Bilkent University

Department of Computer Engineering

CS319 - Object-Oriented Software Engineering

Term Project - Design Report

Project Name: Settlers of Catan

Group No: 3G

Group Members: Mehmet Tolga Tomris
Atakan Arslan
Fatih Çakır
Oğuzhan Dere
Zeynep Berfin Gökalp

Table of Contents

1.Introduction

1.1 Purpose of The System

Settlers of Catan will be a turn-based strategy game, the main goal of the game is to collect 10 victory points by using resources to build the best combinations of roads, settlements or cities on the terrain hexes.

1.2. Design Goals

Maintainability

Game design will be based on object-oriented programming concepts which will allow creating the game in an extensible way. The subsystems will have a hierarchical structure in order to add new features easily without huge changes in upper systems. In addition, code will be written in a way that can be easily understood by others.

Performance

Players while playing the game, will not want to wait for a long time every time they perform a certain action as it is a turn-based game. Therefore, we will design the game in a way that a player's actions will result in less than 2 seconds on average.

.

User Friendliness

Settlers of Catan will have a user-friendly interface that will enable users to easily understand the game basics and controls. Gam object's icons and control icons will have similarities with the board game for easy understanding and memorizing. The interfaces and system should be clear that the game will be enjoyable and the user should not need any extra help.

Robustness

In order to entertain and please the users, the game should not have major bugs that will disrupt the flow of the game. The system should handle errors such as invalid actions by limiting user actions with the game through user-interface and handling exceptions in a safe manner.

1.2.1. Trade-Offs

Cost vs Portability

Having limited time, the game will be only implemented for desktop operating systems that run java. Therefore, users will not be able to play the game on mobile platforms.

Memory vs Performance

Since the game is designed with object-oriented principles, memory usage is not a huge concern. That allows gaining run time efficiency in the design.

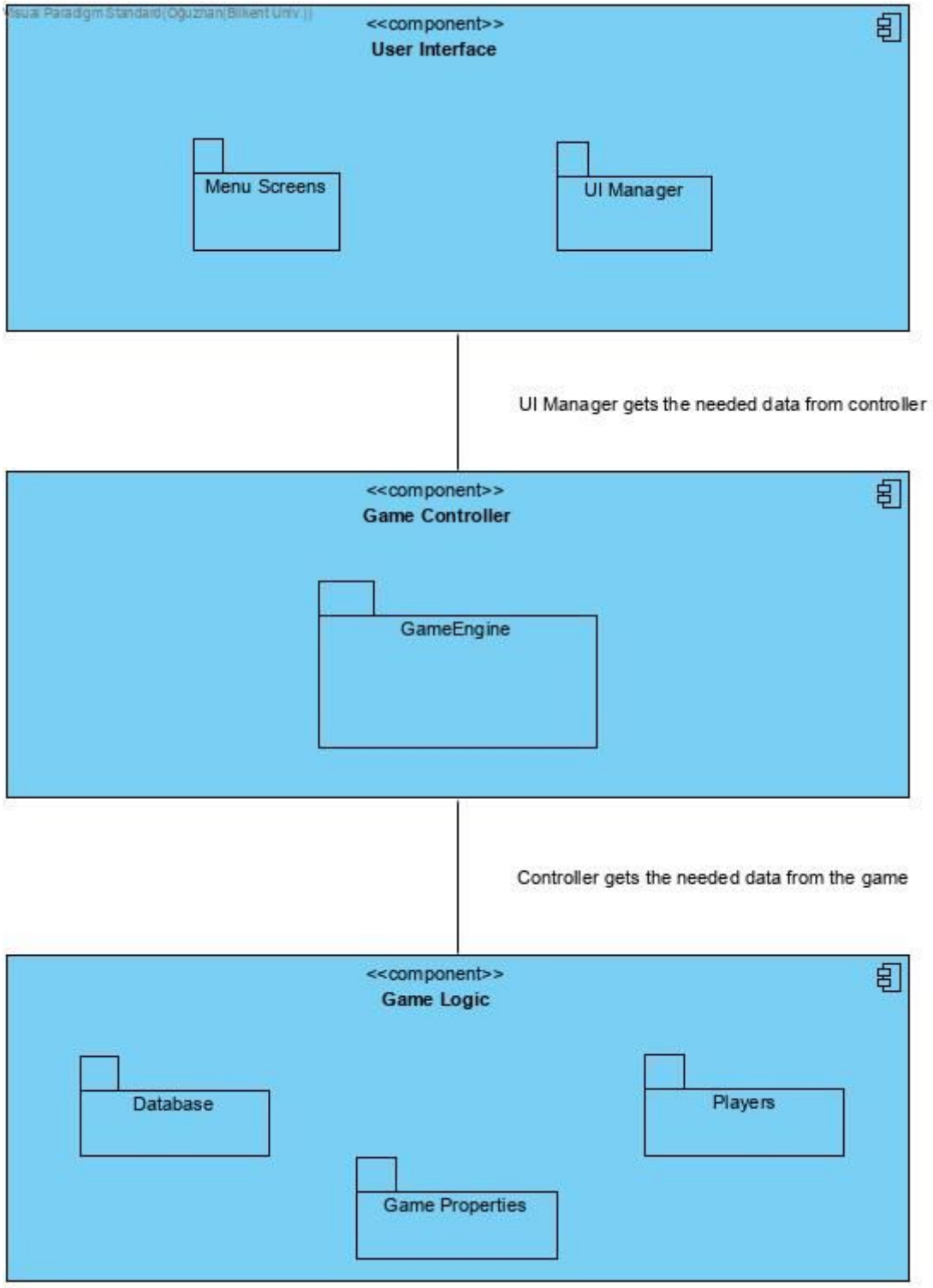
Functionality vs Robustness

Being a game turn-based game and various actions and modes, it is easy for user to try invalid actions. By carefully detecting and handling exceptions, the robustness code will be maintained, too.

2.System Architecture

2.1 Subsystem Decomposition

The system architecture can be thought of as it is divided into three main subsystems. One of these main subsystems is the user interface. The user interface will gather the interface classes such as menu screens and UI Manager. The user interface is the subsystem that will have a connection with the Game Controller subsystem, in order to be provided data. The game Controller subsystem includes the GameEngine Facade class. A detailed description of the Facade Design was covered in section 4.1.1. The data that the User-Interface subsystem gathers from the Game Controller subsystem, will be flowing through the Game Logic subsystem. This subsystem will include the database in it too. Therefore, it can be considered as a fusion of two subsystems: storage and game. Combining these two subsystems into one created a more readable and easy-to-implement system architecture. The Game Logic subsystem is the subsystem that includes the most amount of classes. This subsystem will include packages that are related to storage and game properties. A visualization of the subsystems and their relationship is as follows:



As can be seen from the figure, the Game Logic subsystem has no relation with the user interface. A flow is apparent. Data flow goes from the Game Logic subsystem to the User Interface subsystem. Thus, because of the existence of the hierarchy between subsystems, the architectural style of the system can be considered as a 3-tier architectural style.

2.2 Hardware / Software Mapping

As it is indicated in the requirement analysis, the game is not a multi-user game, which means that it will not use any servers or any external hardware components. Ordinary computer hardware such as processor and memory will be used in the software.

2.3 Persistent Data Management

The game will be implemented so that the user has the option to save the game before s/he exists in the game. In order to provide this functionality, a mechanism that stores the necessary values is needed. A database in the system will be present in order to be able to save the values and keep the game information out of the RAM of the computer too. If the database gets any unexpected condition (such as deletion or unexpected values), it will be modified (or created if necessary) to its initial state in order to prevent any data flow error.

2.4 Access Control And Security

There will be no effort of preservation of the data from other users in the game since there are no other users in the game. There will be only 3 AI players, other than the user. Game will provide its own data through its own database. Therefore, the database will be manually unreachable and secured. All the system will be controlled through a single subsystem, that is a controller. The controller subsystem will prevent any unwanted access of the user and it will be playing the role of a wall between user and core game classes.

2.5 Boundary Conditions

2.5.1 Initialization

For the initial start-up, the game will show the main menu to the user via user interface classes. There is no data needed in order to provide the menu interfaces. However, the game and user interface will obviously use the database in order to be playable. It is important to note that the executable file of the game will open the game and show its menu interface to the user, even if another instance of the game is already running in the background. However, when the user tries to start or load a game, the game will check from the database if it is already running or not. If it is already running, the game will not let the user start or load a game, since the game is already played at that moment.

2.5.2 Termination

When termination is performed in an unusual way, such as closing the game from the right top of its window or closing the computer all of a sudden, the termination will be done without saving. If the buttons are used as expected, the program will save the game to the database.

2.5.3 Failure

The game will try to avoid any error that causes the interface to collapse. Therefore, a database will be always present, in order to prevent total failure. Most of the possible bug expectations will be in the user interface subsystem. The user interface subsystem has a higher chance of functioning as it is not intended.

3. Subsystem Services

3.1 User Interface Subsystem

This component is to display the user interface and menu screens. It will have an interaction with the Game Controller System. It has two subcomponents: the UI Manager and Menu Screens.

3.1.1 UI Manager

This component handles the display of the game environment according to the selected mode of the game. Also, it handles the flow between menu screens and flow during the game.

3.1.2 Menu Screens

This component's responsibility is to display menu screens.

3.2 Game Controller Subsystem

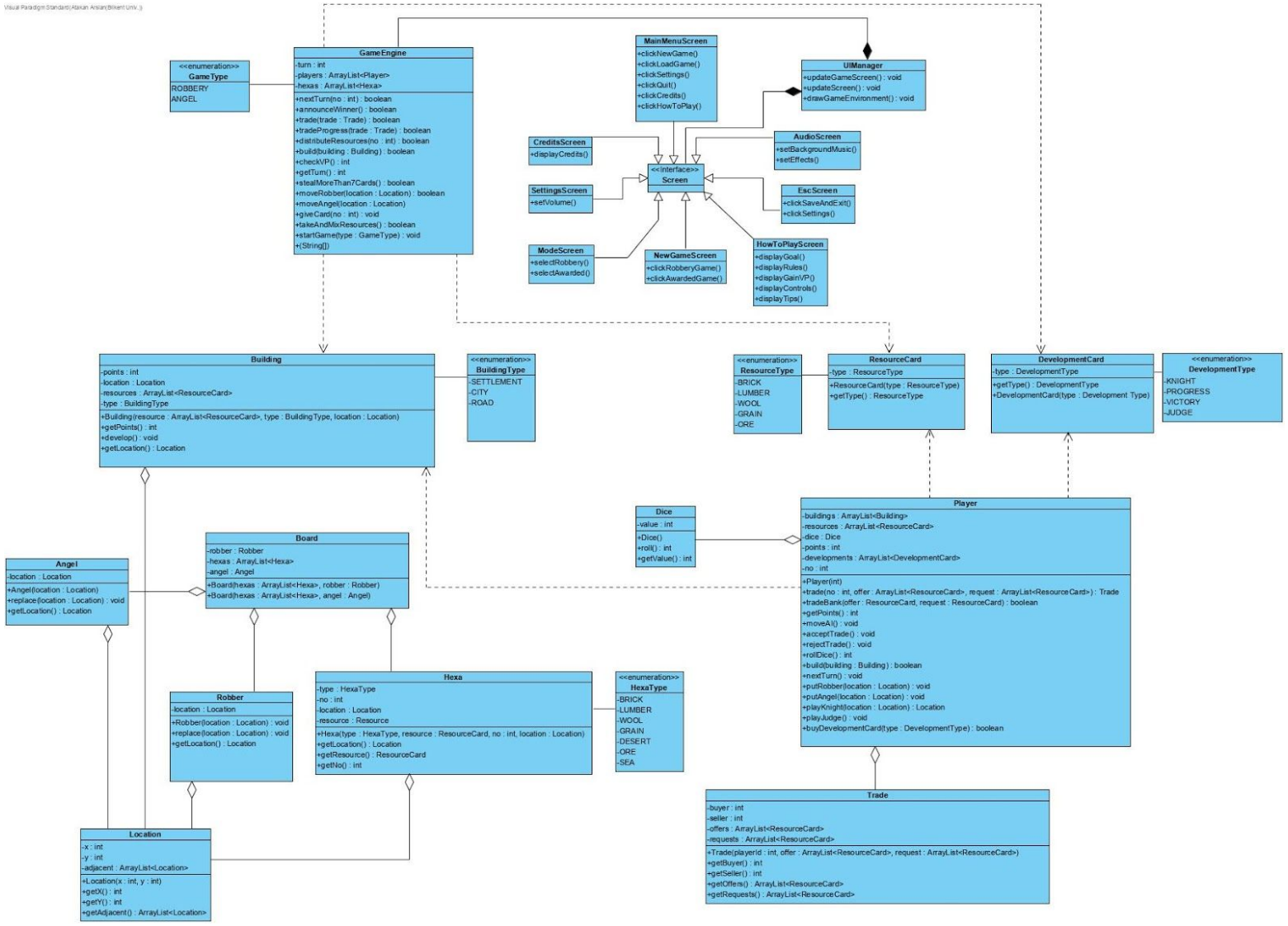
This component basically controls the gameplay and includes actions. Furthermore, it creates the game universe and updates the game in relation to the player's moves in every turn. It provides data to User Interface to carry out game logic.

3.3 Game Logic Subsystem

This component is responsible for game properties and objects. It is also responsible for getting the data from the database and retrieve the data and return it to the Game Controller Subsystem accordingly. It will communicate with subcomponents for relevant services.

4. Low-Level Design

4.1 Object Design

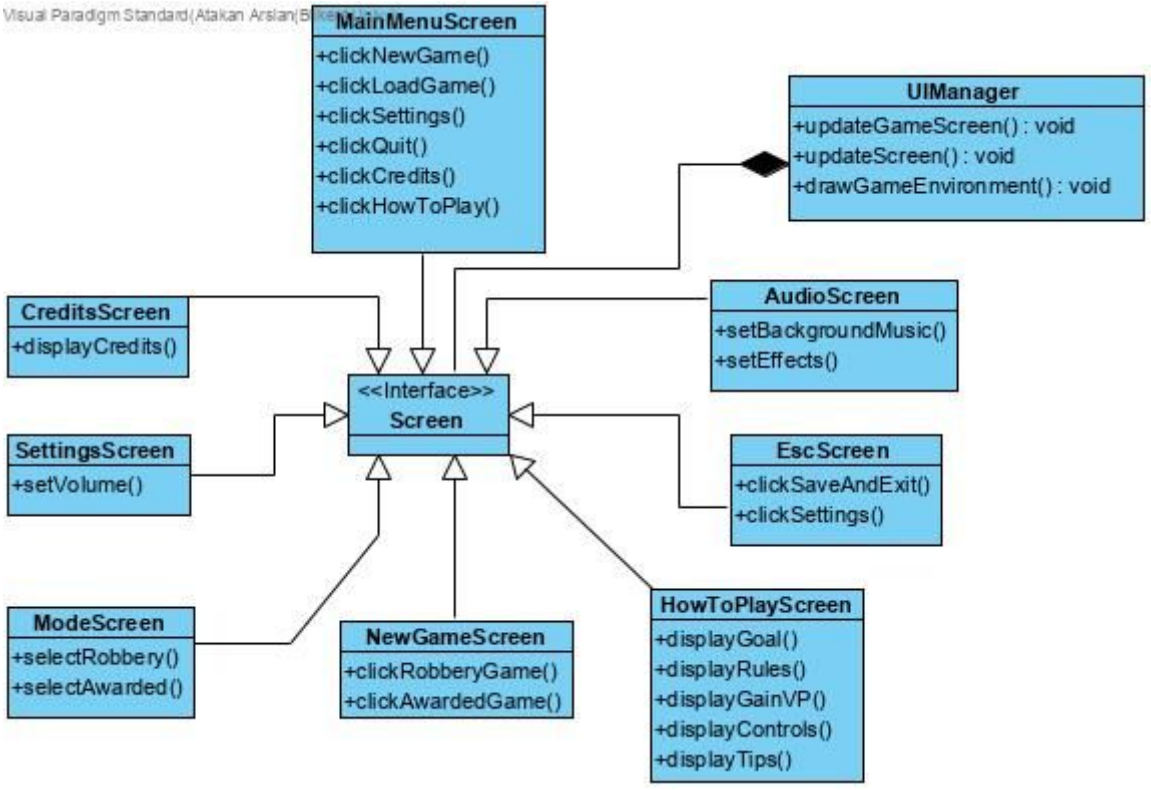


4.1.1 Facade Design

In our game, there is a GameEngine class that acts as a facade class. “Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to the existing systems to hide its complexities.”
[https://www.tutorialspoint.com/design_pattern/facade_pattern.htm]

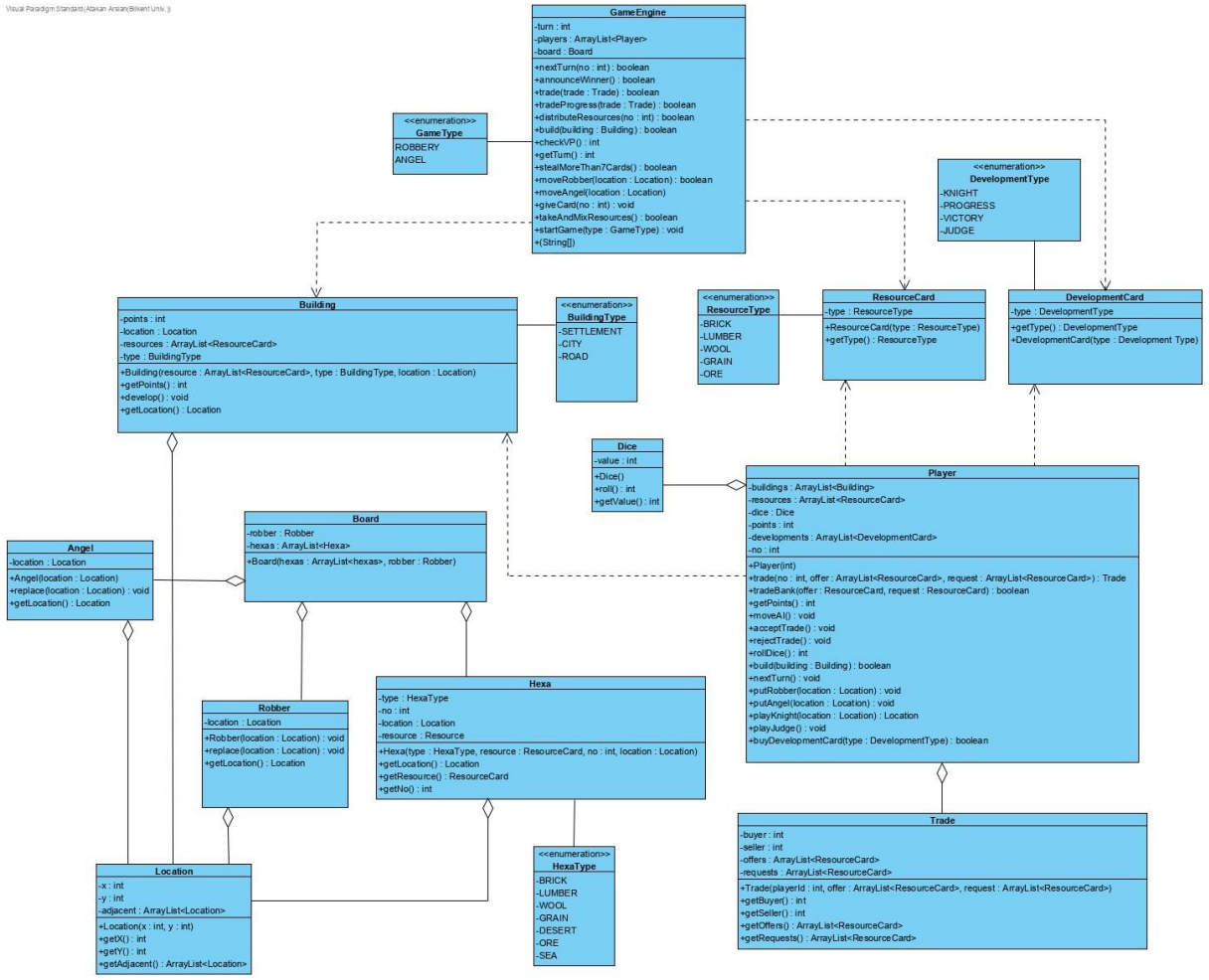
4.2 Layers

4.2.1 User Interface Layer



This layer is a boundary object between player and game logic layer.

4.2.2 Game Logic Layer



This layer is associated with the logical parts of our Catan game. GameEngine class controls all actions in the game. Also, it provides data about the process of the game for the User Interface. This layer creates the game universe and updates the game according to the moves of players in every turn. In addition, it checks whether the game is over or not.

4.3 Class Interfaces

4.3.1 User Interface Layer Class Interfaces

4.3.1.1 MainMenuScreen Class

Operations:

clickNewGame(): Enables player to go to the new game screen.

clickLoadGame(): Enables player to load the latest previously played game.

clickSettings(): Enables player to go the settings screen.

clickQuit(): Enables player to quit the game.

clickCredits(): Enables player to go the credits screen

clickHowToPlay(): Enables player to go the how to play screen.

4.3.1.2 CreditsScreen Class

Operations:

displayCredits(): Enables player to display the credits of the game.

4.3.1.3 SettingsScreen Class

Operations:

setVolume(): Enables player to set the volume.

4.3.1.4 ModeScreen Class

Operations:

selectRobbery(): Enables player to select the Robber Mode.

selectAwarded(): Enables player to select the Angel Mode.

4.3.1.5 NewGameScreen Class

Operations:

clickRobberyGame(): Commands GameEngine class to start up the Robbery Mode.

clickAwardedGame(): Commands GameEngine class to start up the Angel Mode.

4.3.1.6 AudioScreen Class

Operations:

setBackgroundMusic(): Enables player to set the background music.

setEffects(): Enables player to set the effects of the game.

4.3.1.7 ESCScreen Class

Operations:

clickSaveAndExit(): Enables player to save the game and exit.

clickSettings(): Enables player to go to settings.

4.3.1.8 HowToPlayScreen Class

Operations:

displayGoal(): Displays the main goal of the game.

displayRules(): Displays the rules of the game.

displayGainVP(): Explains how to gain victory points in the game.

displayControls(): Displays the controls of the game.

displayTips(): Displays the tips of the game.

4.3.1.9 UIManager Class

Operations:

updateGameScreen(): Provides the flow during the game in accordance with the updated data.

updateScreen(): Provides the flow between menu screens.

drawGameEnvironment(): Creates the whole game environment which includes the map, the board, players, cards, buildings and robber or angel according to the chosen game mode.

4.3.2 Game Logic Layer Class Interfaces

4.3.2.1 GameEngine - Facade Class

Attributes:

turn: Holds the current turn.

players: Represents the players of the game in an array.

hexas: Represents the hexas which creates the map of the game in an array.

Operations:

nextTurn(int): Determines which player will play that turn.

announceWinner(): Announces the winner of the game.

trade(Trade): Sends the player the trade which is created by the proposer. The game engine directs the trade to other players with this method.

tradeProgress(Trade): Exchanges the cards after the trade is accepted.

distributeResources(int): Distributes resource cards to players according to dice's result.

build(Building): Place the building into the map and returns whether placed or not.

checkVP(): Checks victory points of players to determine the winner.

getTurn(): Returns the current turn.

stealMoreThan7Cards(): Takes resource cards from players if they have more than 7 cards. This method is specific for Robber Mode.

moveRobber(Location): Moves robber to a selected position on the map. This method is specific for Robber Mode.

moveAngel(Location): Moves angel to a selected position on the map. This method is specific for Awarded Mode.

giveCard(int): Gives random resource cards to player who rolls 7 in Awarded Mode.

takeAndMixResources(): Takes all resource cards, mix and distribute to all players. This method is Judge Card's operation.

startGame(GameType): Starts the game according to its type.

4.3.2.2 Building Class

Attributes:

points: The building's point. (Road: 0, Settlement : 1, City: 2)

location: The location of the building.

resources: The resources of the building.

type: Type of the building which can be settlement, city or road. This is enumeration.

Operations:

Building(ArrayList<ResourceCard>, BuildingType, Location): Constructor of the building.

getPoints(): Returns point of the building.

develop(): Develops the settlement to city.

getLocation(): Returns the location of the building.

4.3.2.3 BuildingType (Enumeration)

Attributes: SETTLEMENT, CITY, ROAD

4.3.2.4 Location Class

Attributes:

x: The value of the game object in the x-direction.

y: The value of the game object in the y-direction.

adjacents**:** The adjacent locations of the game object.

Operations:

Location(int, int): Constructor of location object. It takes x and y values as parameters.

getX(): Returns x.

getY(): Returns y.

getAdjacents(): Returns adjacents.

4.3.2.5 Angel Class

Attributes:

location: The location of the angel.

Operations:

Angel(Location): Constructor of the angel object which takes a location as a parameter.

replace(Location): Changes the angel's location according to selected location.

getLocation(): Returns location of the angel.

4.3.2.6 Robber Class

Attributes:

location: The location of the Robber.

Operations:

Robber(Location): Constructor of the robber object. It takes its location as a parameter.

replace(Location): Changes the robber's location according to the selected location.

getLocation(): Returns location of the robber.

4.3.2.7 Trade Class

Attributes:

buyer: The offerer's player no.

seller: The offeree's player no.

offers: The resource cards which are offered by the buyer.

requests: The resource cards which are requested from the seller.

Operations:

Trade(int, int, ArrayList<ResourceCard>, ArrayList<ResourceCard>): Constructor of Trade object. It contains buyer and seller's player no and offered and requested resource cards.

getBuyer(): Returns buyer's player number.

getSeller(): Returns seller's player number.

getOffers(): Returns offered resource cards.

getRequests(): Returns requested resource cards.

4.3.2.8 Dice Class

Attributes:

value: The value of rolled 2 dice.

Operations:

Dice(): Constructor of Dice class.

roll(): Rolls dice and returns the total number of them.

getValue(): Returns the value of dice.

4.3.2.9 ResourceCard Class

Attributes:

type: Type of the resource card. (ResourceType enumeration)

Operations:

ResourceCard(ResourceType): Constructor of ResourceCard takes its type as a parameter.

getType(): Returns the type of the resource card.

4.3.2.10 DevelopmentCard Class

Attributes:

type: Type of the development card. (DevelomentType enumeration)

Operations:

DevelopmentCard(ResourceType): Constructor of DevelopmentCard takes its type as a parameter.

getType(): Returns the type of the development card.

4.3.2.11 ResourceType (Enumeration)

Attributes: BRICK, LUMBER, WOOL, GRAIN, ORE

4.3.2.12 DevelopmentType (Enumeration)

Attributes: KNIGHT, PROGRESS, VICTORY, JUDGE

4.3.2.13 Board Class

Attributes:

robber: The robber on the board of the game

angel: The angel on the board of the game.

hexas: The hexas on the board of the game.

Operations:

Board(ArrayList<Hexa>, Robber): Constructor of the board which takes the list of hexas and the robber as parameters. This is used for the Robber Mode.

Board(ArrayList<Hexa>, Angel): Constructor of the board which takes the list of hexas and the angel as parameters. This is used for the Angel Mode.

4.3.2.14 Player Class

Attributes:

buildings: The list of the buildings that a player has.

resources: The list of resource cards that a player has.

dice: An instance of the dice class.

points: The number of victory points that a player has.

developments: The list of the development cards that a player has.

no: The number of the player.

Operations:

Player(int): The constructor of the Player which takes the player's number as a parameter

trade(int, ArrayList<ResourceCard>, ArrayList<ResourceCard>): It enables the player to offer a trade to the other players.

tradeBank(ResourceCard, ResourceCard): Enables player to trade with the bank of the game.

getPoints(): int

moveAI(): Controls the movements of non-player characters. (bots)

acceptTrade(): Enables player to accept the offered trade.

rejectTrade(): Enables player to reject the offered trade.

build(Building): Enables player to put a building on the map. It takes building as a parameter.

rollDice(): Calls the roll method of Dice class.

nextTurn(): Gives turn to the next player.

putRobber(Location): Enables player to put the robber any place on the map that he or she wants.

putAngel(Location): Enables player to put the angel any place on the map that he or she wants.

playKnight(Location): Calls the put robber function and player that uses the knight card can change the place of the robber.

playJudge(): All cards mixed up and distributed randomly to the users.

buyDevelopmentCard(DevelopmentType): Enables player to buy a development card. It takes the type of the development card as a parameter.

4.3.2.15 Hexa Class

Attributes:

type: The type of the hexa. (HexaType enumeration)

no: The number of the hexa.

location: Location of the hexa.

resource: The resource that the hexa includes on it.

Operations:

Hexa(HexaType, ResourceCard, int, Location): Constructor of the hexa which takes parameters HexaType, ResourceCard, int, and location.

getLocation(): Returns the location of the hexa.

getResource(): ResourceCard

getNo(): Returns the number of the hexa.

4.3.2.16 HexaType (Enumeration)

Attributes: BRICK, LUMBER, WOOL, GRAIN, DESERT, ORE, SEA

5. References

1. “The official website for the world of CATAN,” Catan.com | The official website forthe world of CATAN. [Online]. Available: https://www.catan.com/. [Accessed: 25-Oct-2019].