int MAX_LENGTH: This attribute represents the maximum length for cult levels.

private int orderOfCult_3: This attribute holds id of player who took this area

private int orderOfCult_2_1: This attribute holds id of player who took this area

private int orderOfCult_2_2: This attribute holds id of player who took this area

private int orderOfCult_2_3: This attribute holds id of player who took this area

private boolean keyPlaced: This attribute represents if a key is placed.

private int[] playerPositions: This attribute holds values of the positions of players

private int[] powerAwardPositions: This attribute holds values of the positions of power awards.

Methods:

private void setupReligion(int playerCount, int[] initial_religion_points): This method is to set up religion board at the beginning of the game according to chosen factions and game rules.

public int updateReligion(int count, int player_id, boolean key): This method is to update the position of a player in religion.

public int placePriest(int player_id,boolean key): This method is for a player to place their priest.

public boolean isOccupied(int index): This method checks if the place is occupied.

public int addOrderOfReligion(int player_id, boolean key): This method is to add order of religion for a player.

4.5.8. CardsAndTiles Class

Attributes:

public ArrayList<TownTile> townTiles: This attribute holds town tiles lists.

public ArrayList<BonusCard> bonusCards: This attribute holds bonus card lists.

public ArrayList<FavorTile> favorTiles: This attribute holds favor tiles lists.

public ArrayList<ScoringTile> scoringTiles: This attribute holds scoring tiles lists.

public ArrayList<BonusCard> selectedBonusCards: This attribute holds selected Bonus cards lists since in the game, there are only player + 3 bonus cards.

public ArrayList<ScoringTile> selectedScoringTiles: This attribute holds selected scoring tiles lists since in the game, there are only six scoring tiles.

Methods:

public CardsAndTiles(int playerNumber) : Method constructs the CardsAndTiles object and initializes townTiles, bonusCards, favorTiles, scoringTiles, selectedBonusCards with calling selectedBonusCard method, selectedScoringTiles with calling selectedScoringTiles method; and also It calls creteTownTile, createBonusCard, createFavorTile, createScoringTile methods.

public static void createTownTile(ArrayList<TownTile> townTiles): Method initializes town tiles and adds them into townTiles arraylist.

public static void createBonusCard(ArrayList<BonusCard> bonusCards) : Method initializes bonus cards and adds them into townTiles arraylist.

public static void createFavorTile(ArrayList<FavorTile> favorTiles): Method initializes favor tiles and adds them into townTiles arraylist.

public static void createScoringTile(ArrayList<ScoringTile> scoringTiles): Method initializes scoring tiles and adds them into townTiles arraylist.

public int findPlayerCard(int playerId) : Method tries to find player's cardId with given playerId and returns playerCardId.

private ArrayList<ScoringTile> selectedScoringTiles(ArrayList<ScoringTile> scoringTiles): It selects six the scoring tiles randomly and returns selected tiles arraylists.

private ArrayList<BonusCard> selectedBonusCard(ArrayList<BonusCard> bonusCards,int playerNumber) : It selects number of player + 3 bonus cards amoung all bonus cards and it returns selected bonus cards as arraylist.

4.5.8.1. Bonus Card Class

Attributes:

private int notTakenBonus: If card was not taken previous round, it has extra bonus. This attribute show the amount of this bonus.

private int playerId: This attribute holds player id who takes this bonus card.

private int goldBonus: This attributes holds amount of gold bonus which card has.

private int powerBonus: This attributes holds amount of power bonus which card has.

private int workerBonus: This attributes holds amount of worker bonus which card has.

private int priestBonus: This attributes holds amount of priest bonus which card has.

private int shippingRange: This attributes holds amount of shipping range bonus which card has.

private boolean playerOccupied: This attribute determines whether a player took this card in current round or not.

private boolean specialSpade: This attribute determines whether card has special spade action or not.

private boolean specialCult: This attribute determines whether card has special cult action or not.

private boolean isDwelling: This attribute determines whether card has dwelling bonus or not.

private boolean isTradeHouse: This attribute determines whether card has trade house bonus or not.

private boolean isSanctuary: This attribute determines whether card has sanctuary bonus or not.

Methods:

public BonusCard(int notTakenBonus, boolean playerOccupied,int goldBonus, int powerBonus, int shippingRange, int workerBonus, int priestBonus, boolean spacialSpade, boolean specialCult, boolean isDwelling, boolean isTradeHouse, boolean isSanctuary) : This constructs the BonusCard object and sets playerId with -1 as default.

4.5.8.2. Favor Tile Class

Attributes:

private ArrayList<Integer> playerIds: One favor tile can be taken by different players. This attribute is an integer arraylist which holds players' ids.

private int numberOfPlayer: It holds how many player can take this favor tile.

private int islamBonus: This attribute holds amount of islam bonus which tile has. According this amount, player forwards in islam track.

private int christianityBonus: This attribute holds amount of christianity bonus which tile has. According this amount, player forwards in christianity track.

private int hinduismBonus: This attribute holds amount of hinduism bonus which tile has. According this amount, player forwards in hinduism track.

private int jewBonus: This attribute holds amount of jew bonus which tile has. According this amount, player forwards in jew track.

private int neededCombinedPowerTown: When player builds a town, he/she needs 7 combined bonus as default. This attribute provides player with opportunities to reduce this needed combined power.

private int goldBonus: This attributes holds amount of gold bonus which tile has.

private int powerBonus: This attributes holds amount of power bonus which tile has.

private int workerBonus: This attributes holds amount of worker bonus which tile has.

private int victoryPoint: This attributes holds amount of victory point bonus which tile has.

private boolean isSpecialCult: This attribute determines whether tile has special cult action or not.

private boolean isDwelling: This attribute determines whether tile has dwelling bonus or not.

private boolean isTradeHouse: This attribute determines whether tile has trade house bonus or not.

private boolean isTradeHouse: This attribute determines whether tile has town bonus or not.

private boolean isPassingBonusForTradingHouse: This attribute determines whether tile has every round bonus. When players pass the round, according to its number of structure, namely dwelling, trade house, sanctuary, players take victory bonus.

Methods:

public FavorTile(int numberOfPlayer,int islamBonus, int christianityBonus, int hinduismBonus, int jewBonus, int neededCombinedPowerTown, int powerBonus, int workerBonus, int goldBonus, int victoryPoint, boolean isSpecialCult, boolean

isTownBonus, boolean isTradingHouse, boolean isDwellingBonus, boolean isPassingBonusForTradingHouse) : Method constructs the FavorTile object and it initializes the playerIds ArrayList.

4.5.8.3. Scoring Tile Class

Attributes:

private ArrayList<Integer> playerId: One Scoring tile can be taken by different players. This attribute is an integer arraylist which holds players' ids.

private int requiedIslam: To take this tile, there is religious progress requirement. This attribute holds how much progress needs to take this tile in islam track.

private int requiredBudism: To take this tile, there is religious progress requirement. This attribute holds how much progress needs to take this tile in buddhism track.

private int requiredChrist: To take this tile, there is religious progress requirement. This attribute holds how much progress needs to take this tile in christianity track.

private int requiredJudaism: To take this tile, there is religious progress requirement. This attribute holds how much progress needs to take this tile in judaism track.

private int goldBonus: This attributes holds amount of gold bonus which tile has.

private int powerBonus: This attributes holds amount of power bonus which tile has.

private int workerBonus: This attributes holds amount of worker bonus which tile has.

private int victoryPoint: This attributes holds amount of victory point bonus which tile has.

private int priestBonus: This attributes holds amount of priest bonus which tile has.

private int spadeBonus: This attributes holds amount of spade bonus which tile has.

private boolean isDwelling: This attribute determines whether tile has dwelling bonus or not.

private boolean isTradeHouse: This attribute determines whether tile has trade house bonus or not.

private boolean isStronghold: This attribute determines whether tile has sanctuary bonus or not.

private boolean isRequiredIslam: This attribute determines whether tile requires islam progress to get bonuses or not.

private boolean isRequiredHinduism: This attribute determines whether tile requires hinduism progress to get bonuses or not.

private boolean isRequiredChrist: This attribute determines whether tile requires Christianity progress to get bonuses or not.

private boolean isRequiredJudaism: This attribute determines whether tile requires judaism progress to get bonuses or not.

private boolean isRequiredSpade: This attribute determines whether tile requires spade to get bonuses or not.

private boolean isRequiredTown: This attribute determines whether tile requires to build town to get bonuses or not.

Methods:

public ScoringTile(int requiredIslam, int requiredBudism, int requiredChrist, int requiredJudaism, int priestBonus, int powerBonus, int workerBonus, int victoryBonus, int spadeBonus,int goldBonus, boolean isDwellingBonus, boolean isTradingHouseBonus, boolean isStrongHoldBonus, boolean isRequiredIslam, boolean isRequiredHinduism, boolean isRequiredChrist, boolean isRequiredJudaism, boolean isRequiredSpade, boolean isRequiredTown): This method constructs the ScoringTile object and initializes playerId ArrayList.

4.5.8.4. Town Tile Class

Attributes:

private int playerId: This attribute holds the ids of the players those occupied this tile .

private int powerBonus: This attribute determines how many power this tile will give.

private int priestBonus: This attribute determines how many priest this tile will give.

private int keyCount: This attribute determines how many keys this tile will give.

private int victoryBonus: This attribute determines how many victory points this tile will give.

private int christianityPoint: This attribute determines how many christianity this tile will give.

private int hinduismPoint: This attribute determines how many hinduism point this tile will give.

private int islamPoint: This attribute determines how many islam point this tile will give.

private int jewishPoint: This attribute determines how many jewish point this tile will give.

private int workerBonus: This attribute determines how many worker this tile will give.

private int goldBonus: This attribute determines how many gold this tile will give.

private boolean occupied: This attribute shows if any player selected this tile.

Methods:

public TownTile(int powerBonus, int priestBonus, int victoryBonus, int christianityPoint, int hinduismPoint, int islamPoint, int jewishPoint, int workerBonus, int goldBonus): It constructs the TownTile object and sets playerId with -1 and occupied with false as default.

4.5.9. Space Class

Attributes:

boolean isOccupied: This attribute show that whether this space is occupied by a player or not.

String type: This represents the type of the space.

Methods:

public Space(): This constructs the space object and sets isOccupied as false.

public boolean isOccupied(): This method returns whether the space is occupied by a player.

public buildBridge(Space space1): This method builds a bridge between this terrain and the specified terrain.

public Space[] getBridgeList(): This method returns which terrain can be rachable from that terrain with a bridge.

public void setOccupied(boolean occupied): This method sets that terrain as occupied.

public String getType(): This method returns the type of the space.

public void setType(String type): This method sets the type of the space.

4.5.10. Terrain Class

Attributes:

private Structure structure: This attribute represents the structure that built on this terrain object.

Methods:

public Terrain() : This is the constructor of terrain class and initializes the structure.

4.5.11. Structure Class

Attributes:

private int buildingScore: This is the score of building and used for calculating town score.

Methods:

public Structure(): This is the constructor of structure class and sets building score as 0.

protected void setBuildingScore(int buildingScore): This method sets the building score.

4.5.12. SpecialActionToken Class

Attributes:

public boolean isSpade: This attributes represents if spade action is selected.

public boolean isCultTack: This attributes represents if religion action is selected.

public boolean isStrongholdAbility: This attributes represents if using stronghold ability action is selected.

public boolean isFactionAbility: This attribute represents if given instance is a faction ability

Methods:

public SpecialActionToken(): This is the constructor class and initializes all attributes with false.