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.

1.2.1.4. Functionality

Terra Mystica is a complex game, players can do several actions. We tried to implement most of these functions in order to do not restrict the functions of the original game. However, in some corner case we prefer rapid development over functionality.

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. Rapid Development

Since Terra Historica is a game and players look forward for it, we had to complete it before the announced demo date, May 19. In order to achieve this goal we prioritize rapid development over other design goals in some corner cases of game that will not affect user experience so much.

1.2.2.3. 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.

1.2.2.4. 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.

1.2.2.5. Reusability

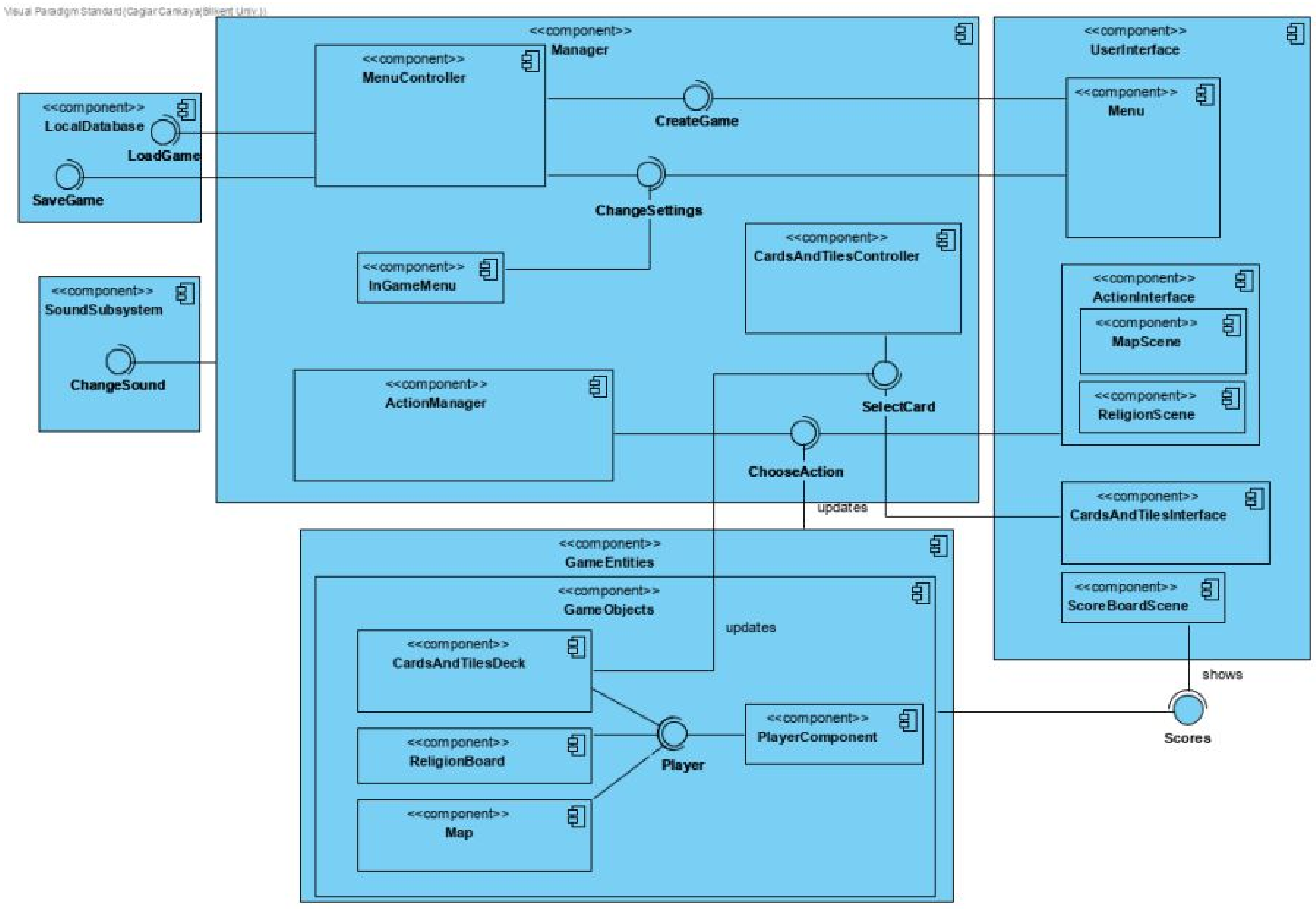
Some classes in Terra Historica can be used in other games without requiring any changes. For instance, main menu class in GUI.

1.2.2.6. 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


Visual Paradigm Standard(Cagiar Cankaya(Bilkent Univ.))
<<component>> Manager
<<component>> MenuController
<<component>> LocalDatabase
LoadGame
SaveGame
CreateGame
ChangeSettings
<<component>> InGameMenu
<<component>> CardsAndTilesController
<<component>> SoundSubsystem
ChangeSound
<<component>> ActionManager
SelectCard
ChooseAction
updates
<<component>> UserInterface
<<component>> Menu
<<component>> ActionInterface
<<component>> MapScene
<<component>> ReligionScene
<<component>> CardsAndTilesInterface
<<component>> ScoreBoardScene
shows
Scores
<<component>> GameEntities
<<component>> GameObjects
updates
<<component>> CardsAndTilesDeck
<<component>> ReligionBoard
<<component>> Map
Player
<<component>> PlayerComponent

Model 1: Subsystem Decomposition

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 *Model 1*. Subsystems will be explained in a detailed manner later.

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

Model Architecture: Subsystem having model architecture is called Game Entities. It contains components of game such as map, religion, cards and tiles and player.

View Architecture: Subsystem having view architecture is called User Interface. It contains components to provide a n interface for user.

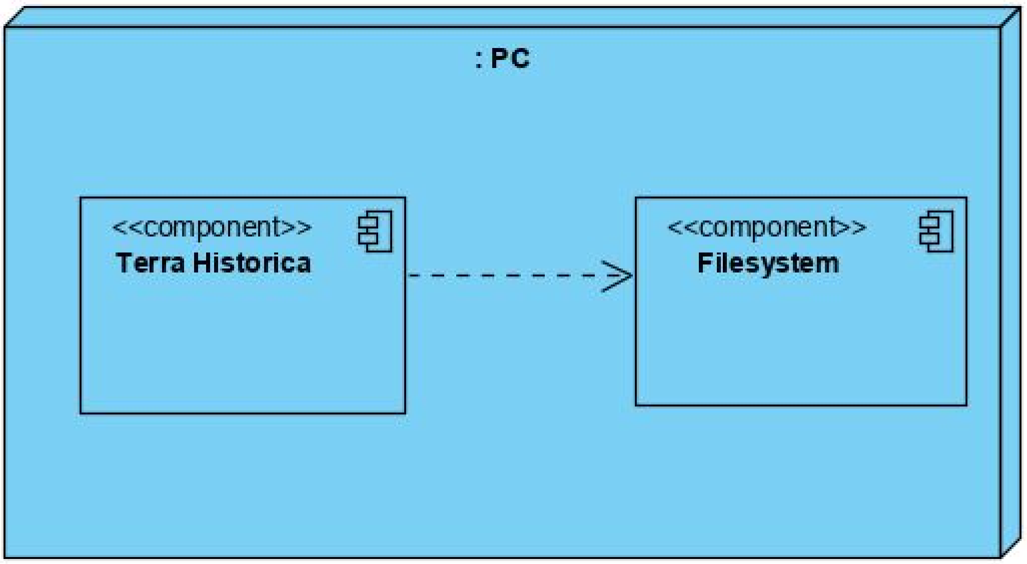
Controller Architecture: Subsystem having controller architecture is called Manager. It contains components for the game that handles inputs and updates model and view accordingly.

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.

2.5.1.1. Save and Load

When players want to save the game, current game's data (eg: which player chose which faction, current resources of each player, which terrains are occupied by which players and which buildings were built on these terrains, whose turn is it etc.) will be sent to local filesystem. For each different game, a new file will be created with new names (eg: game1.txt) and data of these games will be stored on their files with an order. When a saved game is wanted to be loaded, system finds the proper file for the game and pulls necessary data for the game and starts the game.

2.5.2. Termination Of System

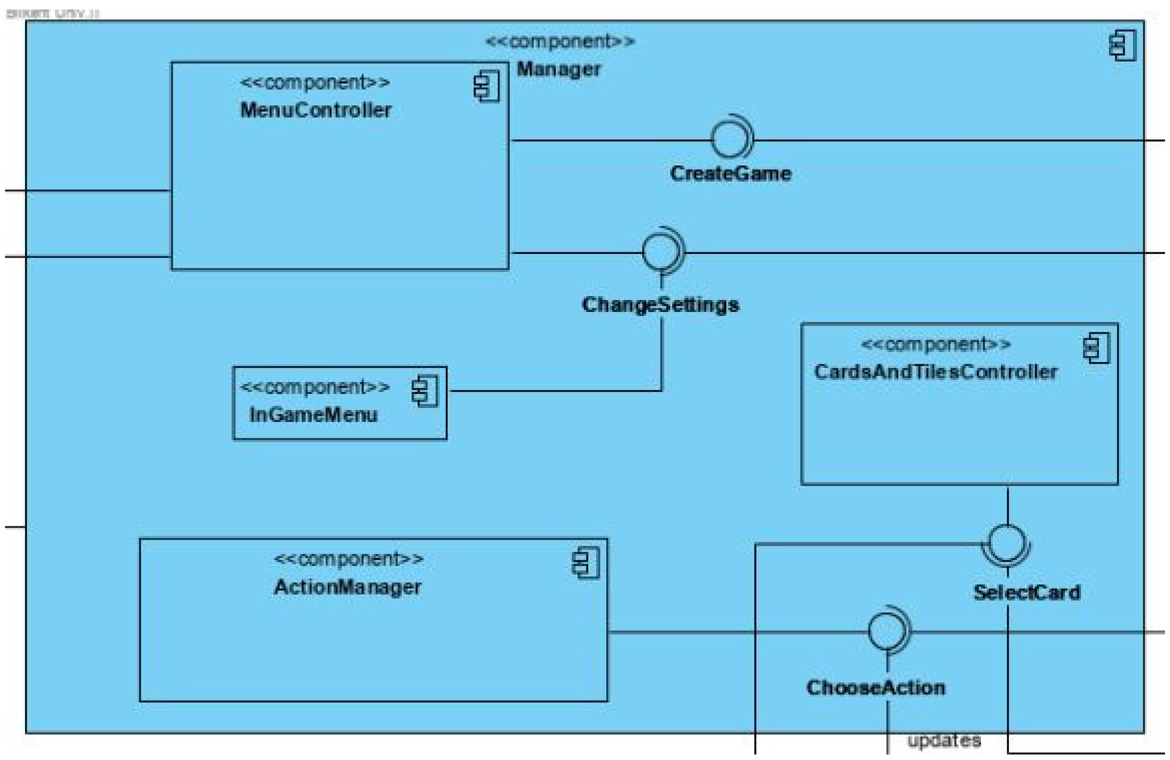
System will be terminated if user chooses to exit in the main menu or in-game menu.

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. Manager Subsystem



Model 3

Manager Subsystem provides the data transformation between entity and the boundary objects. There are four main components in controller subsystem such as MenuController, InGameMenu, ActionManager and CardsAndTilesController.

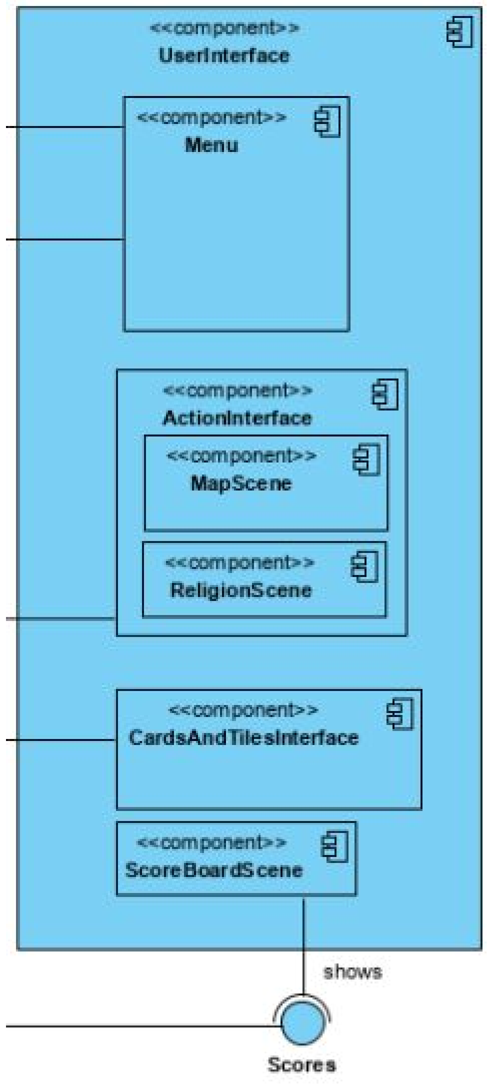
InGameMenu is responsible for controlling data transition between user’s selection related to settings and saved game datas. InGameMenu provides ,as it mentioned before, Change Settings interface in order to give chance to change settings during game. Also, it provides saving or loading the game.

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.

CardsAndTilesController is responsible for controlling data transition between user’s selections related to cards and model objects by providing Select Card interface which is responsible for showing cards and associating players with it in both UserInterface and GameEntities subsystems.

ActionManager is responsible for controlling data transition between user’s selections related to actions and model object those changing such as player, map objects. Furthermore, this subsystem provides ChoseAction interface which also designs change among views related to any action.

3.2. User Interface Subsystem



Model 4

View Subsystem is responsible for providing user interface of Terra Historica. MVC design pattern is used for this subsystem. View Subsystem has 4 main components which are Menu, ActionInterface, CardsAndTilesInterface, ScoreBoardScene. Menu component is responsible for representing Main Menu Views such as CreateGame, LoadGame, ShowCredits and ChangeSettings.

ActionInterface component represents in-game views such as MapScene, ReligionScene components.

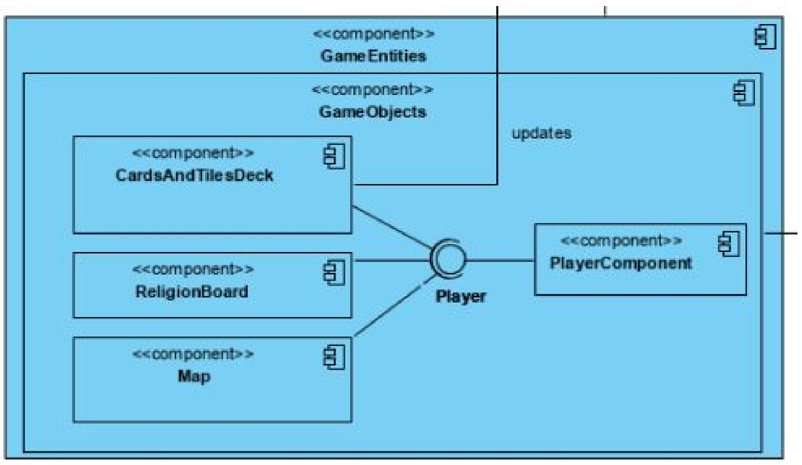
MapScene: Map related all components are included in this component.

ReligionScene: ReligionScene includes views such as ReligionView and ReligionSlotView objects.

CardsAndTilesInterface, this component shows the score table interface of each players.

Finally, ScoreBoardScene is responsible for all view that show score of players. This component using Scores interface in order to update view according to points player have.

3.3. Game Entities Subsystem



Model 5

As mentioned before, GameEntities subsystem represents the model in design pattern MVC. Simply GameEntites are updated by ChoseAction and Select Card interfaces according to player inputs.

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.

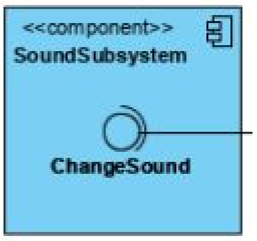
Map Component: This component includes map object with spaces and different structures that can be built on map.

Religion Component: This component includes religion object.

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.

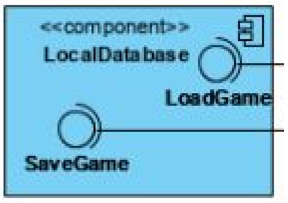
3.4. Sound Subsystem



Model 6

This subsystem is designed to operate sound aspect of the game. This subsystem contains SoundController class and InGameMenuController class. Thanks to this subsystem users can adjust the volume of the game or even mute it.

3.5. Local Database Subsystem



Model 7

This subsystem responsible for saving and loading processes of the game. LoadGameController and FileManager classes are the main classes that operates this subsystem.

Serialization key feature of local database subsystem. Every object that must be saved implements serializable interface. A save file created with the name of the gameId when the game is created. If users choose to save game, required objects will be saved to the file. Load operation is the inverse of save operation. All these are operated by local database subsystem.

4. Low-level Design

4.1. Object Design Trade-offs

4.1.1. Performance vs Security

The game data is stored in local filesystem. Thanks to this when players save or load their games, system will be able to do it without delays related to online database system and internet connection. However other users of the pc can easily access the game data via local filesystems but that is not so important, because the system does not stores any private data. For these reasons we prefer performance over security.

4.1.2. Functionality vs Usability

Although we tried to design a simple and usable system as much as possible; Terra Mystica is a complex board game and we did not want to remove features of the original game. That means priority of functionality is higher than priority of usability for us even if 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.

4.1.4. Portability vs Functionalityy

Terra Historica is designed to make the game as portable as possible. In order to achieve this goal, we chose screen resolution to be fixed as 1280x780 rather than 1920x1080 since, the computers with larger screen resolution can also run the game therefore wider number of players can play game. However, this selection causes abandoning functionality of playing game in full screen.

4.2. Design Patterns

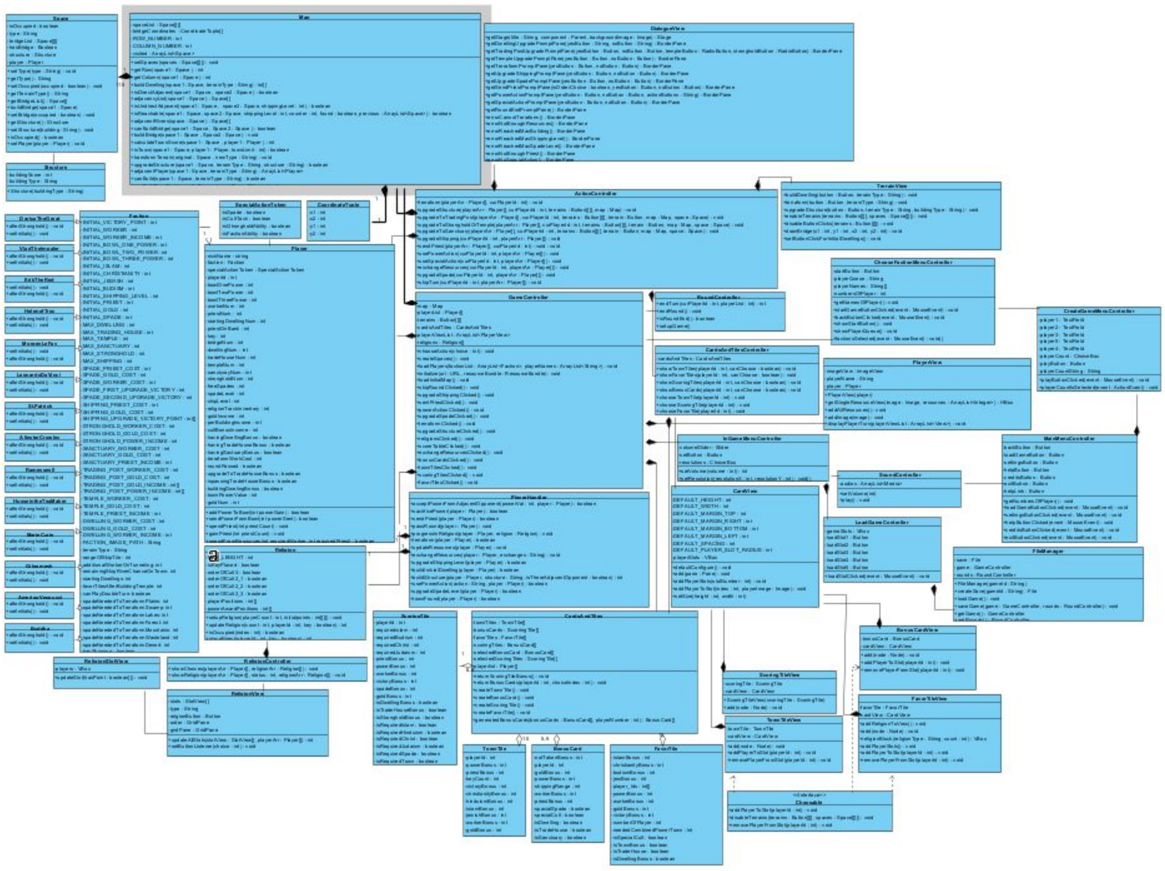
4.2.1. Facade Design Pattern

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.

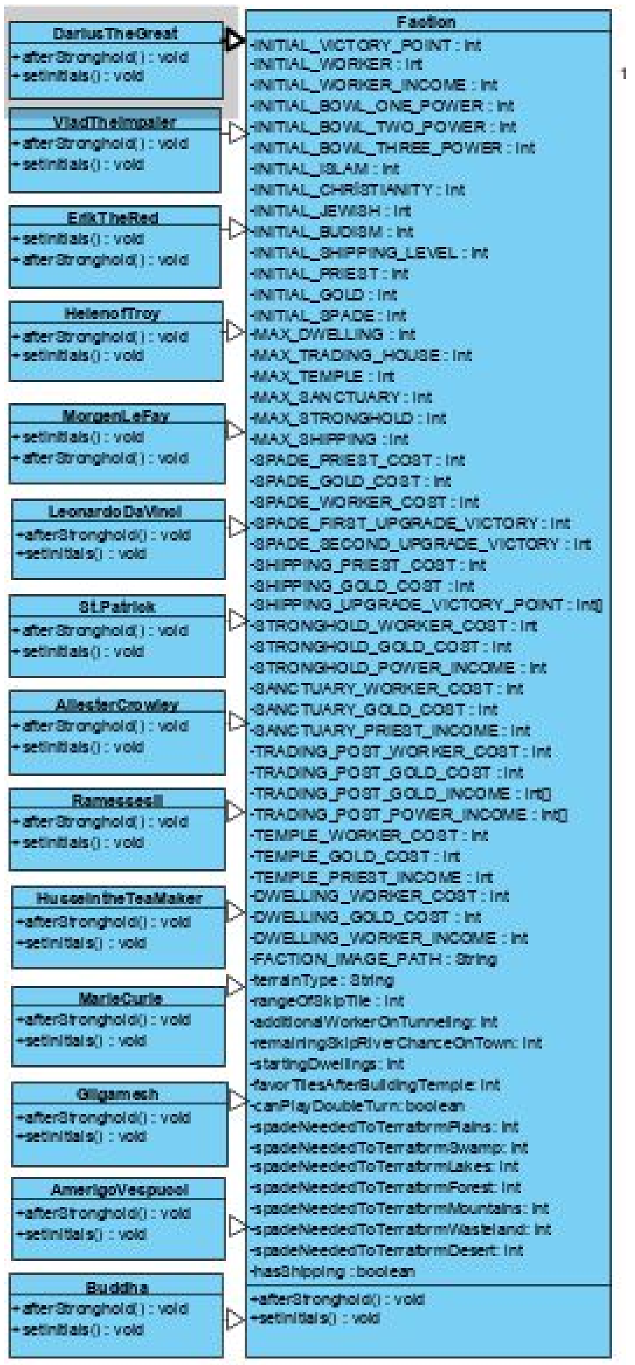
4.2.2. Builder Design Pattern

Terra Historica also implements builder design pattern. In the game, there are various building types which share same attributes and can transform each other (eg: Dwelling, Sanctuary, Stronghold, Temple, TradingHouse). Their methods also are similar. Moreover, each terrain in the map can be transformed to another (eg: Desert to Swamp, Plains to Mountain etc.) by the same method. Therefore, our design fits builder design pattern.

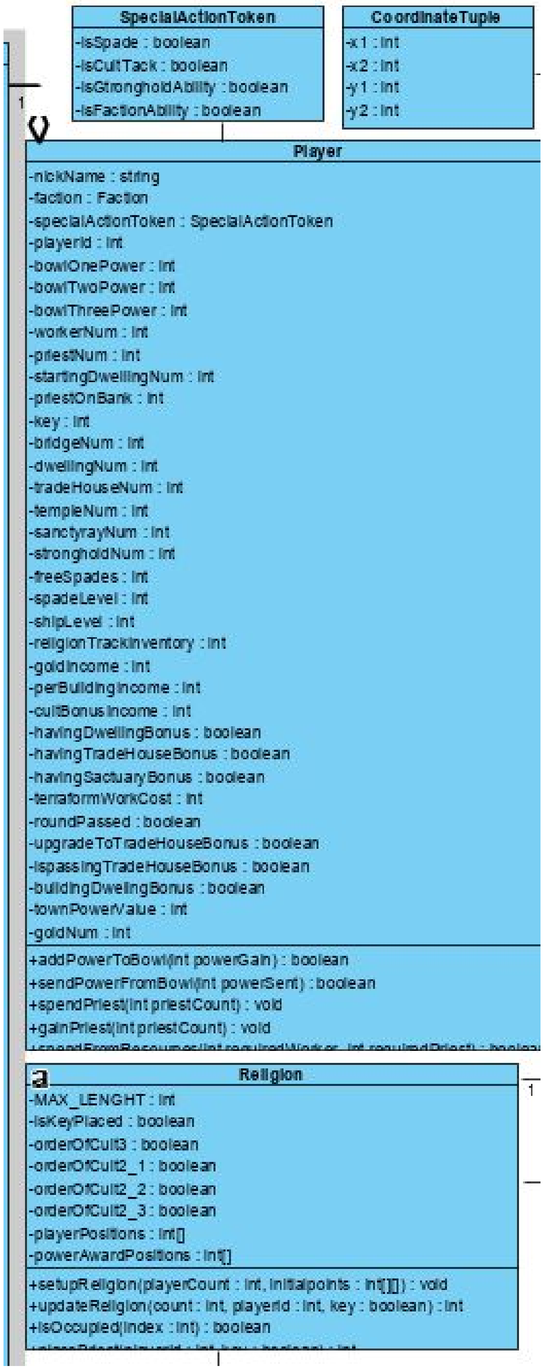
4.3. Final Object Design



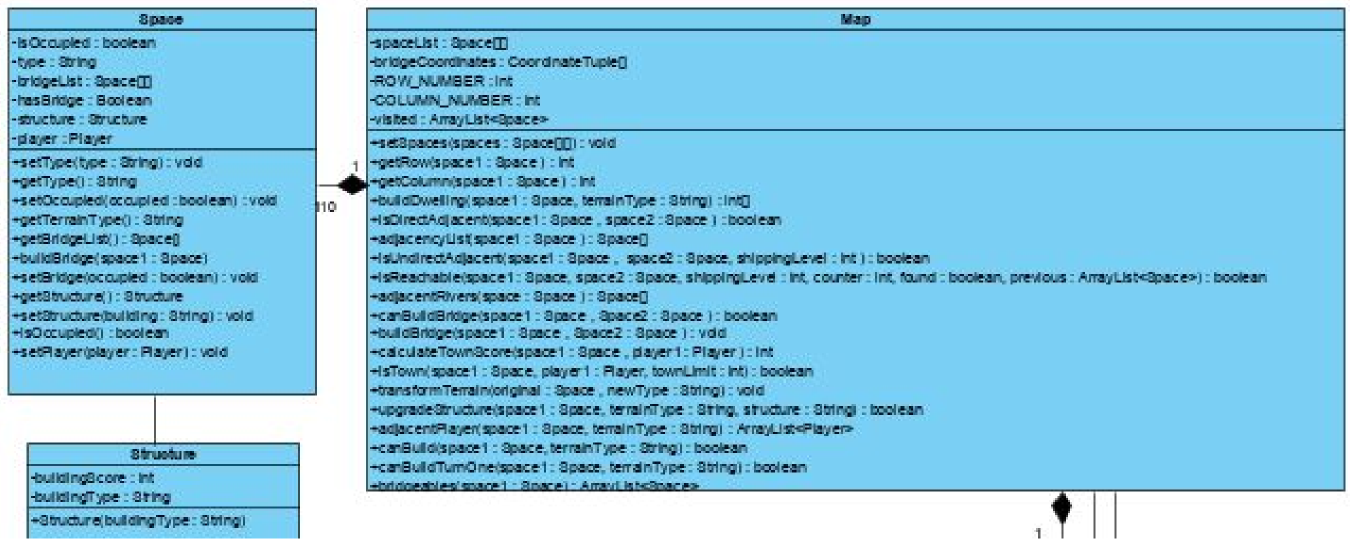
Model 7


DariusTheGreat
+afterStronghold() : void
+setInitials() : void
VladTheImpaler
+afterStronghold() : void
+setInitials() : void
ErikTheRed
+setInitials() : void
+afterStronghold() : void
HelenofTroy
+afterStronghold() : void
+setInitials() : void
MorgenLeFay
+setInitials() : void
+afterStronghold() : void
LeonardoDaVinci
+afterStronghold() : void
+setInitials() : void
St.Patrick
+afterStronghold() : void
+setInitials() : void
AllesterCrowley
+afterStronghold() : void
+setInitials() : void
RamessesII
+afterStronghold() : void
+setInitials() : void
HusseintheTeaMaker
+afterStronghold() : void
+setInitials() : void
MarieCurie
+afterStronghold() : void
+setInitials() : void
Gilgamesh
+afterStronghold() : void
+setInitials() : void
AmerigoVespucci
+afterStronghold() : void
+setInitials() : void
Buddha
+afterStronghold() : void
+setInitials() : void
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
-FACTION_IMAGE_PATH : String
-terrainType : String
-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
+setInitials() : void
1

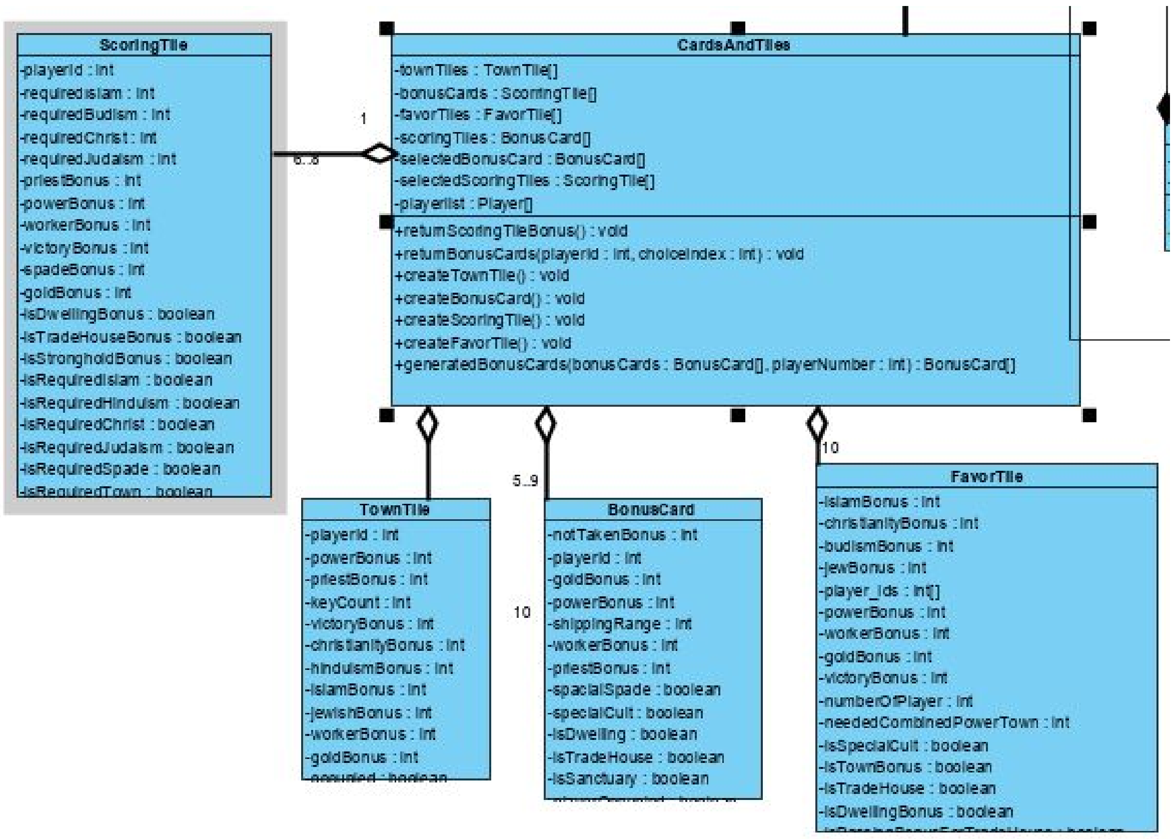
Model 8


SpecialActionToken
-isSpade : boolean
-isCultTack : boolean
-isGtrongholdAbility : boolean
-isFactionAbility : boolean
CoordinateTuple
-x1 : int
-x2 : int
-y1 : int
-y2 : int
1
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
-freeSpades : 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
-goldNum : int
+addPowerToBowl(int powerGain) : boolean
+sendPowerFromBowl(int powerSent) : boolean
+spendPriest(int priestCount) : void
+gainPriest(int priestCount) : void
Religion
1
-MAX_LENGHT : int
-isKeyPlaced : boolean
-orderOfCult3 : boolean
-orderOfCult2_1 : boolean
-orderOfCult2_2 : boolean
-orderOfCult2_3 : boolean
-playerPositions : int[]
-powerAwardPositions : int[]
+setupReligion(playerCount : int, initialpoints : int[][]) : void
+updateReligion(count : int, playerId : int, key : boolean) : int
+isOccupied(index : int) : boolean

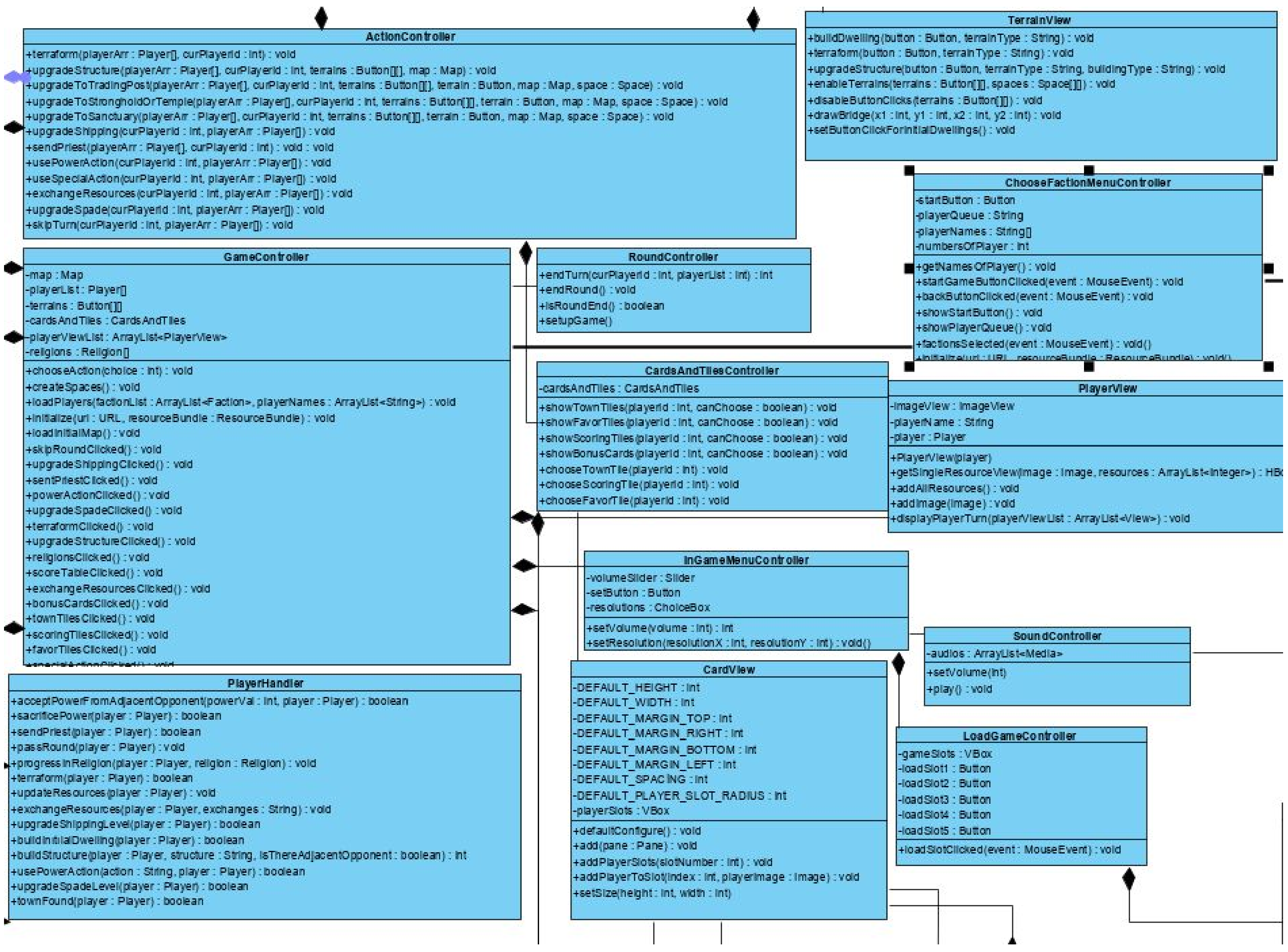
Model 9


Space
-isOccupied : boolean
-type : String
-bridgeList : Space[][]
-hasBridge : Boolean
-structure : Structure
-player :Player
+setType(type : String) : void
+getType() : String
+setOccupied(occupied :boolean) :void
+getTerrainType() : String
+getBridgeList() : Space[]
+buildBridge(space1 : Space)
+setBridge(occupied : boolean) : void
+getStructure() : Structure
+setStructure(building : String) : void
+isOccupied() :boolean
+setPlayer(player : Player) : void
1
110
Map
-spaceList : Space[][]
-bridgeCoordinates : CoordinateTuple[]
-ROW_NUMBER :int
-COLUMN_NUMBER : int
-visited : ArrayList<Space>
+setSpaces(spaces : Space[][]) : void
+getRow(space1 : Space) : int
+getColumn(space1 : Space) : int
+buildDwelling(space1 : Space, terrainType : String) :int[]
+isDirectAdjacent(space1 : Space , space2 :Space) :boolean
+adjacencyList(space1 : Space) : Space[]
+isUndirectAdjacent(space1 : Space , space2 : Space, shippingLevel : int) : boolean
+isReachable(space1 : Space, space2 : Space, shippingLevel : int, counter : int, found : boolean, previous : ArrayList<Space>) : boolean
+adjacentRivers(space : Space) : Space[]
+canBuildBridge(space1 : Space , Space2 : Space) : boolean
+buildBridge(space1 : Space , Space2 : Space) : void
+calculateTownScore(space1 : Space , player1 : Player) : int
+isTown(space1 : Space, player1 : Player, townLimit : int) : boolean
+transformTerrain(original : Space , newType :String) : void
+upgradeStructure(space1 : Space, terrainType : String, structure : String) : boolean
+adjacentPlayer(space1 : Space, terrainType : String) : ArrayList<Player>
+canBuild(space1 : Space, terrainType : String) : boolean
+canBuildTurnOne(space1 : Space, terrainType : String) : boolean
Structure
-buildingScore : int
-buildingType : String
+Structure(buildingType : String)
1

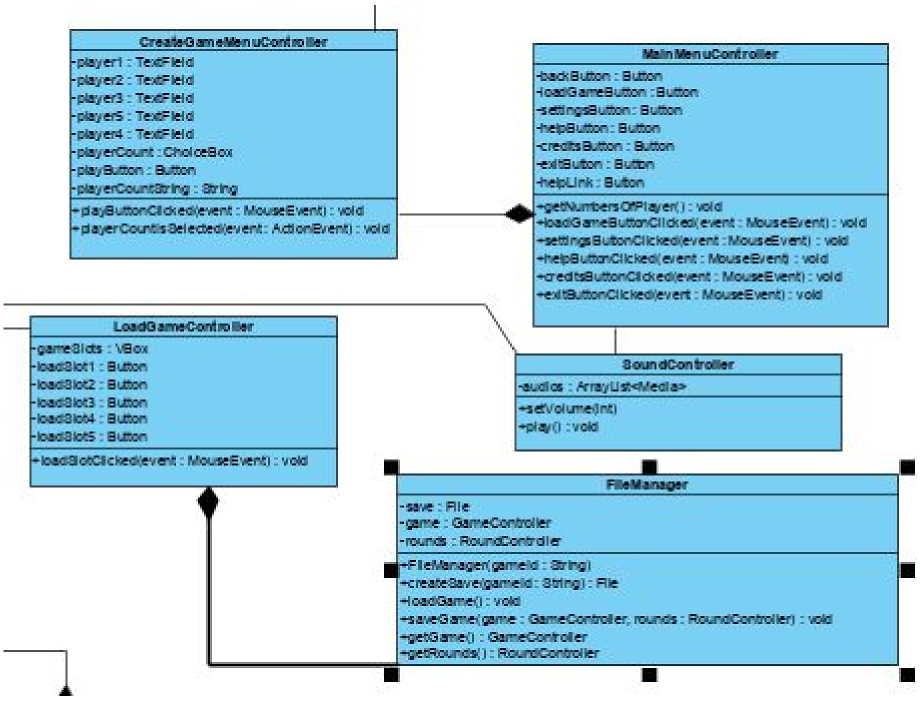
Model 10



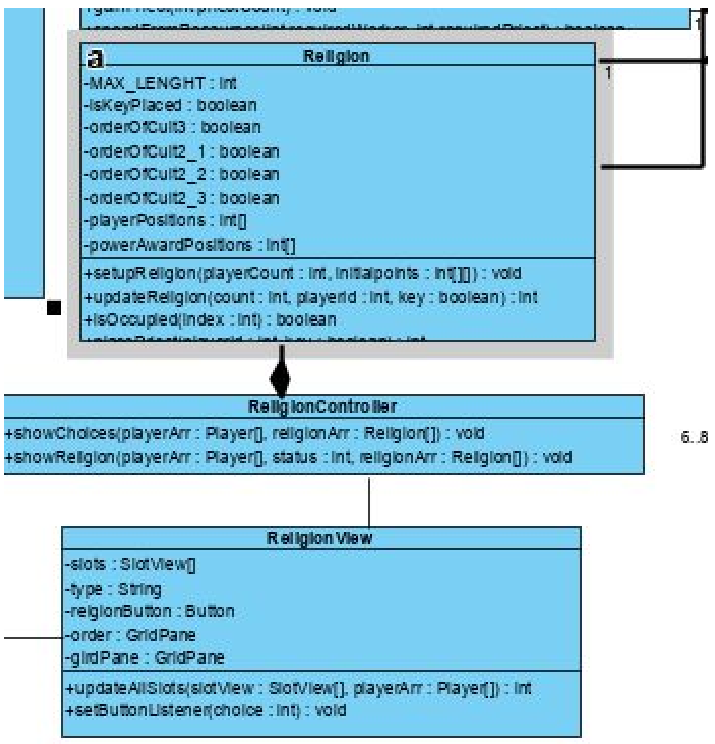
Model 11


ActionController
+terraform(playerArr : Player[], curPlayerId : int) : void
+upgradeStructure(playerArr : Player[], curPlayerId : int, terrains : Button[][], map : Map) : void
+upgradeToTradingPost(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeToStrongholdOrTemple(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeToSanctuary(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeShipping(curPlayerId : int, playerArr : Player[]) : void
+sendPriest(playerArr : Player[], curPlayerId : int) : void : void
+usePowerAction(curPlayerId : int, playerArr : Player[]) : void
+useSpecialAction(curPlayerId : int, playerArr : Player[]) : void
+exchangeResources(curPlayerId : int, playerArr : Player[]) : void
+upgradeSpade(curPlayerId : int, playerArr : Player[]) : void
+skipTurn(curPlayerId : int, playerArr : Player[]) : void
TerrainView
+buildDwelling(button : Button, terrainType : String) : void
+terraform(button : Button, terrainType : String) : void
+upgradeStructure(button : Button, terrainType : String, buildingType : String) : void
+enableTerrains(terrains : Button[][], spaces : Space[][]) : void
+disableButtonClicks(terrains : Button[][]) : void
+drawBridge(x1 : int, y1 : int, x2 : int, y2 : int) : void
+setButtonClickForInitialDwellings() : void
ChooseFactionMenuController
-startButton : Button
-playerQueue : String
-playerNames : String[]
-numbersOfPlayer : int
+getNamesOfPlayer() : void
+startGameButtonClicked(event : MouseEvent) : void
+backButtonClicked(event : MouseEvent) : void
+showStartButton() : void
+showPlayerQueue() : void
+factionsSelected(event : MouseEvent) : void()
+initialize(url : URL, resourceBundle : ResourceBundle) : void()
GameController
-map : Map
-playerList : Player[]
-terrains : Button[][]
-cardsAndTiles : CardsAndTiles
-playerViewList : ArrayList<PlayerView>
-religions : Religion[]
+chooseAction(choice : int) : void
+createSpaces() : void
+loadPlayers(factionList : ArrayList<Faction>, playerNames : ArrayList<String>) : void
+initialize(url : URL, resourceBundle : ResourceBundle) : void
+loadInitialMap() : void
+skipRoundClicked() : void
+upgradeShippingClicked() : void
+sentPriestClicked() : void
+powerActionClicked() : void
+upgradeSpadeClicked() : void
+terraformClicked() : void
+upgradeStructureClicked() : void
+religionsClicked() : void
+scoreTableClicked() : void
+exchangeResourcesClicked() : void
+bonusCardsClicked() : void
+townTilesClicked() : void
+scoringTilesClicked() : void
+favorTilesClicked() : void
+specialActionClicked() : void
RoundController
+endTurn(curPlayerId : int, playerList : int) : int
+endRound() : void
+isRoundEnd() : boolean
+setupGame()
CardsAndTilesController
-cardsAndTiles : CardsAndTiles
+showTownTiles(playerId : int, canChoose : boolean) : void
+showFavorTiles(playerId : int, canChoose : boolean) : void
+showScoringTiles(playerId : int, canChoose : boolean) : void
+showBonusCards(playerId : int, canChoose : boolean) : void
+chooseTownTile(playerId : int) : void
+chooseScoringTile(playerId : int) : void
+chooseFavorTile(playerId : int) : void
PlayerView
-imageView : ImageView
-playerName : String
-player : Player
+PlayerView(player)
+getSingleResourceView(image : Image, resources : ArrayList<Integer>) : HB
+addAllResources() : void
+addImage(image) : void
+displayPlayerTurn(playerViewList : ArrayList<View>) : void
InGameMenuController
-volumeSlider : Slider
-setButton : Button
-resolutions : ChoiceBox
+setVolume(volume : int) : int
+setResolution(resolutionX : int, resolutionY : int) : void()
SoundController
-audios : ArrayList<Media>
+setVolume(int)
+play() : void
PlayerHandler
+acceptPowerFromAdjacentOpponent(powerVal : int, player : Player) : boolean
+sacrificePower(player : Player) : boolean
+sendPriest(player : Player) : boolean
+passRound(player : Player) : void
+progressInReligion(player : Player, religion : Religion) : void
+terraform(player : Player) : boolean
+updateResources(player : Player) : void
+exchangeResources(player : Player, exchanges : String) : void
+upgradeShippingLevel(player : Player) : boolean
+buildInitialDwelling(player : Player) : boolean
+buildStructure(player : Player, structure : String, isThereAdjacentOpponent : boolean) : int
+usePowerAction(action : String, player : Player) : boolean
+upgradeSpadeLevel(player : Player) : boolean
+townFound(player : Player) : boolean
CardView
-DEFAULT_HEIGHT : int
-DEFAULT_WIDTH : int
-DEFAULT_MARGIN_TOP : int
-DEFAULT_MARGIN_RIGHT : int
-DEFAULT_MARGIN_BOTTOM : int
-DEFAULT_MARGIN_LEFT : int
-DEFAULT_SPACING : int
-DEFAULT_PLAYER_SLOT_RADIUS : int
-playerSlots : VBox
+defaultConfigure() : void
+add(pane : Pane) : void
+addPlayerSlots(slotNumber : int) : void
+addPlayerToSlot(index : int, playerImage : Image) : void
+setSize(height : int, width : int)
LoadGameController
-gameSlots : VBox
-loadSlot1 : Button
-loadSlot2 : Button
-loadSlot3 : Button
-loadSlot4 : Button
-loadSlot5 : Button
+loadSlotClicked(event : MouseEvent) : void

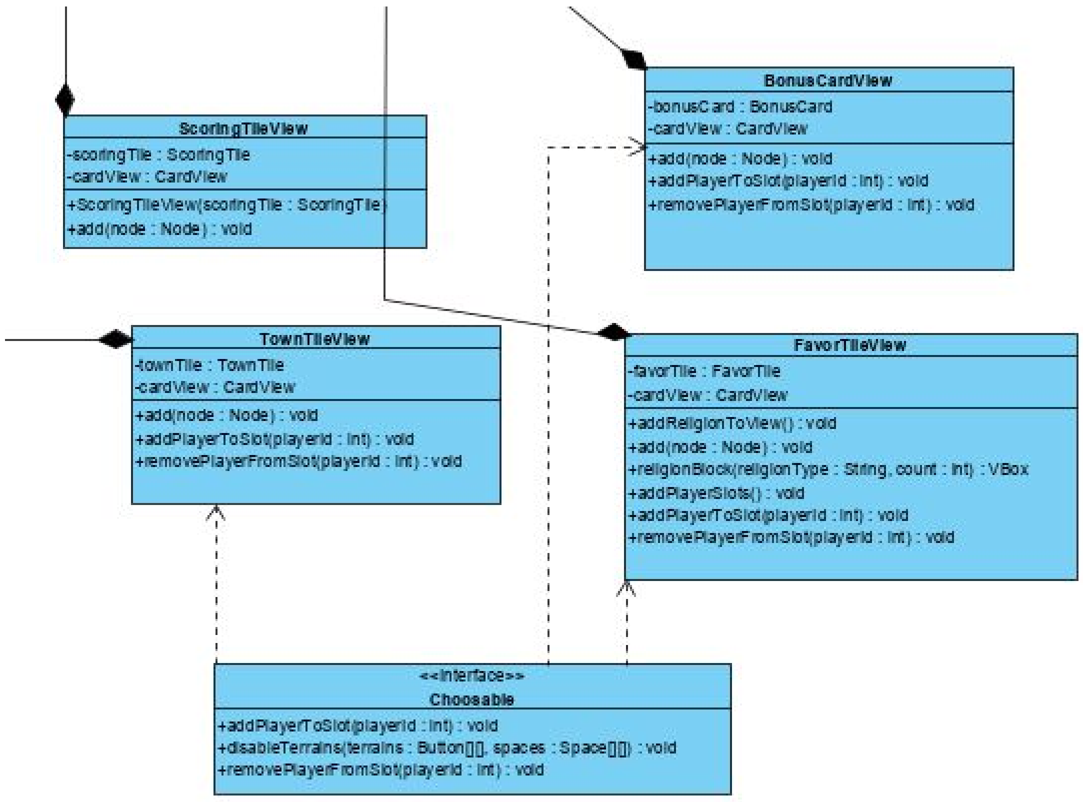
Model 12


CreateGameMenuController
-player1 : TextField
-player2 : TextField
-player3 : TextField
-player5 : TextField
-player4 : TextField
-playerCount : ChoiceBox
-playButton : Button
-playerCountString : String
+playButtonClicked(event : MouseEvent) : void
+playerCountIsSelected(event : ActionEvent) : void
MainMenuController
-backButton : Button
-loadGameButton : Button
-settingsButton : Button
-helpButton : Button
-creditsButton : Button
-exitButton : Button
-helpLink : Button
+getNumbersOfPlayer() : 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
SoundController
-audios : ArrayList<Media>
+setVolume(int)
+play() : void
FileManager
-save : File
-game : GameController
-rounds : RoundController
+FileManager(gameId : String)
+createSave(gameId : String) : File
+loadGame() : void
+saveGame(game : GameController, rounds : RoundController) : void
+getGame() : GameController
+getRounds() : RoundController

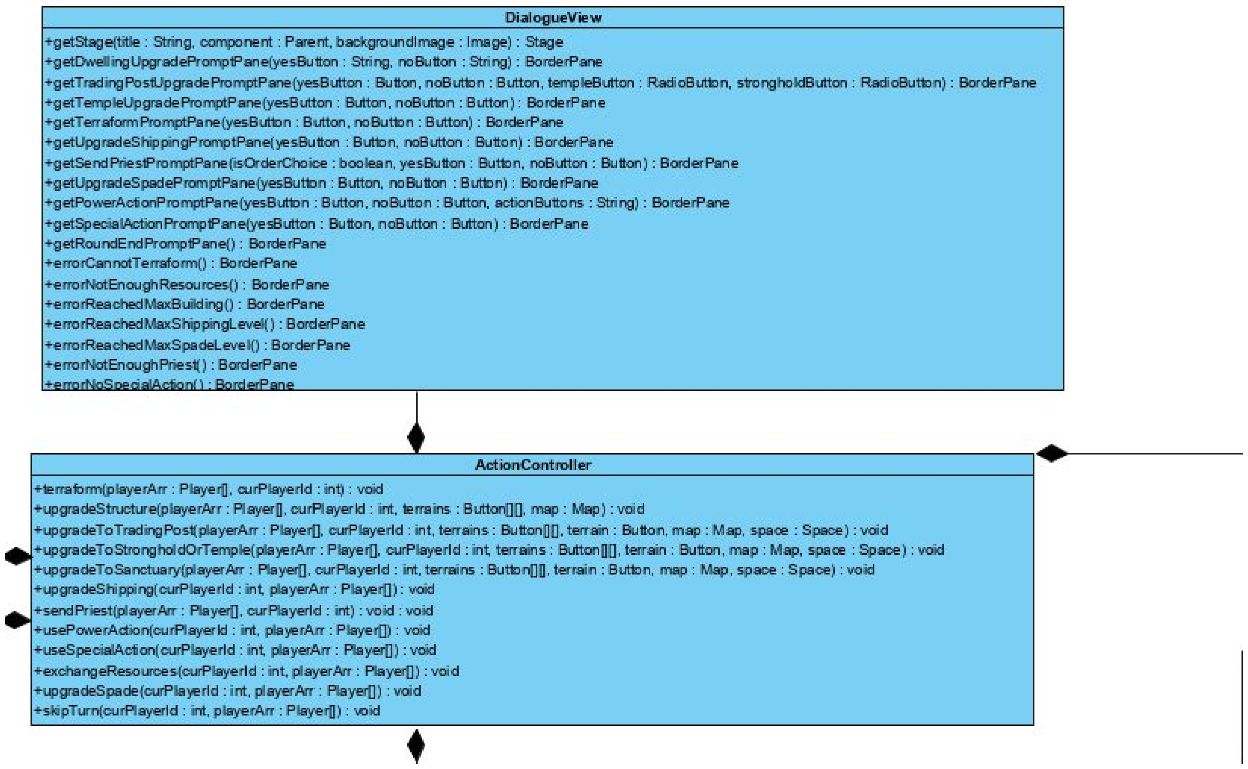
Model 13


Religion
-MAX_LENGHT : int
-isKeyPlaced : boolean
-orderOfCult3 : boolean
-orderOfCult2_1 : boolean
-orderOfCult2_2 : boolean
-orderOfCult2_3 : boolean
-playerPositions : int[]
-powerAwardPositions : int[]
+setupReligion(playerCount : int, initialpoints : int[][]) : void
+updateReligion(count : int, playerId : int, key : boolean) : int
+isOccupied(index : int) : boolean
1
1
6..8
ReligionController
+showChoices(playerArr : Player[], religionArr : Religion[]) : void
+showReligion(playerArr : Player[], status : int, religionArr : Religion[]) : void
ReligionView
-slots : SlotView[]
-type : String
-reigionButton : Button
-order : GridPane
-girdPane : GridPane
+updateAllSlots(slotView : SlotView[], playerArr : Player[]) : int
+setButtonListener(choice : int) : void

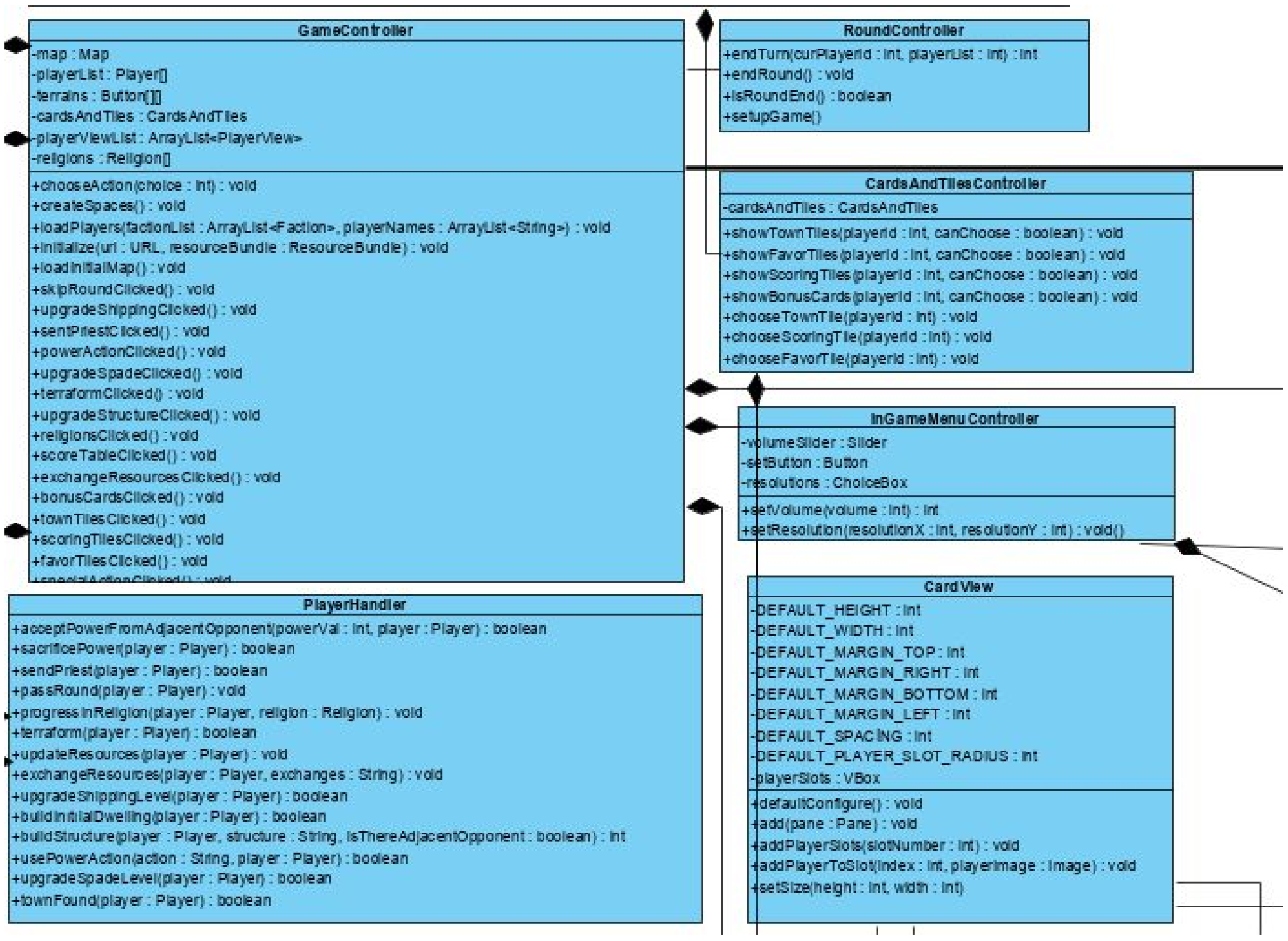
Model 14


ScoringTileView
-scoringTile : ScoringTile
-cardView : CardView
+ScoringTileView(scoringTile : ScoringTile)
+add(node : Node) : void
BonusCardView
-bonusCard : BonusCard
-cardView : CardView
+add(node : Node) : void
+addPlayerToSlot(playerId : int) : void
+removePlayerFromSlot(playerId : int) : void
TownTileView
-townTile : TownTile
-cardView : CardView
+add(node : Node) : void
+addPlayerToSlot(playerId : int) : void
+removePlayerFromSlot(playerId : int) : void
FavorTileView
-favorTile : FavorTile
-cardView : CardView
+addReligionToView() : void
+add(node : Node) : void
+religionBlock(religionType : String, count : int) : VBox
+addPlayerSlots() : void
+addPlayerToSlot(playerId : int) : void
+removePlayerFromSlot(playerId : int) : void
<<interface>>
Choosable
+addPlayerToSlot(playerId : int) : void
+disableTerrains(terrains : Button[][], spaces : Space[][]) : void
+removePlayerFromSlot(playerId : int) : void

Model 15


DialogueView
+getStage(title : String, component : Parent, backgroundImage : Image) : Stage
+getDwellingUpgradePromptPane(yesButton : String, noButton : String) : BorderPane
+getTradingPostUpgradePromptPane(yesButton : Button, noButton : Button, templeButton : RadioButton, strongholdButton : RadioButton) : BorderPane
+getTempleUpgradePromptPane(yesButton : Button, noButton : Button) : BorderPane
+getTerraformPromptPane(yesButton : Button, noButton : Button) : BorderPane
+getUpgradeShippingPromptPane(yesButton : Button, noButton : Button) : BorderPane
+getSendPriestPromptPane(isOrderChoice : boolean, yesButton : Button, noButton : Button) : BorderPane
+getUpgradeSpadePromptPane(yesButton : Button, noButton : Button) : BorderPane
+getPowerActionPromptPane(yesButton : Button, noButton : Button, actionButtons : String) : BorderPane
+getSpecialActionPromptPane(yesButton : Button, noButton : Button) : BorderPane
+getRoundEndPromptPane() : BorderPane
+errorCannotTerraform() : BorderPane
+errorNotEnoughResources() : BorderPane
+errorReachedMaxBuilding() : BorderPane
+errorReachedMaxShippingLevel() : BorderPane
+errorReachedMaxSpadeLevel() : BorderPane
+errorNotEnoughPriest() : BorderPane
+errorNoSpecialAction() : BorderPane
ActionController
+terraform(playerArr : Player[], curPlayerId : int) : void
+upgradeStructure(playerArr : Player[], curPlayerId : int, terrains : Button[][], map : Map) : void
+upgradeToTradingPost(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeToStrongholdOrTemple(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeToSanctuary(playerArr : Player[], curPlayerId : int, terrains : Button[][], terrain : Button, map : Map, space : Space) : void
+upgradeShipping(curPlayerId : int, playerArr : Player[]) : void
+sendPriest(playerArr : Player[], curPlayerId : int) : void : void
+usePowerAction(curPlayerId : int, playerArr : Player[]) : void
+useSpecialAction(curPlayerId : int, playerArr : Player[]) : void
+exchangeResources(curPlayerId : int, playerArr : Player[]) : void
+upgradeSpade(curPlayerId : int, playerArr : Player[]) : void
+skipTurn(curPlayerId : int, playerArr : Player[]) : void

Model 16


GameController
-map : Map
-playerList : Player[]
-terrains : Button[][]
-cardsAndTiles : CardsAndTiles
-playerViewList : ArrayList<PlayerView>
-religions : Religion[]
+chooseAction(choice : int) : void
+createSpaces() : void
+loadPlayers(factionList : ArrayList<Faction>, playerNames : ArrayList<String>) : void
+initialize(url : URL, resourceBundle : ResourceBundle) : void
+loadInitialMap() : void
+skipRoundClicked() : void
+upgradeShippingClicked() : void
+sentPriestClicked() : void
+powerActionClicked() : void
+upgradeSpadeClicked() : void
+terraformClicked() : void
+upgradeStructureClicked() : void
+religionsClicked() : void
+scoreTableClicked() : void
+exchangeResourcesClicked() : void
+bonusCardsClicked() : void
+townTilesClicked() : void
+scoringTilesClicked() : void
+favorTilesClicked() : void
RoundController
+endTurn(curPlayerId : int, playerList : int) : int
+endRound() : void
+isRoundEnd() : boolean
+setupGame()
CardsAndTilesController
-cardsAndTiles : CardsAndTiles
+showTownTiles(playerId : int, canChoose : boolean) : void
+showFavorTiles(playerId : int, canChoose : boolean) : void
+showScoringTiles(playerId : int, canChoose : boolean) : void
+showBonusCards(playerId : int, canChoose : boolean) : void
+chooseTownTile(playerId : int) : void
+chooseScoringTile(playerId : int) : void
+chooseFavorTile(playerId : int) : void
InGameMenuController
-volumeSlider : Slider
-setButton : Button
-resolutions : ChoiceBox
+setVolume(volume : int) : int
+setResolution(resolutionX : int, resolutionY : int) : void()
PlayerHandler
+acceptPowerFromAdjacentOpponent(powerVal : int, player : Player) : boolean
+sacrificePower(player : Player) : boolean
+sendPriest(player : Player) : boolean
+passRound(player : Player) : void
+progressInReligion(player : Player, religion : Religion) : void
+terraform(player : Player) : boolean
+updateResources(player : Player) : void
+exchangeResources(player : Player, exchanges : String) : void
+upgradeShippingLevel(player : Player) : boolean
+buildInitialDwelling(player : Player) : boolean
+buildStructure(player : Player, structure : String, isThereAdjacentOpponent : boolean) : int
+usePowerAction(action : String, player : Player) : boolean
+upgradeSpadeLevel(player : Player) : boolean
+townFound(player : Player) : boolean
CardView
-DEFAULT_HEIGHT : int
-DEFAULT_WIDTH : int
-DEFAULT_MARGIN_TOP : int
-DEFAULT_MARGIN_RIGHT : int
-DEFAULT_MARGIN_BOTTOM : int
-DEFAULT_MARGIN_LEFT : int
-DEFAULT_SPACING : int
-DEFAULT_PLAYER_SLOT_RADIUS : int
-playerSlots : VBox
+defaultConfigure() : void
+add(pane : Pane) : void
+addPlayerSlots(slotNumber : int) : void
+addPlayerToSlot(index : int, playerImage : Image) : void
+setSize(height : int, width : int)

Model 17

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

public void getNumberOfPlayers(): A method that returns number of players in 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 String[] playerNames: A String array that holds the name of the players

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 factionSelected(MouseEvent event): Listener method that assign player faction to the faction that ben chosen

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

4.5.4. Map Class

Attributes:

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

public CoordinateTuple[] bridgeCoordinates: This attribute represents the tuples for built bridges.

final int ROW_NUMBER =9: This attribute represents the number of row spaces for map.

final int COLUMN_NUMBER = 13: This attribute represents the number of column spaces for map.

ArrayList<Space> visited: This attribute represents the space list that are visited.

Methods:

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

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.

public ArrayList<Space> bridgeables(Space space1): This method is to get spaces that bridges can be built.

4.5.5. Player Class

Attributes:

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

private 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 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 key: This attribute represents the number of town keys player earned from founding town.

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

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

private int townScore: This attribute represents the progress of player for founding 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 freeSpade: This attribute represents the number of free spades.

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.

Methods:

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

public boolean spendFreeSpade(String terrainType): This method is to spend free spades for terraforming.

4.5.6. PlayerHandler Class

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

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

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

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

public boolean buildStructure(Player player): This method is to upgrade a dwelling to,

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

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

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

public void terraform(Player player): This method is to terraform a landscape.

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

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

public boolean updateResources(Player player): This method is to update the resources of players at the end of the rounds.

public boolean buildInitialDwelling(Player player): This method is to build initial dwellings before game round 1 starts.

public boolean townFound(Player player): This method is to update player variables after founding town.

4.5.7. 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.

Methods:

public void setInitials: This method is to set the initial values according to faction.

public void afterStronghold: This method is to have the special faction abilities after building stronghold.

4.5.8. 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 isKeyPlaced: 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.

4.5.9. ReligionController Class

Methods:

public void showChoices(Player[] playerArr, Religion[] religionArr): Shows and updates players' choices related to religions.

public void showReligions(Player[] playerArr, int status, Religion[] religionArr): Shows the current status of the religion diagram.

4.5.10. 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.

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.

4.5.11. CardsAndTilesController Class

Attributes:

public CardsAndTiles cardsAndTiles: This attribute holds CardsAndTiles object.

Methods:

public void showTownTiles(int playerId, boolean canChoose): It controls to show town tiles.

public void showScoringTiles(int playerId, boolean canChoose): It controls to show scoring tiles.

public void showFavorTiles(int playerId, boolean canChoose): It controls to show favor tiles.

public void showBonusCards(int playerId, boolean canChoose): It controls to show bonus cards.

public void chooseTownTile(int playerId): It control that player chooses town tiles.

public void chooseScoringTile(int playerId): It control that player chooses scoring tiles.

public void chooseFavorTile(int playerId): It control that player chooses favor tiles.

public void chooseBonusCard(int playerId): It control that player chooses bonus card.

4.5.12. CardView Class

Attributes:

private final int DEFAULT_HEIGHT: It holds view's height as default.

private final int DEFAULT_WIDTH: It holds view's width as default.

private final int DEFAULT_MARGIN_TOP: It holds view's top of the margin as default.

private final int DEFAULT_MARGIN_RIGHT: It holds view's right of the margin as default.

private final int DEFAULT_MARGIN_BOTTOM: It holds view's bottom of the margin as default.

private final int DEFAULT_MARGIN_LEFT: It holds view's left of the margin as default.

private final int DEFAULT_SPECING: It holds view's specing as default.

private final int DEFAULT_PLAYER_SLOT_RADIUS: It holds view's player slot radius as default.

private VBox playerSlots: It holds players as vertical.

Methods:

public void defaulthConfigure(): It configures view as default.

public void add(Pane pane): It adds pane into view.

public void addPlayerSlots(int slowNumber): It adds player slots into view.

public void addPlayerToSlot(int index, Image playerImage): It adds player into slots.

public void setSize(int height, int width): It sets the size.

4.5.12.1. 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.

4.5.12.2. 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.

4.5.12.3. Scoring Tile Class

Attributes:

private int playerId: It holds player's id who takes this tile.

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.

4.5.12.4. 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.

4.5.13. 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.

Space[] [] bridgeList:

boolean hasBridge: This attribute show that whether this space contains bridge.

Structure structure: This attribute holds structure on the space.

Player player: This attribute holds the player whose structure is on the space.

Methods:

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

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

4.5.14. Structure Class

Attributes:

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

private String buildingType: This attribute holds the type of building such as Sanctuary, Stronghold, Temple, TradingPost and Dwelling.

Methods:

public Structure(String type): This is the constructor of structure class and sets building score according to given building type.

4.5.15. 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.

4.5.16. FileManager Class

Attributes:

private File save: This attribute is the save file of the game.

private GameController game: This attribute includes game-related contents to be saved.

private RoundController rounds: This attribute includes round-related contents to be saved.

Methods:

public FileManager(String gameId): This method is the constructor method of File Manager.

public File createSave(String gameId): This method creates and returns a save file with game id.

public void saveGame(GameController game, RoundController rounds): This method uses serializaton in order to save required objects to the save file.

public void loadGame(): This method deserializes objects in the file and assigns them to related variables.

4.5.17. LoadGame Controller Class

Attributes:

privateVbox gameSlots: This attribute includes the game slots to be loaded.

private Button loadSlot1: This attribute represents a game slot to be loaded.

private Button loadSlot2: This attribute represents a game slot to be loaded.

private Button loadSlot3: This attribute represents a game slot to be loaded.
private Button loadSlot4: This attribute represents a game slot to be loaded.

Methods:
public void loadSlotClicked(MouseEvent event): This method determines which game slot to be loaded.

4.5.18. RoundController Class

Methods:
public int endTurn(int curPlayerId, int[] playerList): This method ends the turn of a player.
public void endRound(): This method terminates the round.
public boolean isRoundEnd(): Returns whether the round is completed or not.
public void setupGame(): This method setups the game.

4.5.19. SoundController Class

Attributes:

private ArrayList<Media> audios: This list represents the soundtrack of the game.

Methods:
public void setVolume(int volume): This method sets the volume of the music
public void play(): This method plays the music.

4.5.20. ActionController Class

Methods:
public void terraform(Player[] playerArr, int curPlayerId): This method enables terraform action for player with specific ID.
public void upgradeStructure(Player[] playerArr, int curPlayerId, Button[][] terrains, Map map): This method enables Upgrade Structure action for player with specific ID.
public void upgradeToTradingPost(Player[] playerArr, int curPlayerId, Button[][] terrains, Button terrain, Map map, Space space): This method enables upgrade to trading post action for player with specific ID.
public void upgradeToStrongholdOrTemple(Player[] playerArr, int curPlayerId, Button[][] terrains, Button terrain, Map map, Space space): This method enables upgrade to stronghold or temple action for player with specific ID.
public void upgradeToSanctuary(Player[] playerArr, int curPlayerId, Button[][] terrains, Button terrain, Map map, Space space): This method enables upgrade to sanctuary action for player with specific ID.

public void upgradeShipping(int curPlayerId, Player[] playerArr): This method enables upgrade shipping level for player with specific ID.

public void sendPriest(Player[] playerArr, int curPlayerId): This method enables sending priest option for player

public void usePowerAction(int curPlayerId, Player[] playerArr): This method enables power actions for player with specific ID.

public void useSpecialAction(int curPlayerId, Player[] playerArr): This method enables special actionfor player with specific ID.

public void exchangeResources(int curPlayerId, Player[] playerArr): This method enables exchanging resources level for player with specific ID.

public void upgradeSpade(int curPlayerId, Player[] playerArr): This method enables upgrade spadelevel for player with specific ID.

public void skipTurn(int curPlayerId, Player[] playerArr): This method enables skip turn for player with specific ID.

4.5.21. GameController Class

Attributes:

private Map map:

private Player[] playerList: Player array that holds all the players

private Button[][] terrains: terrain buttons that are displayed in the game map

private CardsAndTiles cardsAndTiles: instances of Cards and Tiles

private ArrayList<PlayerView> playerViewList: PlayerView array list that holds the views of the players

private Religion[] religions: Religion array that holds all the religions

Methods:

public void chooseAction(int choice): This method will link the player with the chosen action

public void createSpaces(): This method create spaces for the model of the map

public void loadPlayer(ArrayList<Faction> factionList, ArrayList<String> playerNames): This method load players in the beggining of the game with their corresponding factions.

public void initialize(URL url, ResourceBundle resourceBundle): This method initializes the GameController class

public void loadInitialMap(): This method loads the map at the beginning of the game

public void skipRoundClicked(): This methods skips the round of current player

public void upgradeShipping(): This method upgrades the shipping of the current player

public void sendPriest(): This method send priest for the current player

public void powerActionClicked(): This method use power actions for the current player

public void upgradeSpadeClicked(): This method upgrade spade levels for the current player

public void terraformClicked(): This method terraforms a terrain tile for the current player

public void upgradeStructureClicked(): This method upgrade a structure for the current player

public void religionsClicked(): This method shows the Religion tab

public void scoreTableClicked(): This method shows score table of the game

public void exchangeResources(): This method opens the exchange resources tab

public void bonusCardsClicked(): This method shows the bonus cards

public void townTilesClicked(): This method shows the town tiles

public void scoringTilesClicked(): This methods shows the scoring tiles

public void favorTilesClicked(): This method shows the favor tiles

public void speacialActionClicked(): This method shows the special actions that player can use

public void skipTurn(): This method skip turns for current player.

4.5.22. InGameMenuController Class

Attributes:

public Slider volumeSlider: This slider adjusts the volume.

public Button setButton: This buttons applies the settings.

public ChoiceBox resolution: Users can choose preferred resolution with this box.

Methods:

public void setVolume(int volume): This method sets the preferred volume level.

public void setResolution(int x, int y): This method sets the preferred resolution level.

4.5.23. DialogueView Class

Methods:

public Stage getStage(String title, Parent component, Image backgroundImage): This method returns the stage with specific condition and components

public BorderPane getDwellingUpgradePromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when upgrading the dwelling

public BorderPane getTradingPostUpgradePromptPane(Button yesButton, Button noButton, RadioButton templeButton, RadioButton strongholdButton): This method returns a Border Pane that will be represented when upgrading the trading post

public BorderPane getTempleUpgradePromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when upgrading the temple

public BorderPane getTerraformPromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when terraforming a terrain tile

public BorderPane getUpgradeShippingPromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when upgrading the shipping level

public BorderPane getUpgradeSpadePromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when upgrading the spade

public BorderPane getSendPriestPromptPane(Boolean isOrderChoice, String yesButton, String noButton): This method returns a Border Pane that will be represented when sending a priest

public BorderPane getPowerActionsPromptPane(String yesButton, String noButton, String actionButtons): This method returns a Border Pane that will be represented when using the power actions

public BorderPane getSpecialActionPromptPane(String yesButton, String noButton): This method returns a Border Pane that will be represented when using the special actions

public BorderPane getRoundEndPromptPane(): This method returns a Border Pane when the round ends

public BorderPane errorCannotTerraform(): This method returns a Border Pane with a error message when the user cannot terraform

public BorderPane errorNotEnoughResources(): This method returns a Border Pane with a error message when the user does not have enough resources

public BorderPane errorReachedMaxBuilding(): This method returns a Border Pane with a error message when the user reaches the max building

public BorderPane errorReachedMaxShippingLevel(): This method returns a Border Pane with a error message when the user reaches the max shipping level

public BorderPane errorReachedMaxSpadeLevel(): This method returns a Border Pane with a error message when the user reaches the max spade level

public BorderPane errorNotEnoughPriest(): This method returns a Border Pane with a error message when the user does not have enough priest

public BorderPane errorNoSpecialAction(): This method returns a Border Pane with a error message when the user does not have any special actions

5. Improvement Summary

In the first iteration, we designed our subsystem decomposition very much like model-view-controller. In the light of our feedback, we updated our subsystem composition that now it uses model-view-controller architecture rather than model-view-controller subsystem. Also, we updated our boundary conditions by including load game and save game boundaries. For our design trade-offs, there were some conflicts between trade-offs and design goals in terms of maintainability and usability. We updated them accordingly to the our design goals and conflicts between trade-offs and design goals are resolved. For final object design, our view(UI) classes were not included for iteration 1. For this iteration, we added our view(UI) classes to the final object design. Finally, we added other missing attributes and methods to our final object design that data-flow and consistency between them has been improved radically.

6. Glossary&References

- *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development*, by Craig Larman, Prentice Hall, 2004, ISBN: 0-13-148906-2
- *Object-Oriented Software Engineering*, by Timothy C. Lethbridge and Robert Laganiere, McGraw-Hill, 2001, ISBN: 0-07-709761-0
- *Developing Software with UML, Object-Oriented Analysis and Design in Practice*, by Bernd Oestereich, Addison-Wesley, 1999, QA76.9.03503713 1999.
- *Object-Oriented Analysis and Design with Applications, 2nd ed.*, by G. Booch, Benjamin/Cummings, Redwood City, CA, 1994, QA76.64.B66 1994.
- *Head First Design Patterns*, by Bert Bates et al, O'Reilly Media, 2009.
- “DoDAF, NAF and MODAF Read More,” *Ideal Modeling & Diagramming Tool for Agile Team Collaboration*. [Online]. Available: https://www.visual-paradigm.com/. [Accessed: 09-May-2020].
- “Developer Tools for Professionals and Teams,” *JetBrains*. [Online]. Available: https://www.jetbrains.com/. [Accessed: 09-May-2020].