Bilkent University
Department of Compter Engineering

Object-Oriented Software Engineering Project

CS 319 Project: Conquest

Design Report

Ali Onur Geven, Furkan Usta, Furkan Taşkale,Halil Bülent Orhon

Instructor: Uğur Doğrusöz

TA: İstemi Bahçeci

Table of Contents

Table Of Figures

Analysis Report

Object Oriented Software Engineering: Conquest

1 Introduction

Conquest is a turn based local multiplayer strategy game. Players control their own territories and troops in turns to conquer other people's territories. Player that conquers all the territories wins the game. Players are able to upgrade and extend their troops to get upper hand in the battle.

Conquest will be played on the World Map and the main goal of the game is to be the Conqueror of the World. As the game starts areas on the map will be distributed randomly to each player. Each player will have different kinds of troops and every troop will have its own attributes. To be able to conquer the world players will have chances to create, upgrade troops and also attack to their neighbor areas. Players will gain gold for winning wars and they will spend this gold on upgrades and creation of troops. In each turn there will be limited number of upgrades and time spent. Whoever conquers the world will be the winner of the game.

Figure 1 Opening Menu of Conquest

2 Proposed System

2.1 Overview

2.1.1 Gameplay and Control

At the beginning of the game each region in the map will be distributed to the players; and each player will start with a number of troops that can be placed. The aim of each player the be the conqueror of the world by capturing all the regions. Throughout their conquests they will be assigned to some missions, and also will be able to upgrade their troops. At the start of each turn every player will get extra troops that can be placed. Although, at the beginning each player gets the same amount of extra units, later on some player might gain advantages and get more extra troops.

Main control device is mouse. Players will be able to click one of their regions and either attack to other enemy territories or move their troops to their another regions. In both action, map's color will change to show available options, also users will be able to select how many troops they want to use.

2.1.2Map

The game will be played on a world map divided into regions, instead of countries, to distribute the game over the whole map.

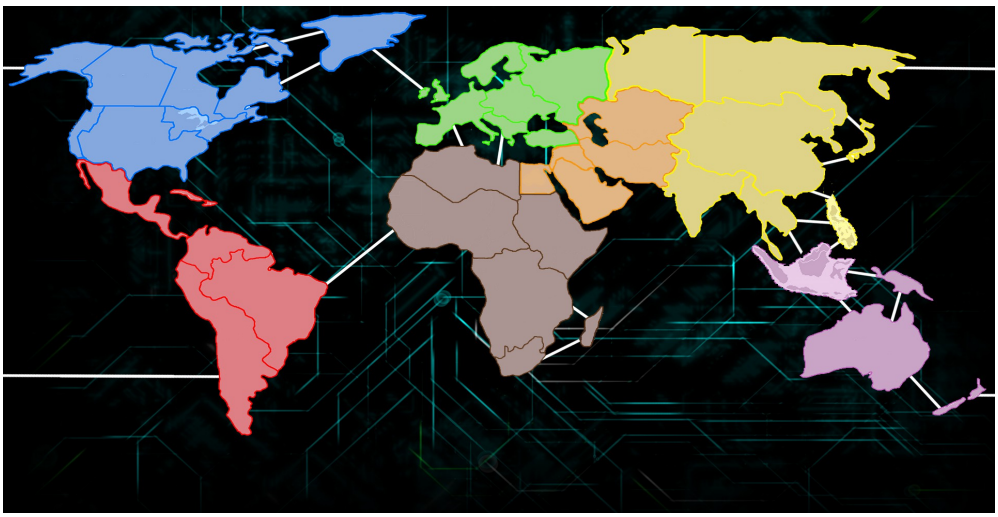


Figure 2 World Map that Conquest will be played on[6]

2.1.2.1 Continent

Continents are valuable if a user manages to capture all the regions in a continent. Depending on the number of regions in the continent and connections to the other regions, that user will gain extra troops at the beginning of each turn.

2.1.2.2 Region

It is only possible to move from a region to another region if they are sharing a border. However, some regions will have ferries that enables continents to be connected by allowing attacks and reinforcements over the sea.

2.1.3 Troops

There are 3 kind of troops in the game, every unit will have an attack speed, attack power, defense power and upgrades.

Infantry

Infantries are the starting units, and they have the lowest attack power and health among all the units. However, they have faster attack speed than tanks.

Figure 3 Infantry[3]

Tank

Tanks have the highest defense power but they have lower attack power than jets, and lower attack speed than troops.

Figure 4 Tank[4]

Jet

Jets have the highest attack power but they are in the middle for both defense power and attack speed.

Figure 5 Jet[5]

2.1.4 Troop Reinforcements

Other than attacking to the neighbor regions, users can move their troops to along their regions to reinforce. Depending on the distance between regions sending reinforcements may take more than a turn, and throughout their travel those unit will not be considered as part of the region they are moving along. Thus they, will not join any attacks or defense, they will also continue their travel even if the connection between the source and destination is disrupted by an another player.

Every reinforcement will take at least a turn. That means, a user cannot gather all the neighbor troops to one region and attack from there in one turn. These reinforcements will appear at the end of the user's turn. Moreover, users can choose how many units they want to send but at least one of the units has to stay in the region to protect it.

2.1.5 Upgrade and Experience System

At each attack, winning party gains experience points, depending on the casualties. Those points can be used to upgrade troops. Available upgrades of troops can be found in the troop tree which help player to plan on upgrades. Upgrades may increase attack speed, attack power or defense power. Upgrades to a specific type will be applied to all the present units of that type as well. In their turn users can both attack and upgrade; however, all the upgrades become effective when the turn ends.

2.1.6 Attacking

A user can attack to an another region if they are sharing a border or they are connected by ferries. User can choose as many units to attack but at least one unit has to stay in the region. In other words, user cannot abandon his/her region.

An attacking session lasts until there are no more units to fight in one of the parties. A session may consist of several turns (these turns are different than users') at each turn all the units in each side will be matched randomly, and according their stats battle will be simulated. These turns will be repeated until the war is over. If the last match of the last turn is about the end in a draw, then the defender is counted as winner, and his last unit does not die.

2.1.7 Scoring

Scoring system is used for deciding the winner of the game. There will be a leader board which keeps the player in order with respect to their scores. Players will gain points for conquering areas and gaining troops. For each upgrade of the troops there will be scores that players can gain. Every player will start form 0 point. Some of the possible outcomes of the game:

- **Player won with conquering the World** : When all the areas are conquered by one of the players, the game will be done and the player will be the winner of the game.
- **Player surrendered** : When a player surrenders, he will be removed from the game and he will take the last place in the leader board even if that players score is more than another player.
- **Player won with highest score in leader board :** When all the players agreed the end game vote, the game will be finished and the player with the highest score will be the winner of the game.

2.1.8 Achievements

Achievements system will be used for granting bonuses throughout the game. Achievements are as follows:

- When a player conquers all of the areas in a continent, the player will be granted an extra numbers of troops.

2.2 Functional Requirements

2.2.1 Play Game

- The Players should be able to select an area.
- The Players should attack from an selected area.
- The Players should be able to select the area that will be attacked.

- The System should inform the Players about the numbers of troops in selected area.
- The System should inform the Players about the types, abilities of troops in selected area.
- The System should color areas according to the Players colors who is the master of the area.
- The System should place the map on the screen.
- The System should display battle logs on the screen when a battle ends.
- The System should display the leader board on the screen.
- The System should change the color of the areas after its master has been changed.
- The System should update the scores of players whenever they are changed.
- The System should show the time on the screen.
- The System should update the time on the screen in each second
- The System should inform the Players after scores changed.
- The System should inform the Players when an achievement is unlocked.
- The System should inform the Players when an area is lost or conquered.
- The System should inform the Players when the game has ended.
- The System should inform the Players about their placements in leader board at the end of the game.

Figure 6 Main Menu of Conquest

2.2.2 Pause Game

- The Player should be able to pause the game while it is in the Player's turn.
- The System should pause the game, stopping all movements and time when the pause button is pressed.
- The System should display the Pause Menu when the game is paused.
- The Player should be able to continue the game pressing the continue button in the pause menu.

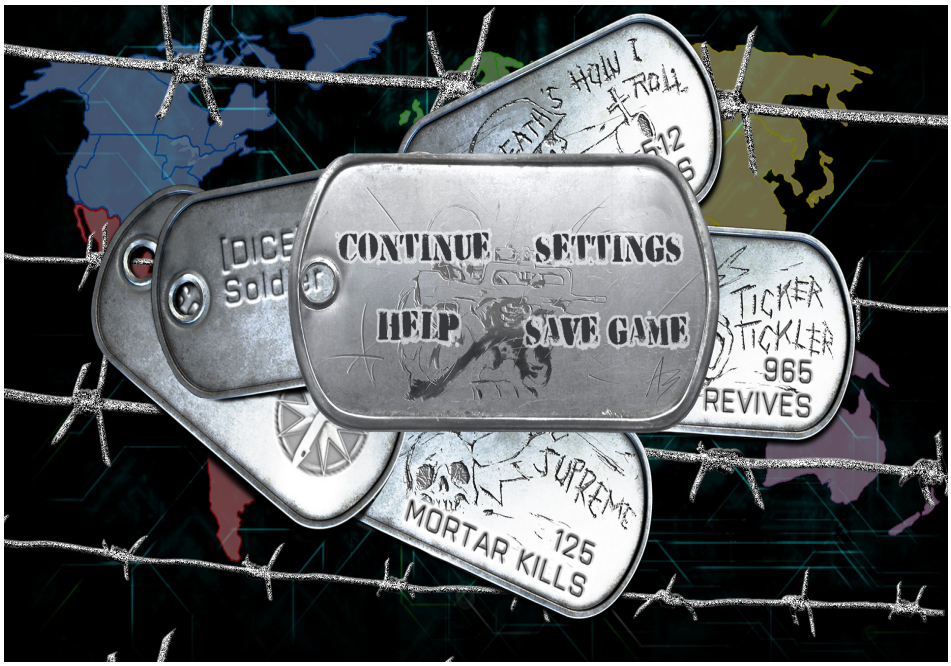


Figure 7 Pause Menu of Conquest

2.2.3 Save Game

- The Player should be able to save the game whenever the game is paused.
- The Player should be able to click the Save Game Button in the Pause Menu.
- The System should be able to save the areas, players, troops, time and achievements in the game whenever the Save Game Button is pressed while the game is paused.

2.2.4 Load Game

- Player should be able to load the game whenever he wants.
- Player should be able to click the Load Game Button in the Pause Menu.
- The system should be able to load the areas, players, troops, time and achievements in the game whenever the Load Game Button is pressed and Player has chosen which save he wants to load.

2.2.5 Upgrade Troops

- Player should be able to upgrade his troops.
- System should display the effects of an upgrade.
- System should block the player if player does not have enough experience to unlock an upgrade.

2.2.6 View and Change Preferences

- Player must be able to change his color which will be shown on the map.
- Player must be able to change his user name during the game.
- System should display the current preferences to Player.
- Player must be able to mute or unmute the system.

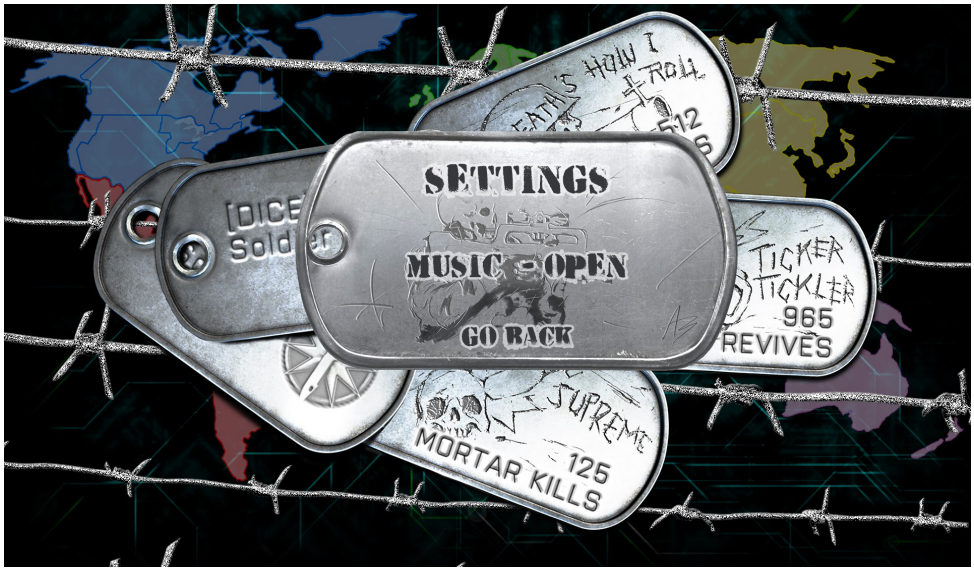


Figure 8 Settings Menu of Conquest

2.2.7 View Help

- The System must explain the gameplay controls to Player.
- The System must explain the upgrades of the troops to Player.
- The System must explain the basic concepts of the game to Player.

2.2.8 View Credits

- The System should be able to show credits.
- Player should be able to see credits menu during the game.

2.2.9 Play Sound

- System should play a sound when the war is over.
- System should play a sound when player upgrades his troops.

2.2.10 View Troop Tree

- Player should be able to see the troop tree during the game.

2.3 Non-functional Requirements

2.3.1 Usability

- Game should be well designed for players to be able to play it on the same computer with ease.
- Menus, HUDs and such visuals must be clear for the players to understand them without confusion.

2.3.2 Performance

- Game should be able to boot and start quickly.
- Fight simulations should be quick to calculate.

2.3.3 Reliability

- Game should not lose the Player profiles and games last status in a power-loss situation.
- Game should not lose the Player profiles and games last status if the System crushes.

2.3.4 Supportability

- Game must be upgradable and extendable with DLCs and future updates.

2.4 Pseudo Requirements/ Constraints

- Game will be implemented in java.
- Adobe Photoshop will be used for map, troop and menu designs.

2.5 System Models

2.5.1 Scenarios

Scenario 1

Example of Use Case: PauseGameAndContinueGame

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game.

Exit Conditions:

- Player Ali is playing the game.

Flow of Events:

1. Player Ali presses Pause button
2. Conquest pauses the game
3. Conquest shows the Pause Menu to Player
4. Player Ali presses Continue to Game Button
5. Conquest let Ali and his opponent continue to game from where they left off

Scenario 2

Example of Use Case: PauseGameAndLearnAboutTheUpgrades

Actors: Player Veli

Entry Conditions:

- Player Veli is playing the game

Exit Conditions:

- Player Veli is playing the game

Flow of Events:

1. Player Veli presses the Pause Button
2. Conquest pauses the game and freezes the screen
3. Conquest shows the Pause Menu
4. Player Veli presses Help button from the Pause Menu
5. Conquest displays a text about the upgrades
6. Player Veli presses continue to game button
7. Conquest continues the game

Scenario 3

Example of Use Case: View Credits

Actors: Player Ali

Entry Conditions:

- Player Veli is on Main Screen

Exit Conditions:

- Player Veli is on Main Screen

Flow of Events:

1. Player Ali chooses Credits from the menu

2. Conquest displays a text about credits
3. Player Ali reads the credits
4. Player Ali presses Main Menu button
5. Conquest displays the Main Screen

Scenario 4

Example of Use Case: ChangePlayerColorDuringTheGame

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game

Exit Conditions:

- Player Ali is playing the game

Flow of Events:

1. Player Ali presses Settings Button
2. Conquest displays Settings Page
3. Player Ali sees the views for changing his color
4. Player Ali presses Apply button
5. Conquest updates the settings and changes color of Player Ali
6. Player Ali presses return to game button
7. Conquest continues the game from where Ali left off

Scenario 5

Example of Use Case: LoadGame

Actors: Player Ali

Entry Conditions:

- Player Ali is on Main Screen

Exit Conditions:

- Player Ali is playing the game

Flow of Events:

1. Player Ali presses Load Game Button
2. Conquest displays the selection menu with saved games

3. Player Ali selects one of the saved games
4. Conquest displays the game info
5. Player Ali selects an another saved game
6. Conqeust updates the game info
7. Player Ali selects Load Game
8. Conquest opens the selected game

Scenario 6

Example of Use Case: AttackAnotherUser

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game and his turn

Exit Conditions:

- Player Ali is playing the game and his turn has ended

Flow of Events:

1. Player Ali selects one of his regions
2. Conqeus displays Action Buttons (Transfer, Attack)
3. Player Ali selects Attack Button
4. Conquest displays the applicable regions
5. Player Ali selects an applicable region
6. Conqeust shows a menu to select number of units
7. Player Ali enters the number of units
8. Player Ali presses Attack Button
9. Conquest simulates the battle
10. Conqeust displays the result
11. Conquest updates the map

Scenario 7

Example of Use Case: TransforTroops

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game

Exit Conditions:

- Player Ali is playing the game

Flow of Events:

1. Player Ali selects one of his regions
2. Conqeus displays Action Buttons (Transfer, Attack)
3. Player Ali selects Transfer Button
4. Conquest displays the applicable regions
5. Player Ali selects an applicable region
6. Conqeust shows a menu to select number of units (and also displays required turns)
7. Player Ali enters the number of units
8. Player Ali presses Transfer Button
9. Conquest transfers the units when the required number of turns has passed

Scenario 8

Example of Use Case: GameEnds

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game

Exit Conditions:

- Player Ali is on Main Menu

Flow of Events:

1. Player Ali attacks last of his enemies' regions.
2. Player Ali wins the battle.
3. Conquest freezes the game and calculates statistics
4. Conqeust displays the End screen
5. Player Ali presses Quit Button
6. Player Ali returns to the Main Menu

Scenario 9

Example of Use Case: MuteVolume

Actors: Player Ali

Entry Conditions:

- Player Ali is playing the game

Exit Conditions:

- Player Ali is on Main Menu

Flow of Events:

1. Player Ali pauses the game
2. Conquest displays Pause screen
3. Player Ali presses the Options Button
4. Conquest displays Options Menu
5. Player Ali presses Mute Button
6. Conqeust mutes the sound
7. Player Ali presses Back Button (Still in the Pause Menu)
8. Player Ali presses Resume Button
9. Conquest closes the Pause Menu and continues the Game

2.5.2 Use-Case Model

Use cases of Conquest are represented with the use case diagram. Verbal descriptions of use cases are included below.

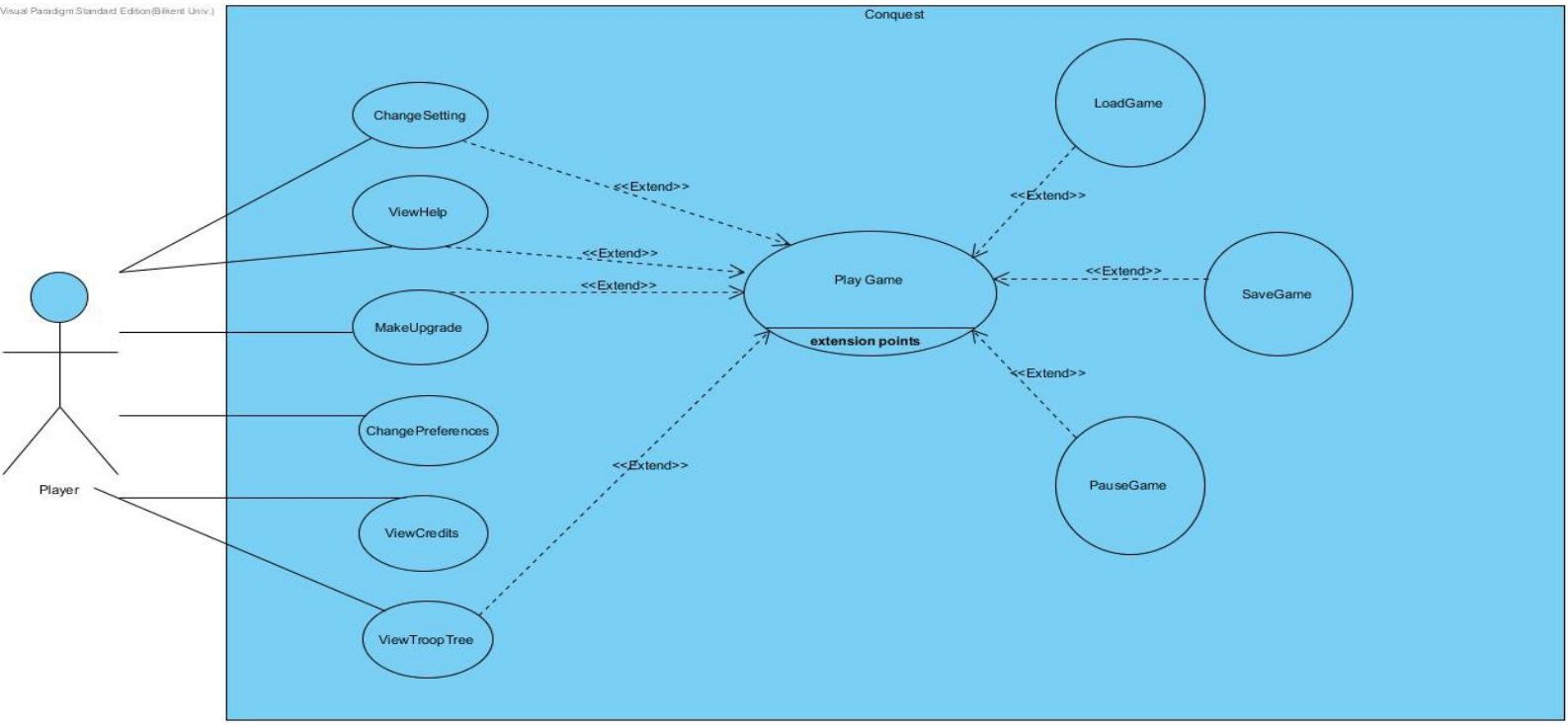


Figure 9 Use Case Diagram of Conquest

ChangeSettings: Player can view Settings page and change game settings.

ViewHelp: Player can view Help page to learn playing the game.

MakeUpgrade: Player can view upgrade his/her troops.

ChangePreferences: Player can change his/her region colors from preferences menu.

ViewCredits: Player can request to view Credits page to see developer details.

ViewTroopTree: Player can view troop tree to determine which upgrades he/she will be doing in the future.

PlayGame: Player can request to play the game by clicking play game button from opening menu.

PauseGame: Player can request to pause the game while playing the game.

SaveGame: Player can request to save the game while playing the game.

LoadGame: Player can load a previous game by clicking the load game button from opening menu.

2.5.2.1 Change Settings

Use Case Name: ChangeSettings

Actors: Player

Entry Conditions:

- ChangeSettings use case extends PlayGame use case
- Player is on Main Menu OR
- Player is on Pause Menu OR

Exit Conditions:

- Player is on Main Menu OR
- Player is on Pause Menu

Main Flow of Events:

1.Player presses Settings button

2. Conquest displays Settings page

3. Player views buttons for setting volume level

4. Player views buttons for turning the music on or off

5. Player changes settings by turning music on or off

6. Player selects volume level

7. Player presses Opening Menu or Pause Menu button

8. Conquest updates the settings

9. Conquest displays Main Menu or Pause Menu

2.5.2.2 ViewHelp

Use Case Name: ViewHelp

Actors:Player

Entry Conditions:

• ViewHelp use case extends PlayGame use case

• Player is on Main Menu OR

• Player is on Pause Menu

Exit Conditions:

• Player is on Main Menu OR

• Player is on Pause Menu

Main Flow of Events:

1. Player presses Help button

2. Conquest displays Help page

3. Player views tutorials to learn how to play Conquest

4. Player presses Opening Menu button

5. Conquest displays Main Menu or Pause Menu

2.5.2.3 MakeUpgrade

Use Case Name: MakeUpgrade

Actors: Player

Entry Conditions:

• MakeUpgrade use case extends ViewTroopTree use case

• Player is on World Map

Exit Conditions:

• Player is on Opening Menu

Main Flow of Events:

1-Player presses upgrade troops button

2-Conquest displays available upgrades

3-Player views available upgrades

4-Player chooses an upgrade

5-Conquest updates the upgraded troops conditions

6-Player presses back button

7-Conquest displays World map

2.5.2.4 ChangePreferences

Use Case Name: ChangePreferences

Actors:Player

Entry Conditions:

- ChangePreferences use case extends PlayGame use case
- Player is on the Opening Menu

Exit Conditions:

- Player is on the Opening Menu

Main Flow of Events:

1. Conquest displays Opening Menu with the purchased items
2. Conquest shows player color
3. Player views available colors
4. Player changes current selection
5. Conquest updates Player selections
6. Player presses Main Menu button
7. Conquest displays Main Menu

2.5.2.5 ViewCredits

Use Case Name: ViewCredits

Actors:Player

Entry Conditions:

- Player is on the Opening Menu

Exit Conditions:

- Player is on the Opening Menu

Main Flow of Events:

1. Player presses Credits button
2. Conquest displays Credits page

3. Player views credits to learn developer details

4. Player presses Main Menu button

5. Conquest displays Main Menu

2.5.2.6 ViewTroopTree

Use Case Name:ViewTroopTree

Actors:Player

Entry Conditions:

- Player is on the Opening Menu

Exit Conditions:

- Player is on the Opening Menu

Main Flow of Events:

1-Player presses view troop tree button

2-Conquest displays troop tree

3-Player learns about the troop upgrades

4-Player presses Main menu button

5-Conquest displays Main menu

2.5.2.7 PlayGame

Use Case Name:PlayGame

Actors:Player

Entry Conditions:

- Player selected Play Game from opening menu
- Player selected Load Game from opening menu

Exit Conditions:

- Player won the game and returned to main menu
- Player surrendered the game returned to main menu

Main Flow of Events:

1-Player clicks play game button.

2-Conquest creates the game

3-Player plays the game:

a.Player attacks

b.Player upgrades troops

c.Player reinforces troops

d.Conquest handles battles

e.Conquest handles troop upgrades, controls and updates time, points and of the Player

4-Player finishes the game

5-Conquest checks which player is the winner

6-Conquest declares the winner

7-Conquest directs use to Main Menu

2.5.2.8 PauseGame

Use Case Name: PauseGame

Actors: Player

Entry Conditions:

• PauseGame use case inherits PlayGame use case

• Player is playing a Level OR

Exit Conditions:

• Player is on World Map OR

• Player is playing the game

• Player changes Settings

• Player views Help page

Main Flow of Events:

1. Player presses pause button

2. Conquest pauses the game screen

3. Conquest displays Pause Menu

4. Player selects to continue

5. Conquest continues the game

2.5.2.9 SaveGame

Use Case Name:SaveGame

Actors:Player

Entry Conditions:

- Save Game inherits the PlayGame use case
- Player is on the pause menu

Exit Conditions:

- Player is on the pause menu

Main Flow of Events:

- Player selects Save Game button from Pause Menu
- Conquest Saves Game
- Conquest declares wheter the game is saved or not
- Conquest directs player to Main Menu

2.5.2.10 LoadGame

Use Case Name:LoadGame

Actors:Player

Entry Conditions:

- Load Game use case extends Play Game use case
- Player is on the Opening Menu

Exit Conditions:

- Player is on the Opening Menu

Main Flow of Events:

1-Player selects the Load Game button

2-Conquest displays saved games

3-Player selects which saved game will be played

4-Conquest loads the selected game

2.5.3 User Interface

2.5.3.1. Openning Scene

This is the opening scene of conquest where players starts the game.

Figure 10 Opening Scene of Conquest

2.5.3.2. Main Menu

This is the menu where players can start a new game, load a previosu game, change setting, get help from tutorials, see credits and exit the game.

Figure 11 Main Menu Of Conquest

2.5.3.3. Choose Players Menu

This menu appears after the player selects starting a new game option from main menu and selects number of players (min 2, max 4) and starts the game or he/she can return to the main menu.

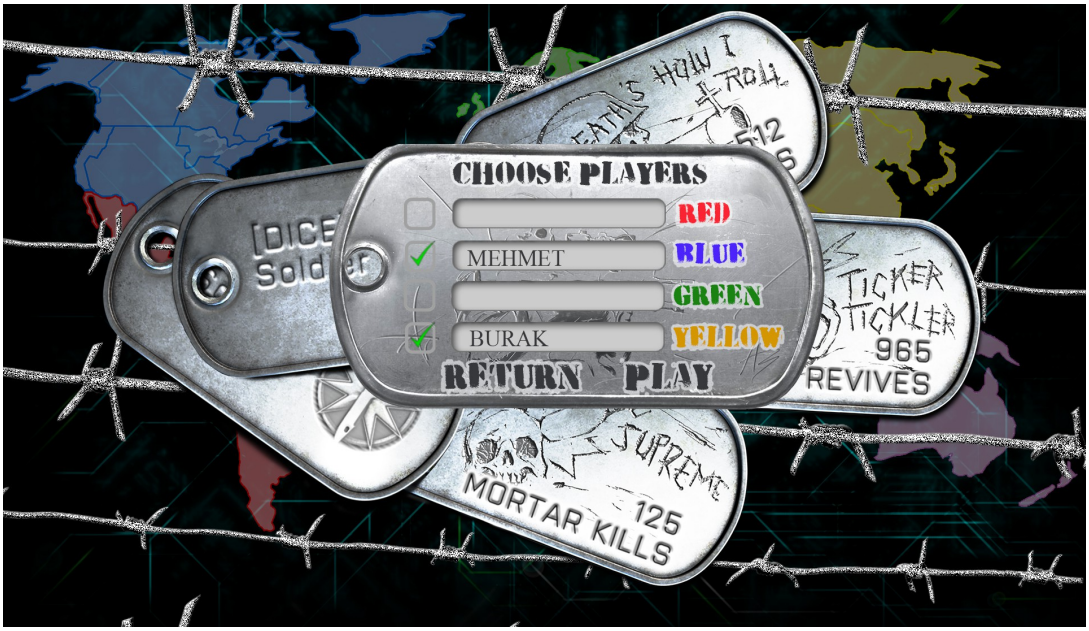


Figure 12 Player Selection Menu of Conquest

2.5.3.4. Pause Menu

This menu can be opened inside the game and player can continue to play or save the game and exit or change settings, see help tutorials or return to main menu.

Figure 13 Pause Menu of Conquest

2.5.3.5. Settings Menu

In this menu player can change settings such as making sound on and of and return to pause screen.

Figure 14 Settings Menu of Conquest

2.5.3.6. Upgrade Menu

This screen can be opened inside the game and player can see his/her remaining experince points, available upgrades for special troops, upgrade costs, troop levels and troop stats after and before the upgrade.

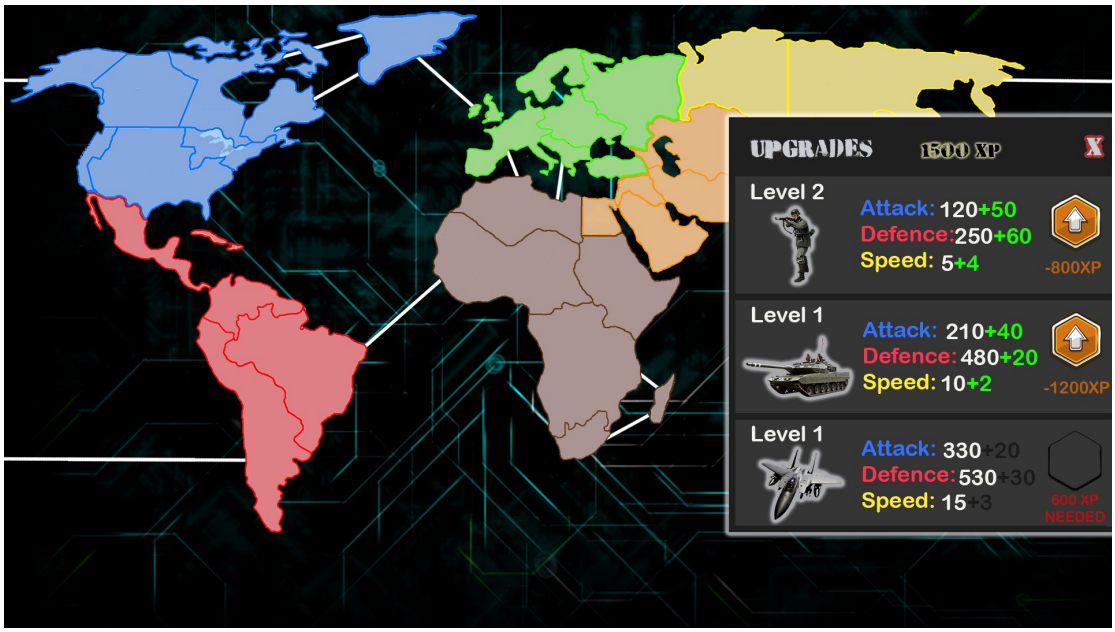


Figure 15 Upgrade Menu of Conquest

2.5.3.7. Map

This is the map which Conquest will be played on. Player will be able to see his regions according to his color.

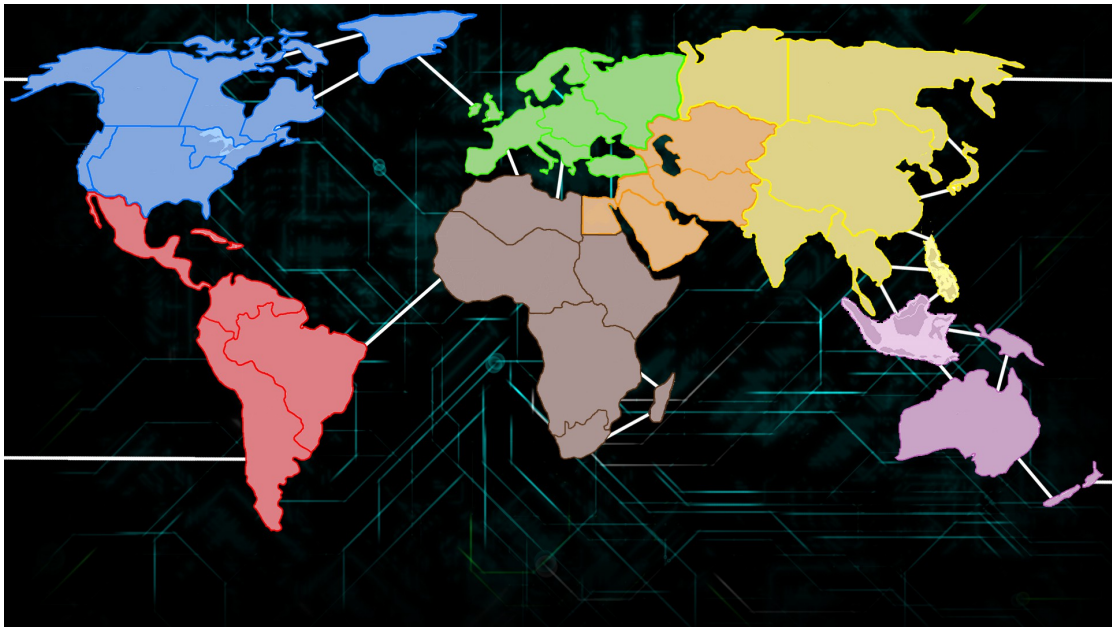


Figure 16 Play Map of Conquest

2.5.3.8. Leaderboard

This menu will appear at the end of the game. Plaer statistics will be shown on the menu. Player can be able to restart the same game from beginning or return to main menu.

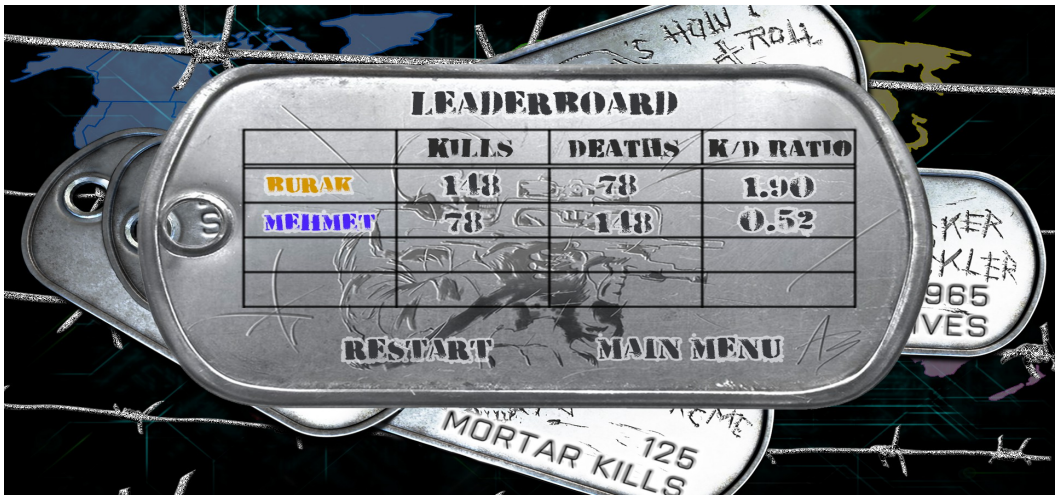


Figure 17 Leaderboard of Conquest

3 Analysis

3.1 Object Model

3.1.1 Domain Lexicon

User/**Player**: Person who plays Conquest

Game: Concept of overall system, Conquest

Troops: Soldiers that each player have.

Jet: Very fast and powerful type of a plane.

Tank: Heavy and powerful machines that used on guerilla warfare.

Infantry: Soldiers on foot on battlefield.

Bonus: A kind of extra troop reinforcement to the player from the game itself according to special conditions.

K/D Ratio: Player will kill or lose troops in wars and this statistic will be held until the game in order to display players ratio on leaderboard.

Time: The amount of minutes specified for each level

3.1.2 Class Diagram

The below diagram demonstrates classes in Conquest application. The Class diagram includes Boundary, Control and Entity objects and shows relations between these objects.

Control Objects

GameManager: Acts as a bridge between classes and provides required information

UpgradeManager: Checks and performs, if available, the selected upgrade

ActionManager: Provides connection between user inputs and required actions (for War and Reinforce)

WarManager: Performs the requested War action between two regions

UserManager: Container for User objects

SettingsManager: Sets and provides settings

SoundManager: Plays/Pauses music

MapManager: Holds the information about regions, their owners and TroopManagers

ReinforcemenetManager: Checks and performs requested Reinforce action between two regions

TroopManager: Container for Troop objects and their levels

BonusManager: Checks and provides bonus troops to users

ScreenManager: Draws the given Jpanel objects to the main frame

Entity Objects

Troop: Interface class for troops

Jet: is a specialized Troop

Tank: is a specialized Troop

Infantry: is a specialized Troop

User: Holds the information about the User (color, name, statistics etc.)

Region: Holds the information about regions (id, name, neighbors)

Continent: Consists of a Region array serves for BonusManager

Boundary Objects

MapView: Draws the game map and initializes regions

Menu: interface for creating sub-menus

SettingsMenu: Creates the required Panel for Settings Menu

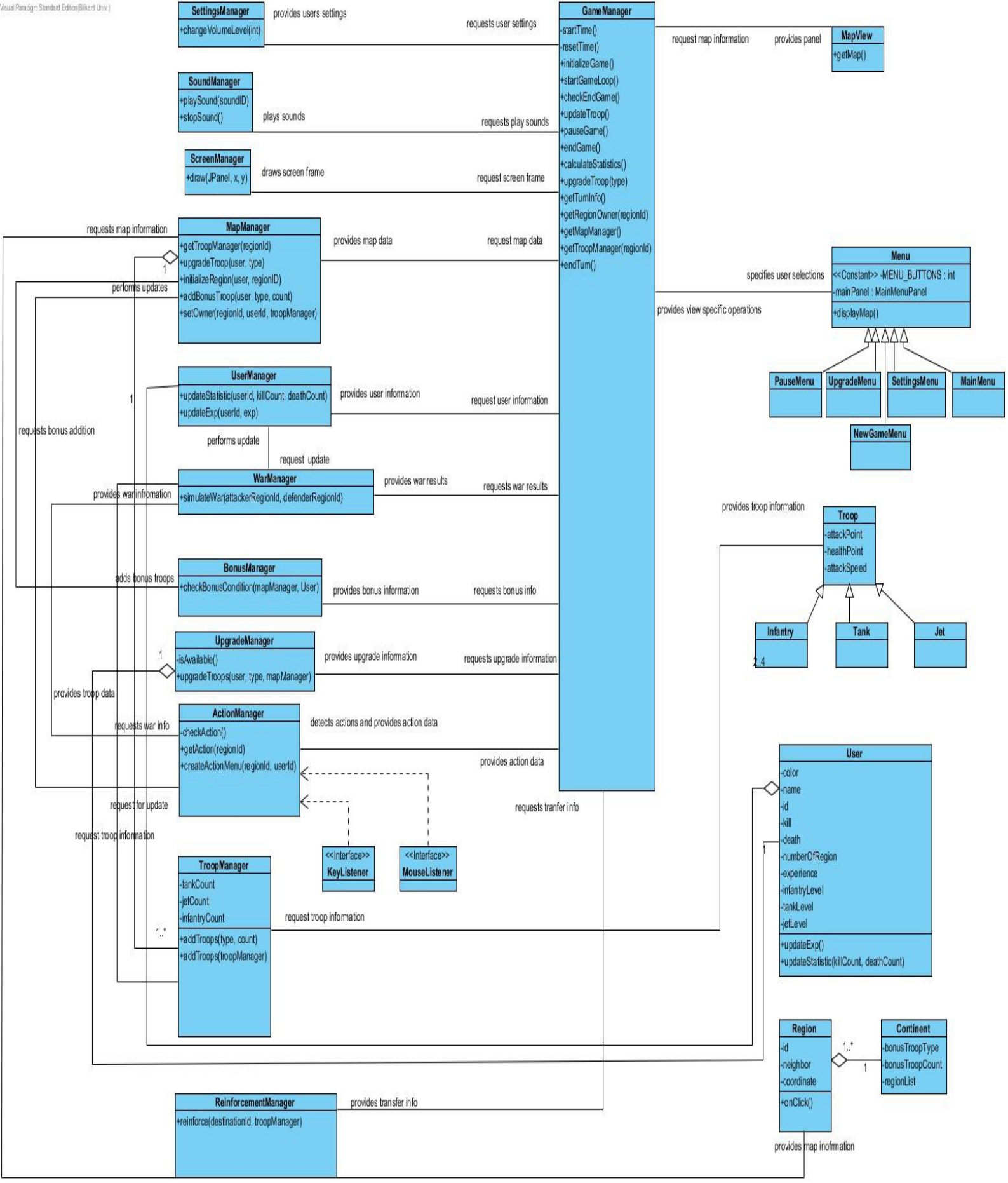


Figure 18 Class Diagram of Conquest

NewGameMenu: Creates the new game screen for adding users and selecting their names and colors

3.2 Dynamic Models

3.2.1. State Chart and Activity Diagrams

3.2.1.1. State Chart Diagram of Troops

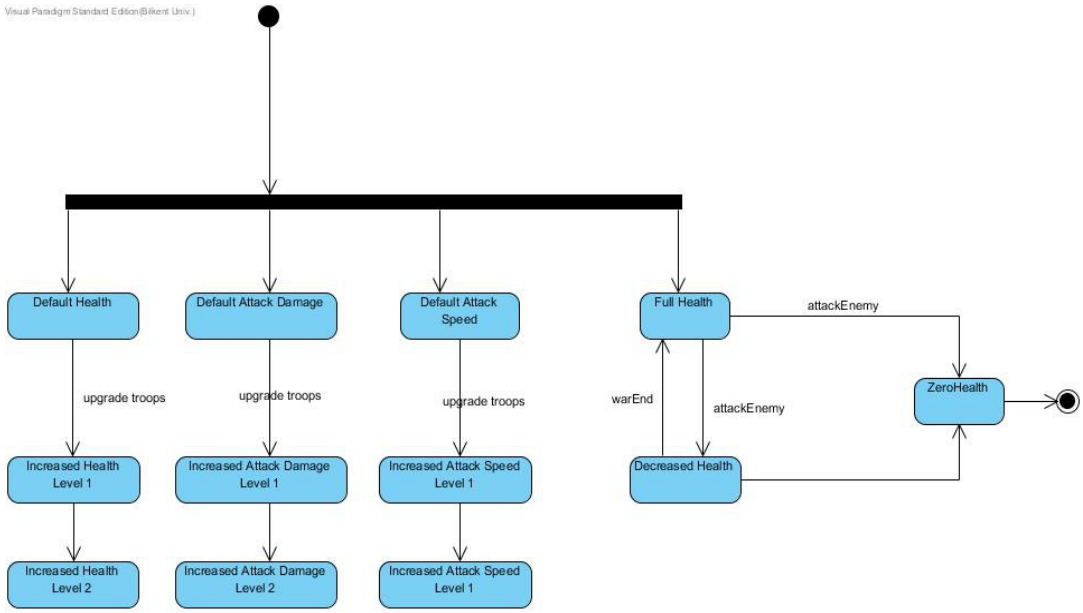


Figure 19 State Diagram of Troops

The above State Chart diagrams demonstrates the dynamic behavior of Troop class. State Chart diagram for Troop will be examined in 3 subgroups, damage, attakcSpeed, health. All these activities occur concurrently.

When the game starts troops will have default health, default attack damage and default attack speed. When there is and upgrade on any troop their stats will change and level up and for example, default health state will go to increased health state, default attack damage state will go on increased attack damage state, default attack speed state will go on increased attack speed state.

When there is a war between to players the troops will attack each other and lose health accordingly to the opposite troop's attack damage and attack speed until one of them gets a "0" health this means the troops is dead and will be deleted from the game. The ZeroHealth state is the final state since the troop will be deleted after getting a zero health point.

3.2.1.2 Activity Diagram for Main Menu

The diagram above shows the overall dynamic behavior of the game Conquest, game navigations and operations. Conquest starts up with a Start Screen that greets the users. After Start Button is pressed Main Menu Screen is displayed where user can navigate between screens like File explorer- that loads save files, which are basic txt files, and starts saved games-, Choose Players Screen, Setings Screen, Credits Screen and Help Screen. Load game Button opens a file explorer for user to choose a save file and load it to

play saved game which is represented as Gameplay in the diagram. If loading is canceled game returns back to Main Menu Screen. Another way to start Gameplay is by selecting New Game Button which takes user to Choose Players Screen in which Players up to 4 Players select their team color, enter names and start game with the Play Button. User can also select the Return Button to go back to the Main Menu Screen. With the Settings Button user is taken to the Settings Screen. Here settings such as turning game music on or off can be changed and with the Return Button Main Menu Screen is displayed. User can also see credits and help screens by pressing Credits Button and Help Button respectively. From each screen, user can go back to Main Menu Screen with the Return Button.

Activity Diagram for Overall Game Flow

In Gameplay, user can pause the game with the Pause Button which displays the Pause Screen. From the Pause screen user can return to game with the Continue Button, go back to Main menu with the Main Menu Button, and Exit Game with the Exit Button. To keep the diagram focused on the main activities, some activities in the Gameplay and Pause Screen are not show in the favor of simplicity. But it should be mentioned that from the Pause Screen, user can save the current game with the Save Button, and can also access the Setting Screen and Help Screen by clicking Settings Button and Help Button respectively.

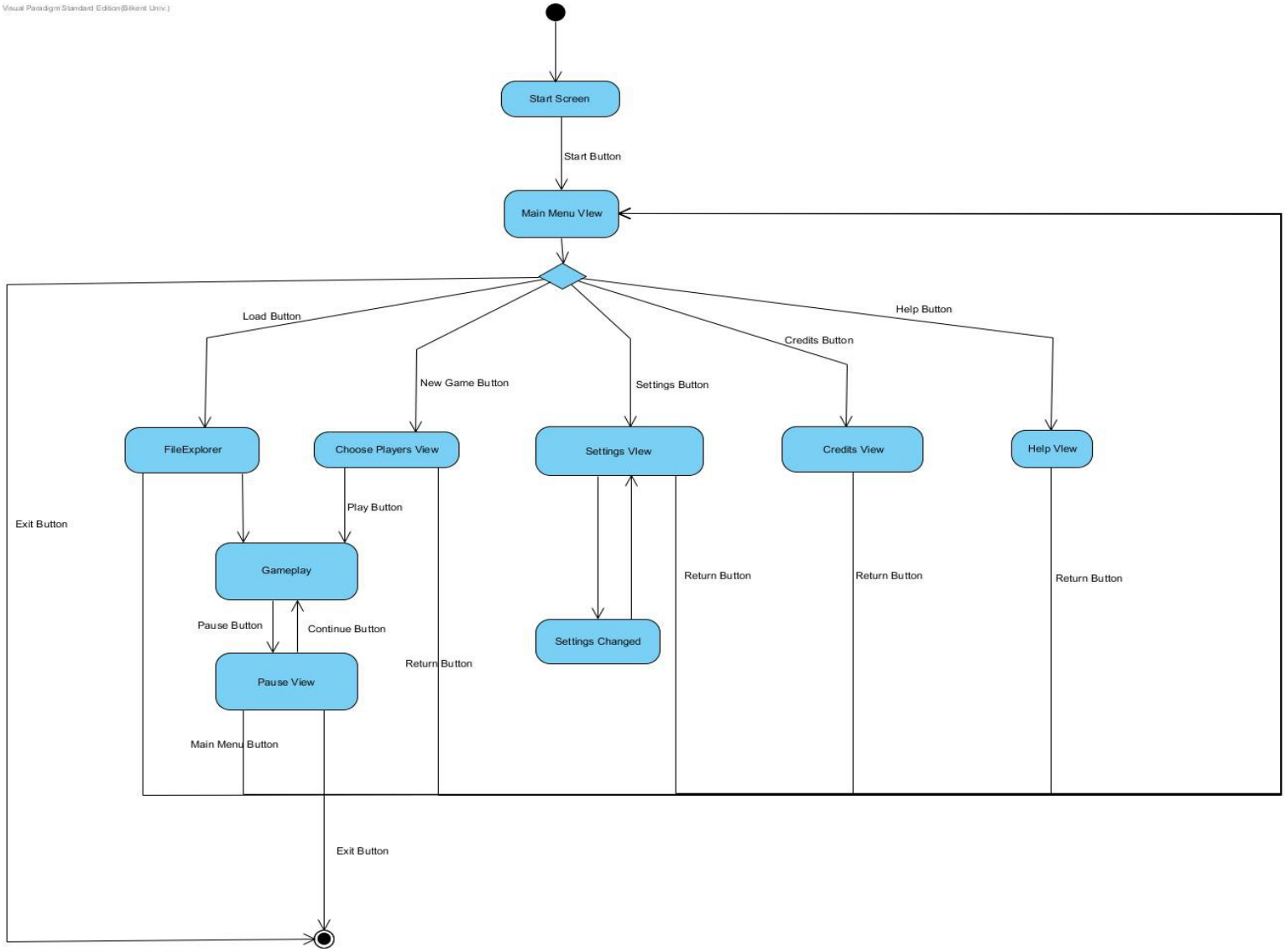


Figure 20 Activity Diagram of Main Menu

3.2.1.3 Activity Diagram for New Game

New Game Scenario:

Ali opens the game. Clicks 'New Game' button. In the following screen he chooses his color blue and writes his name. Then, his friend Veli chooses red for himself and writes his name. Then lastly Ayşe picks her color and name, after that clicks the 'Start Game' button. Game loads the map and randomly distributes each region to the three player. They each distribute their units over the regions that they own. Then Ali starts the game and makes his first move.

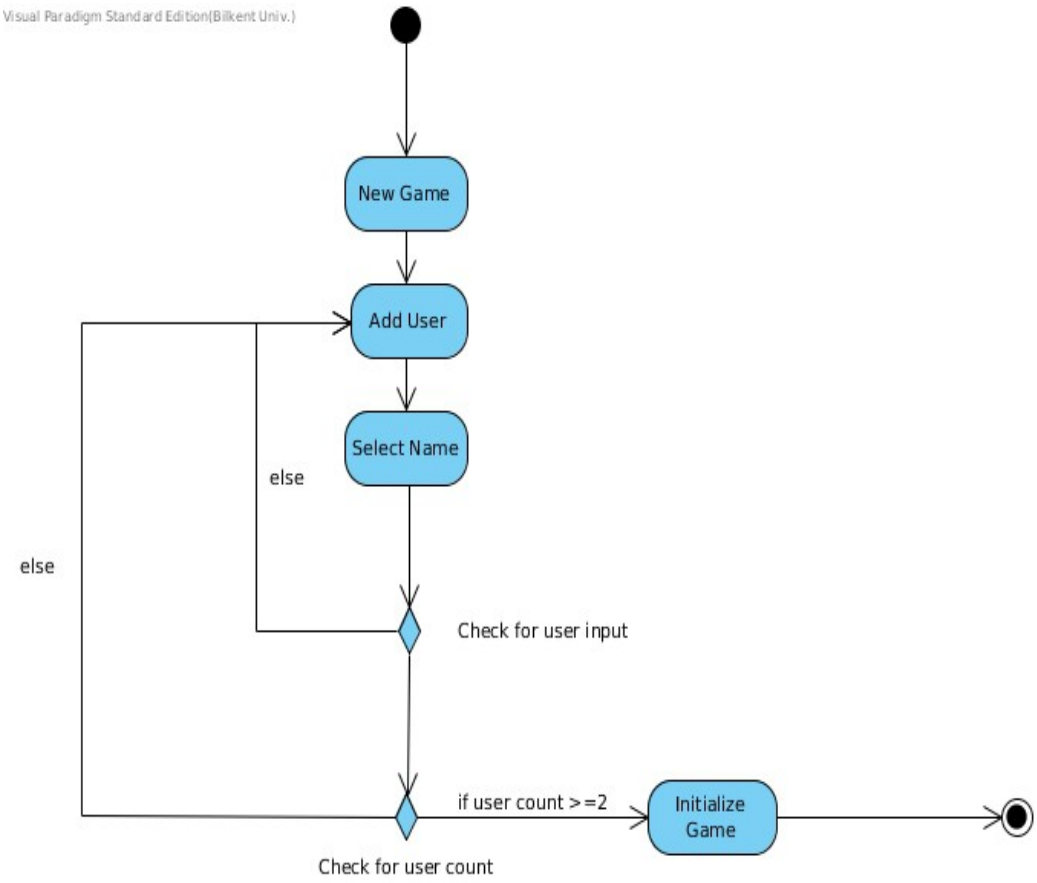
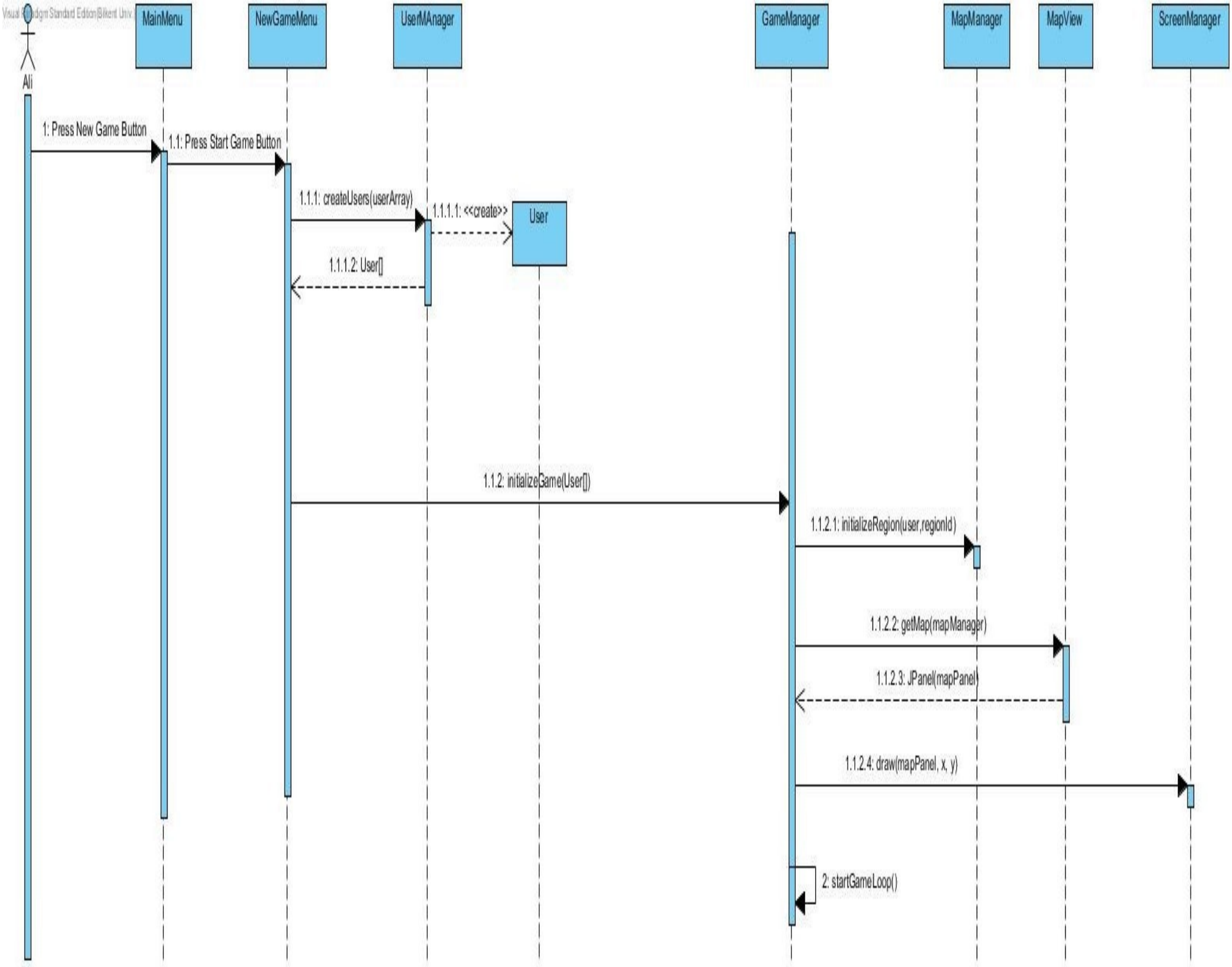


Figure 21 Activity Diagram of New Game

3.2.2. Sequence Diagrams

3.2.2.1. Starting a New Game


Ali
MainMenu
NewGameMenu
UserMAnager
GameManager
MapManager
MapView
ScreenManager
User
1: Press New Game Button
1.1: Press Start Game Button
1.1.1: createUsers(userArray)
1.1.1.1: <<create>>
1.1.1.2: User[]
1.1.2: initializeGame(User[])
1.1.2.1: initializeRegion(user,regionId)
1.1.2.2: getMap(mapManager)
1.1.2.3: JPanel(mapPanel)
1.1.2.4: draw(mapPanel, x, y)
2: startGameLoop()

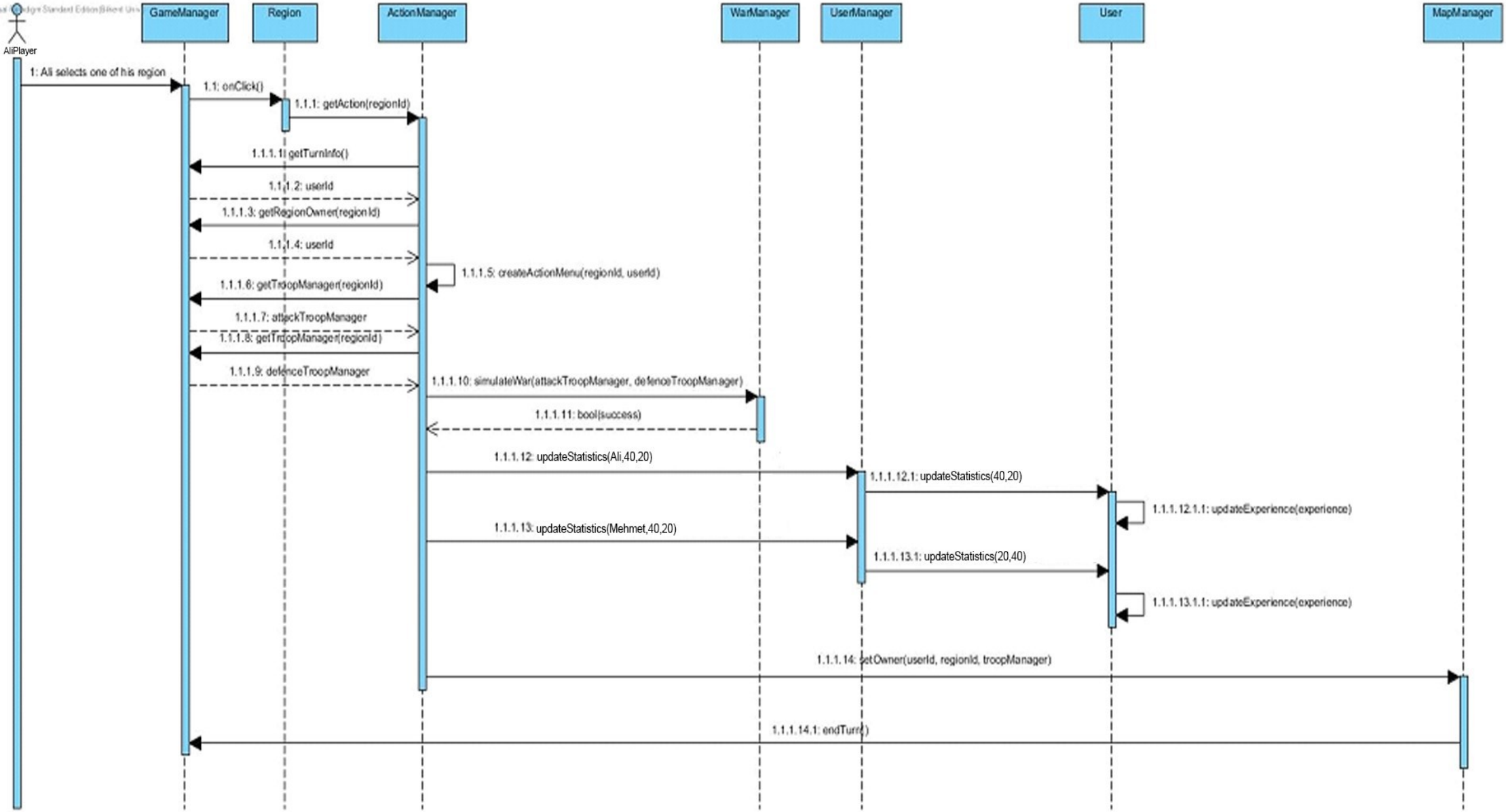

AliPlayer
GameManager
Region
ActionManager
WarManager
UserManager
User
MapManager
1: Ali selects one of his region
1.1: onClick()
1.1.1: getAction(regionId)
1.1.1.1: getTurnInfo()
1.1.1.2: userId
1.1.1.3: getRegionOwner(regionId)
1.1.1.4: userId
1.1.1.5: createActionMenu(regionId, userId)
1.1.1.6: getTroopManager(regionId)
1.1.1.7: attackTroopManager
1.1.1.8: getTroopManager(regionId)
1.1.1.9: defenceTroopManager
1.1.1.10: simulateWar(attackTroopManager, defenceTroopManager)
1.1.1.11: bool(success)
1.1.1.12: updateStatistics(Ali,40,20)
1.1.1.12.1: updateStatistics(40,20)
1.1.1.12.1.1: updateExperience(experience)
1.1.1.13: updateStatistics(Mehmet,40,20)
1.1.1.13.1: updateStatistics(20,40)
1.1.1.13.1.1: updateExperience(experience)
1.1.1.14: setOwner(userId, regionId, troopManager)
1.1.1.14.1: endTurn()

Figure 21 Sequence Diagram of Starting a New Game

3.2.2.2 Upgrading Troops

Scenario: While playing the game and player Ali has the turn, player Ali presses the upgrade button on the UpgradeMenu and Upgrade Menu requests GameManager to upgrade player Ali's troops. Then, GameManager asks UpgradeManager to if upgrade is available for player Ali. If upgrading is available for player Ali, then GameManager requests UpgradeManager to make the upgrade. Then UpgradeManager calls the MapManager to specify the regions of the player because all the regions of player Ali must be upgraded. After specifying regions of player Ali, MapManager requests each TroopManager of player Ali's regions to upgrade the troops of its region.

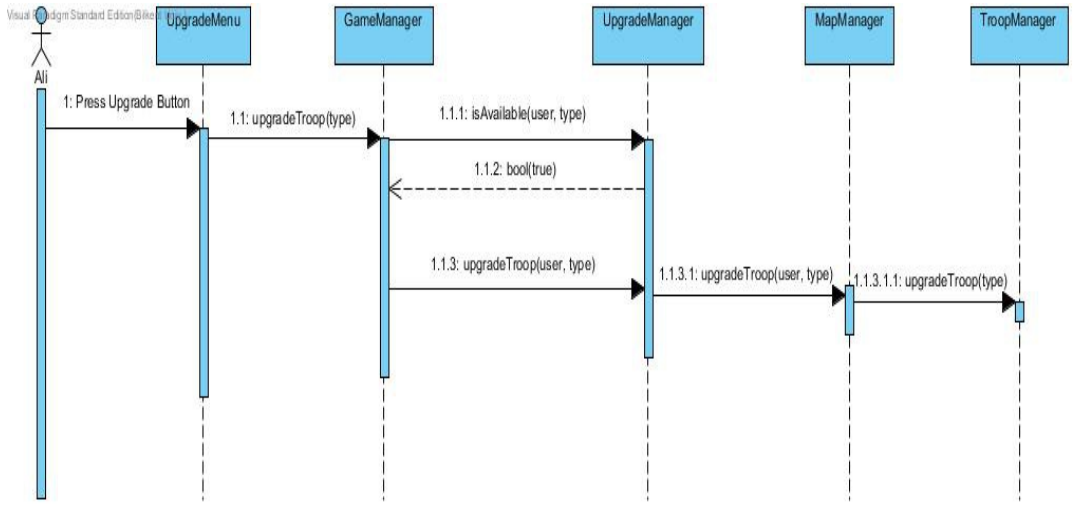


Figure 22 Sequence Diagram of Upgrading Troops

3.2.2.3 Attacking an Enemy

Scenario: (Player Ali attacks an enemy and win the war) While playing the game and player Ali has the turn, Ali clicks on one of his regions and then the click listener of his region sends its own id to ActionManager and ActionManager requests the id of the Ali from GameManager. After GameManager sends the id of the Ali, ActionManager requests the id of the owner of the clicked region. If the both requested id are the same, then ActionManager displays the ActionMenu. After Ali chooses the region which he wants to attack ActioManager requests the defender troop and the attacker troop and sends the troops to the WarManager to handle the war. After handling the war, WarManager sends the result of the war to ActionManager and ActionManager calls UserManager to update the statistics of the user related to the war then UserManager calls the User object and the User object update its own experience which is gained on the war. After experiencing is done, ActionManager calls the MapManager to change the owner of the region and make Ali the owner of the region because he won the war then MapManager requests GameManager to end the turn.

Figure 23 Sequence Diagram of Attacking an Enemy

3.2.2.4 Earning Bonus

Scenario: In the beginning of Player Ali's turn GameManager request BonusManager to checkwhat is the Bonus condition of Player Ali with the checkBonusCondition(mapManager, user) function. According to the condition Map manager requests to add bonus troops with the addBonusTroop(user,type,count) function which calls TroopManagers addTroop(type,count) function and adds appropriate amount of bonus troops to Player Ali's army.

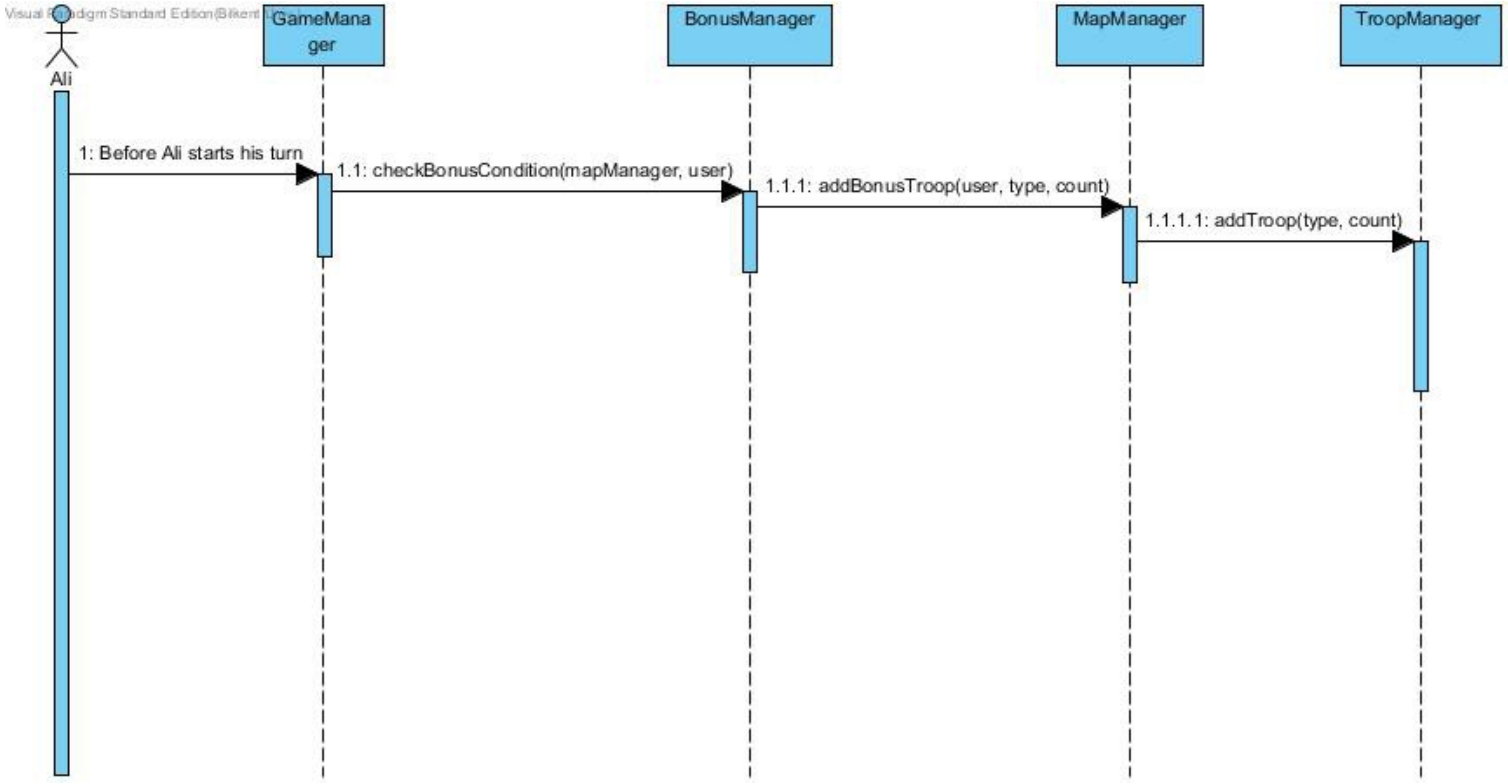


Figure 24 Sequence Diagram of Earning Bonus

3.2.2.3 Reinforcement of Troops

Scenario: While playing the game Player Ali decides to move his troops from one of his regions to one of his other regions. Ali first selects one of his regions that he wants to reinforce from and selects reinforcement option from the menu that opens up. Ali selects types and counts of the troops that he wants to send. Ali selects the region that he wants to send troops to. With the leave of the troops from the selected region Ali's turn ends.

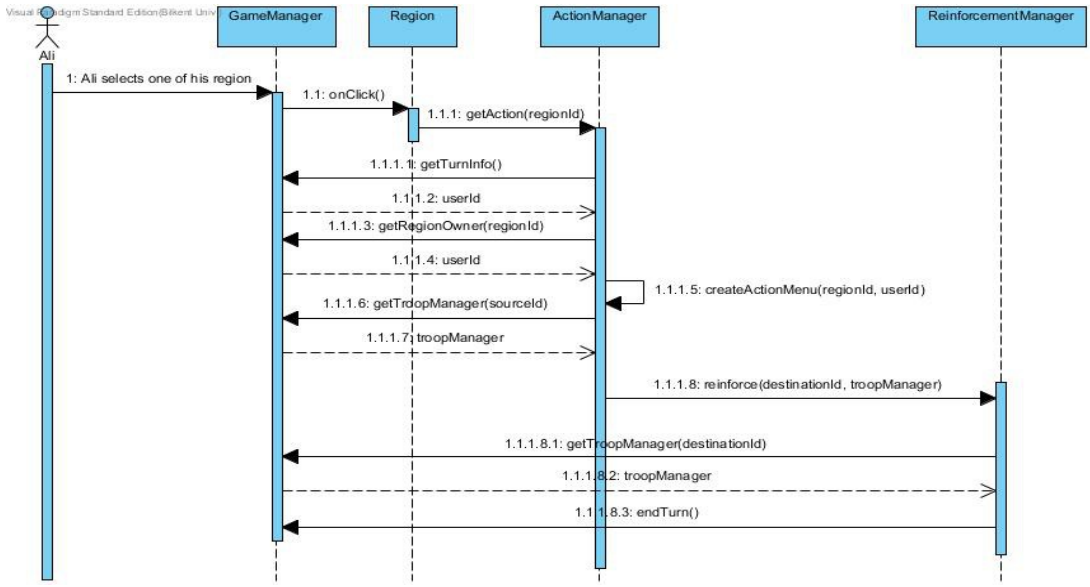


Figure 25 Sequence Diagam of Reinforcement of Troops

4 Design

4.1. Design Goals

Reliability

Reliability is the ability of the system to perform the operations which are required. Our goal is to make our system reliable, so that our system should be able to perform any operations in the game until user decided to stop playing the game. So, the system should avoid crashes while performing any operation. To do that, system should be implemented very carefully and if a crash happens, system should not lose any data of the user.

Usability

User-Friendliness

In our project, the user friendliness will be one of the most important goals of our project. In order to achieve this, our system will be tested by the users and we will get feedbacks from these users and make our system more user friendly. Also, a clear user interface will be needed in order to achieve our goal.

Ease of Use

Our software will be easy to use because of its friendly interface. Navigating in the game will be very understandable and our play game guide will guide users to make them use our software comfortably.

Understandability

Every part of our project will be understandable by users, developers and reviewers. To achieve this goal, we are designing our reports very clearly. We are also focusing on the consistency and completeness during development process.

Performance

Rapid-Development

Since our project will be an object oriented software, our design models and UML diagrams will help us in the implementation part of the project and we are aiming to reduce the implementation part as much as we can by drawing our diagrams and writing our reports carefully.

High Performance

Our aim to make our system as fastest as we can. It is not a graphical game but it has a little war simulation which is handled by WarManager. We will try to simulate the wars as fastest as we can so that users will not be have to wait for the game to simulate the war.

Supportability

Flexibility

We are aiming to make our system flexible so that our system will be able to adapt itself for future changes. To achieve that, we will minimize the coupling of the components so that most of the changes will not affect the whole system and our system will be flexible.

Modifiability

To make our software modifiable, the changes in the system should be easy to implement. When some modification occured in a part of the system, it should not affect the whole system and all parts of the system should work seperately. To achieve this goal, subsystems of the system should be seperate so that a change in a subsystem will not affect the other subsystems.

Maintainability

We will try to minimize the complexity of the system, so that we will be able to update the system or fix the bugs easily. To do this, we need to apart our system to subsystems correctly and minimize the complexity of the system.

Adaptability

We are planning to design our system adaptable so that our system will allow other developers to customize our system easily, so developing new version of the system or updating the system by other developers will be easy and other developers will have right for customizing. Since Java is one of the cross-platform portable programming language it will support our adaptability goal and operating system constraints will be handled by java. Also our code will be easy to read and understand by other developers since it will be an open source project.

Trade-offs

Functionality vs. Ease of Use

In our game, most of the times, mouse will be used to play the game and we will not have keyboard shortcuts to play the game. It will be easy to use our software because it is easy to use mouse but this is a trade off and easy of use is more important for us.

High Performance vs. Adaptability

Our software will be implemented in Java. We could have use another language to maximize the performance such as C++ but we decided Java because of the adaptability issues. So, Adaptability is more important for us than high performance.

High Performance vs. Reliability

We want to make our system reliable and we want to prevent any data loss. It means that we need to store the data of the users as much as we can and it will affect the performance of the game. But high performance is valuable if a system is reliable. Reliability is more important than high performance but their weight must be balanced to make an optimized system.

4.2. Subsystem Decomposition

Conquest system is decomposed to three parts: Game View, Game Control and Game Model. These layers are ordered hierarchically and they have run-time dependency. Game View subsystem contains Boundary classes and View components. This subsystem is responsible from interacting with user and handling view operations. It is the top layer and calls Game Control subsystem to handle game-related operations. Game Control subsystem is the middle layer and it contains classes responsible from the fundamental functions and operations of the game. The Game Control subsystem provides services related to all operations of the game including level operations. This subsystem calls Game Model subsystem for retrieving necessary data for game operations. Game Model is the lowest layer. It provides data-related services such as data retrievals and updates. It consists of all classes containing game data.

Closed Architecture

Conquest architecture is a closed architecture in which layers can access to services of only layer below them. Hence, Game View subsystem can only call services from Game Control subsystem and not from Game Model subsystem. Even though this architecture decreases Efficiency, it gains Flexibility to system since layers become more independent of each other. Moreover, a closed architecture is also more Maintainable since the run-time dependencies decrease.

Coupling and Coherence of Subsystems

In our subsystem decomposition, we aimed and achieved high coherency. Classes within subsystems share a common concept and they perform similar and related operations. They interact with each other frequently for implementing related functionalities. F SoundManager is called

by GameManager for sound operations. This strong relation and unity among subsystem classes indicate that we have high coherence within our subsystems.

We also aimed and achieved low coupling. Hence, subsystems mostly do not depend on each other. There are weak relations between subsystems therefore, they can act more independently. For instance, only connection between Game View subsystem and Game Control subsystem is between GameManager and Menu and ScreenManager. Hence changes in a subsystem generally do not affect other subsystems. Our subsystem architecture demonstrates that we have low coupling.

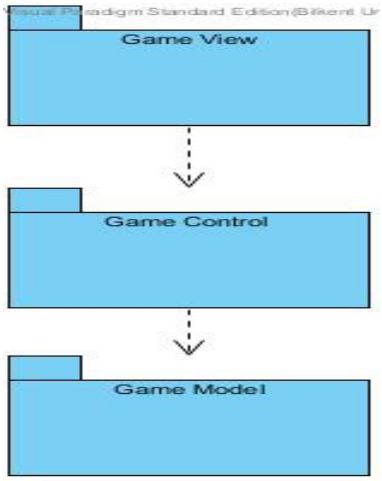


Figure 26 Basic Layers of Conquest

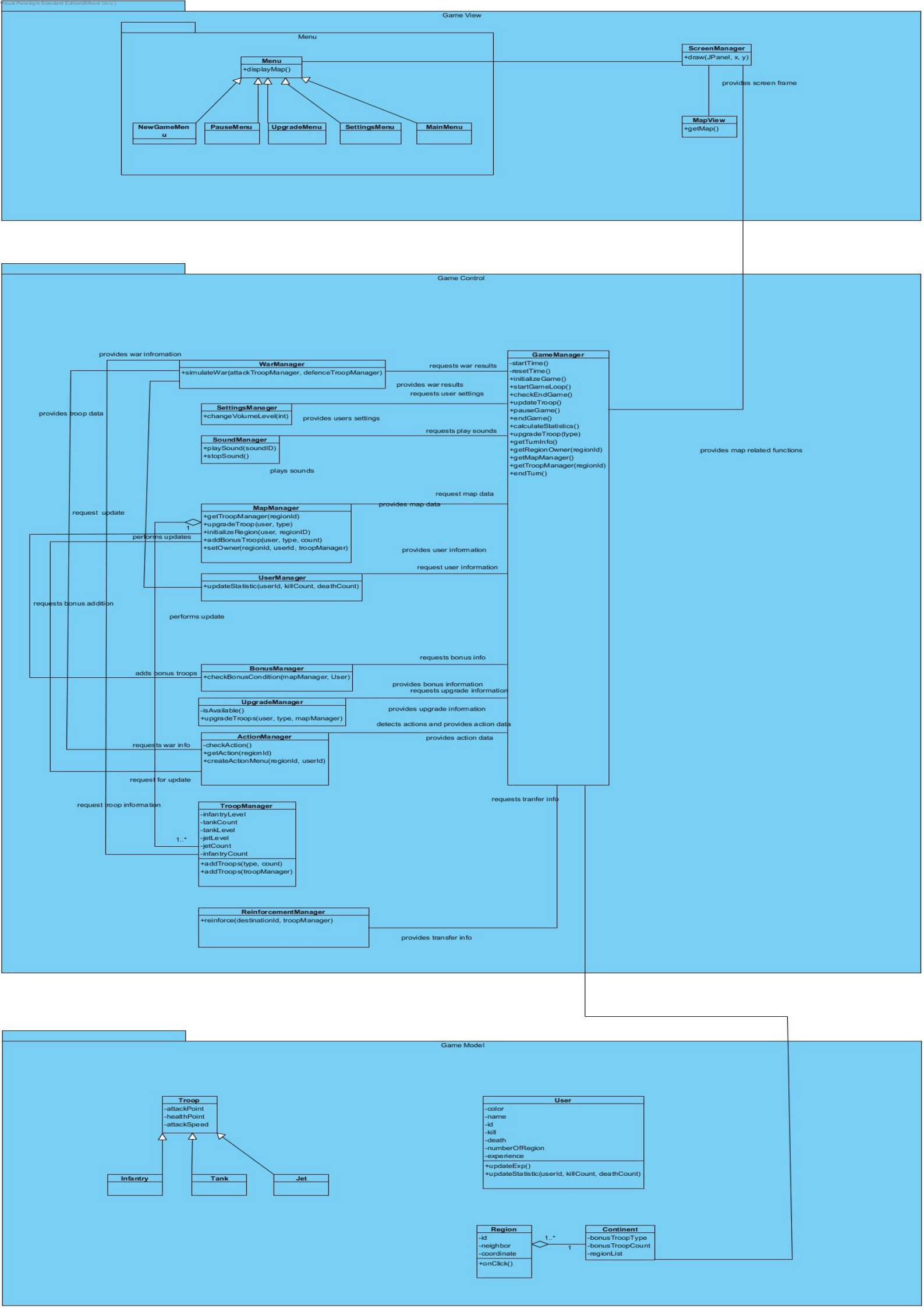

Game View
Menu
Menu
+displayMap()
NewGameMenu
PauseMenu
UpgradeMenu
SettingsMenu
MainMenu
ScreenManager
+draw(JPanel, x, y)
provides screen frame
MapView
+getMap()
Game Control
provides war infromation
WarManager
+simulateWar(attackTroopManager, defenceTroopManager)
requests war results
provides war results
requests user settings
GameManager
-startTime()
-resetTime()
+initializeGame()
+startGameLoop()
+checkEndGame()
+updateTroop()
+pauseGame()
+endGame()
+calculateStatistics()
+upgradeTroop(type)
+getTurnInfo()
+getRegionOwner(regionId)
+getMapManager()
+getTroopManager(regionId)
+endTurn()
provides troop data
SettingsManager
+changeVolumeLevel(int)
provides users settings
requests play sounds
SoundManager
+playSound(soundID)
+stopSound()
plays sounds
provides map related functions
request map data
provides map data
request update
MapManager
+getTroopManager(regionId)
+upgradeTroop(user, type)
+initializeRegion(user, regionID)
+addBonusTroop(user, type, count)
+setOwner(regionId, userId, troopManager)
performs updates
provides user information
request user information
UserManager
+updateStatistic(userId, killCount, deathCount)
requests bonus addition
performs update
requests bonus info
BonusManager
+checkBonusCondition(mapManager, User)
adds bonus troops
provides bonus information
requests upgrade information
UpgradeManager
-isAvailable()
+upgradeTroops(user, type, mapManager)
provides upgrade information
detects actions and provides action data
ActionManager
-checkAction()
+getAction(regionId)
+createActionMenu(regionId, userId)
provides action data
requests war info
request for update
request troop information
TroopManager
-infantryLevel
-tankCount
-tankLevel
-jetLevel
-jetCount
-infantryCount
+addTroops(type, count)
+addTroops(troopManager)
1..*
requests tranfer info
ReinforcementManager
+reinforce(destinationId, troopManager)
provides transfer info
Game Model
Troop
-attackPoint
-healthPoint
-attackSpeed
Infantry
Tank
Jet
User
-color
-name
-id
-kill
-death
-numberOfRegion
-experience
+updateExp()
+updateStatistic(userId, killCount, deathCount)
Region
-id
-neighbor
-coordinate
+onClick()
1..*
1
Continent
-bonusTroopType
-bonusTroopCount
-regionList

Figure 27 Conquest Subsystem Decomposition with Subsystem Details

4.3.Architectural Patterns

Model View Controller Architectural Style

Conquest is designed to have a architectural style similar to Model- View-Controller pattern. GameManager class and Menu class provides the only connection between Controller and View classes. GameManager conveys any requests from other controllers to the Menu class. Thus, limits the number of different connections between classes, and keeps them more isolated. This design principle also allows us to change the implementations of both other Controller classes and View classes without affecting the interacting between them and increases the flexibility.

Although there is only one way of interaction between Controllers and View, other Controller classes can directly access to their directly related Model classes without the help of GameManager, also some of them communicate with other Controller classes. This is done to both provide ease of implementation and to cut the middle-man in repeatedly performed actions. However, there connections are designed with the Principle of least privilege in mind; therefore, they are tried to be kept at minimum and not used unless it is considered as crucial.

4.4.Hardware/ Software Mapping

Conquest does not require advanced hardware. It has little to no animation. Although it has lots of objects, they are mostly processed under the hood and visually not all of them are shown. Hence, the computation rate is not demanding for a single processor. One processor will be necessary to maintain the game without flaws. Similarly, there is no need for additional memory space except for saving the current game. The game mainly needs mouse as an external I/O device for the game control, however keyboard is also required to enter the player names. Although Conquest is a multiple-player game it is played only in one computer. Hence, it does not require connection to a server or internet or any other device. Therefore, the only node in the system is UserComputer. The UserComputer should have JDK installed. Keyboard and Mouse I/O devices are also represented as external nodes. The components of Conquest are associated with subsystems; User Interface (View), Game Control (Control) and Game Model (Model). Game Control component provides game operations interface to User Interface. Game Model component provides data operations and Game Control uses this interface. This architecture is represented with below diagrams.

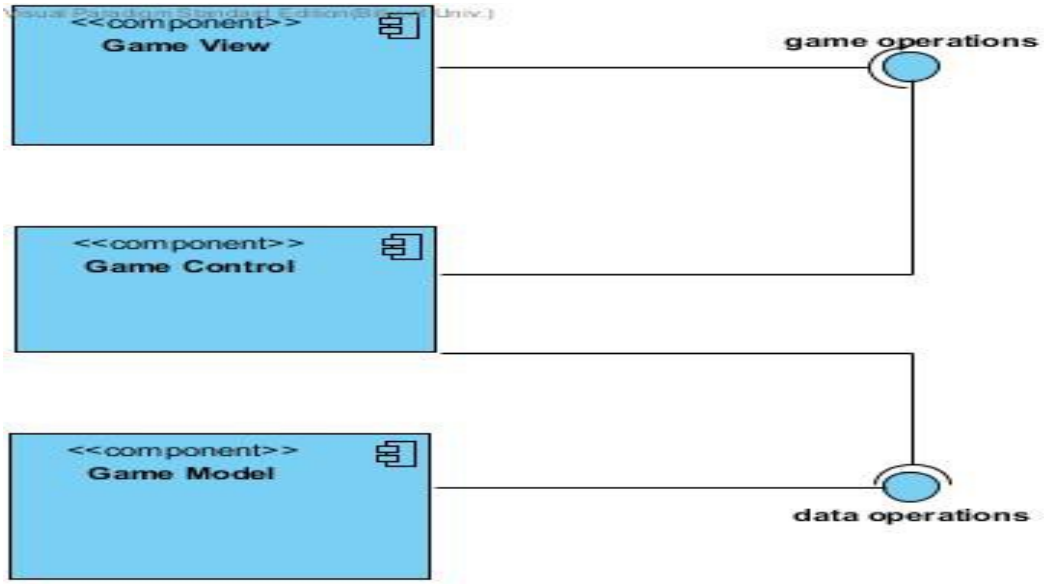


Figure 28 Component Diagram of Conquest

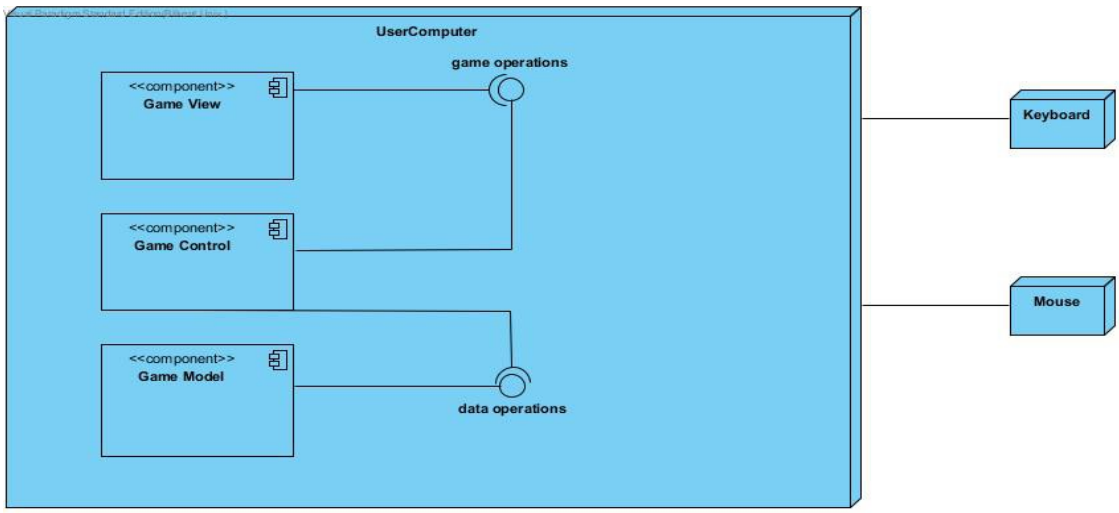


Figure 29 Deployment Diagram of the Conquest

4.5. Addressing Key Concerns

4.5.1. Persistent Data Management

In Conquest only games saved by user will be persistent, otherwise none of the data will be stored in the hard disk and will be lost when the game is closed. When the game is saved current status of the map will be recorded into a text file. This status will be saved in a format similar to JSON. Moreover, Since there is only one computer active, there will only be one reader and writer to the file. Therefore, it is appropriate to use simple text file instead of database system because overhead of the database system will be unnecessary for loading simple text files and those load and save options will not be performed frequently.

4.5.2 Access Control and Security

Since the game does not require any internet connection and can only be played with one computer, it does not have any authentication system; and since there is no access control, there cannot be a security issue about the system.

4.5.3. Global Software Control

In Conquest we have a main controller class, GameManager, and it decides flow of events. Since Conquest is a turn-based game and it depends on the mouse events, event-driven control is chosen. Therefore, the software

system will have a simpler structure and centralized design will be handled within the main loop of the game.

4.5.4. Boundary Conditions

Initialization

Conquest is provided as an executable .jar file. Hence, it does not require installation. It can be ran directly on any computer which has Java installed.

Termination

User will be able to quit the game any time he/she wants after pausing the game. Before the quit there will be a prompt to the user asking whether he/she wants to save the game before quit.

Failure

Conquest contains relatively small set of images and sounds. Thus, it will able to perform additional check during the initialization of the game to be sure that there is no corruption in the files. If the game is loading from a save file, it will check beforehand if there is an error in the file to prevent failure during the game.

If the system crashes, it will lost any unsaved game data. However, there will be an option for auto-save after turns to prevent data-loss.

5. Subsystem Services

5.1 Design Patterns

Façade Design Pattern

In our project we used Façade Design Pattern in order to manage the classes easily from a façade class. This design pattern provides more of the design goals that provided in this report. Hence we decided to use Façade design pattern on our project.

We used Façade design pattern on two subsystems such as; Game Control and User Interface. In Game Control subsystem. The façade class of the Game Control subsystem is GameManager and for User Interface subsystem the façade class is Main Menu.

5.2 User Interface Subsystem

User Interface Subsystem is used for graphical functions of our system. It also manages graphical components and Menu transactions.

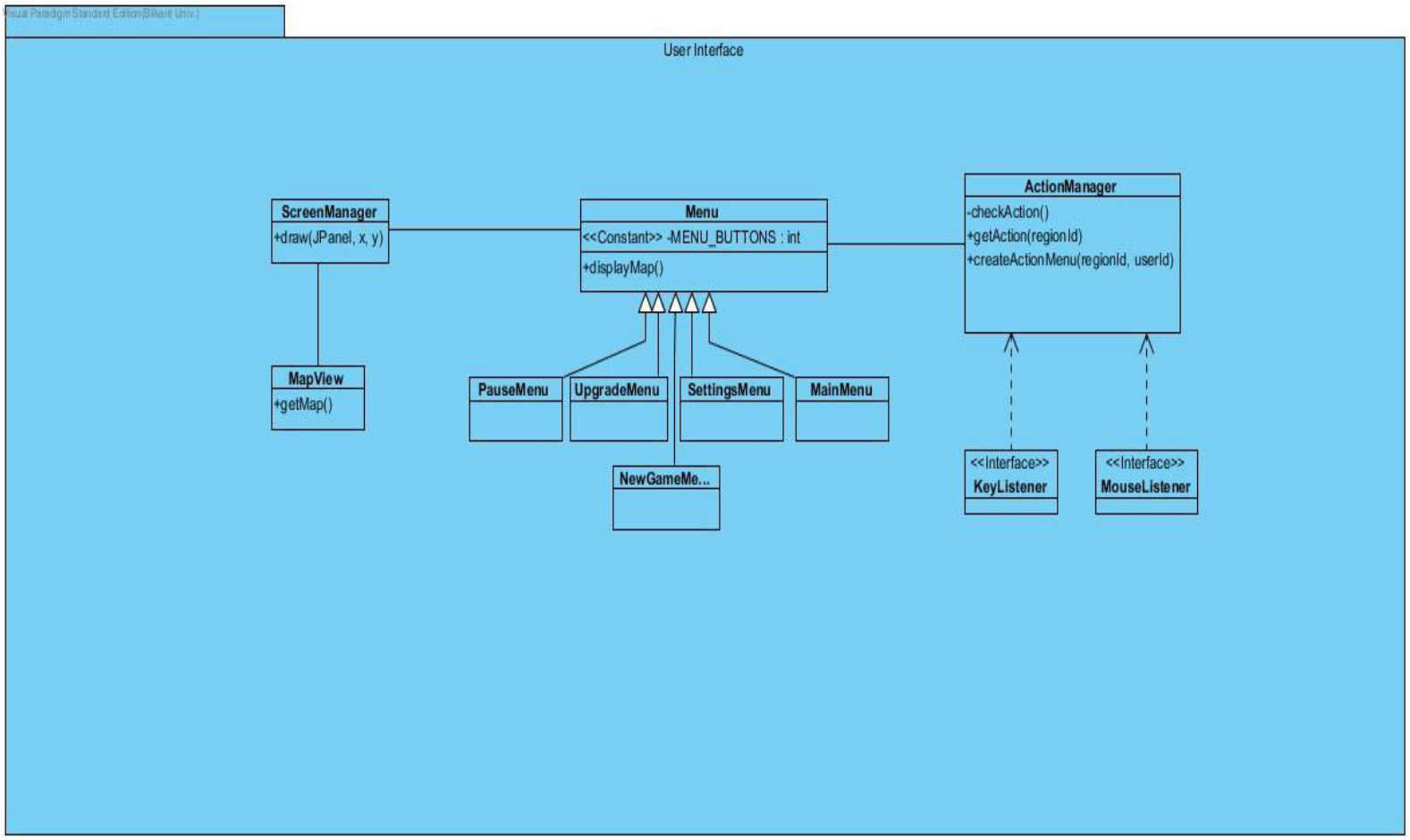
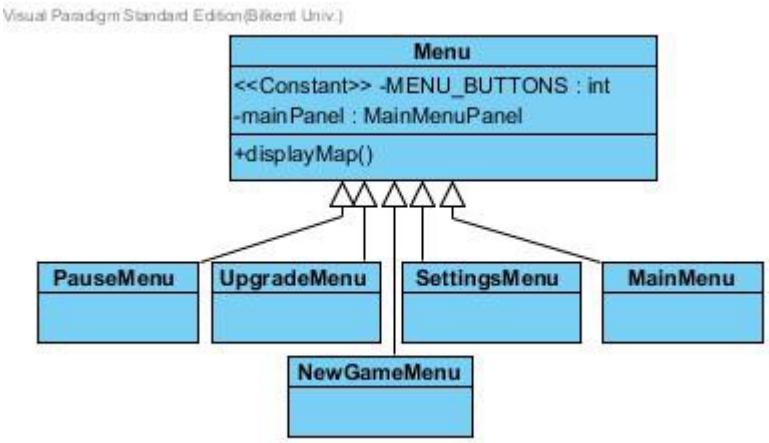


Figure 30 User Interface Subsystem of Conquest

MenuClass



Attributes:

private int MENU_BUTTONS: This attribute is a constant integer and it determines the number of the Menu buttons on the screen.

private JPanel mainPanel: This attribute is a Jpanel and it will be shown to user after adding the necessary buttons to this panel

Methods:

public void displayMap(): This method takes no arguments and when user presses a button to exit the Menu and go back to the game this function will be called and performs the operations to exit the Menu and go back to the game.

- **PauseMenu Class**
- **UpgradeMenu Class**
- **SettingsMenu Class**
- **MainMenu Class**
- **NewGameMenu Class**

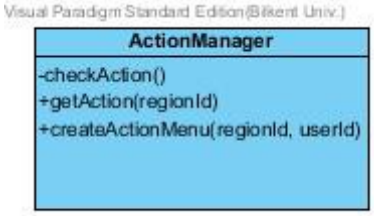
These classes are the child class of the Menu Class and they will all have the mainPanel and MENU_BUTTONS attributes and also the method named displayMap().

ScreenManager :

Methods:

public void draw(JPanel, x, y) : This method takes a panel object and x,y coordinates and draws the panel on the x,y coordinate.

ActionManager



Attributes

Constructors

ActionManager(MapManager m) : this constructor takes and mapManager argument and creates and action manager object using mapManager object

MapView :

Methods :

public void getMap() :

5.2 Game Control Subsystem

Game Control Subsystem is used to control the Conquest and the logic behind it. This subsystem has 5 components inside it. They are all used for managing the system.

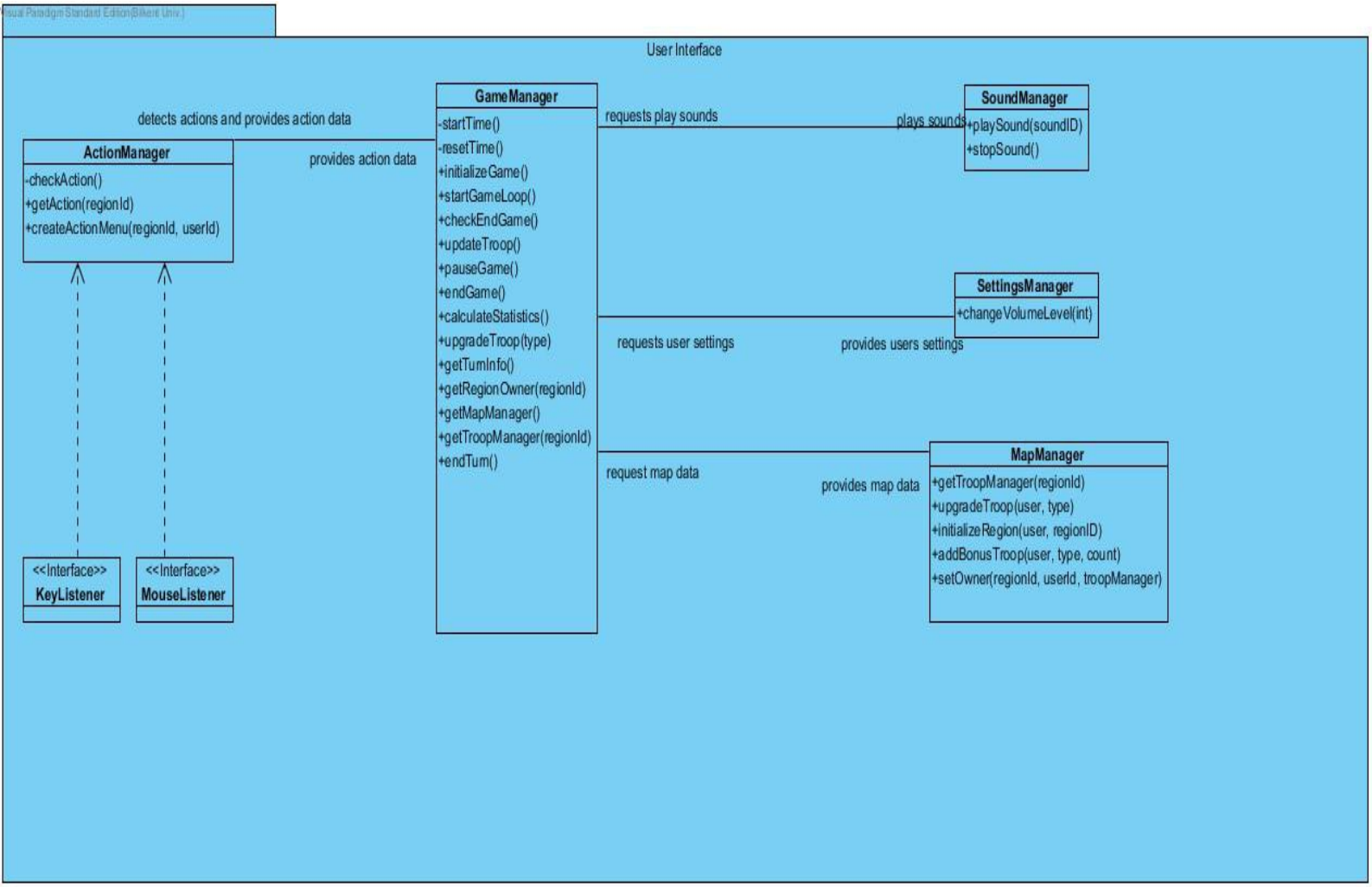
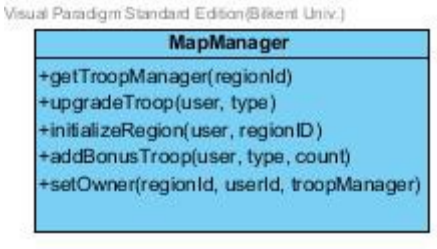


Figure 31 Game Control Subsystem of Conquest

MapManager class



Attributes:

private Region[] regions : Holds all the regions available on the game map.

private TroopManager[] troops: Holds troop information regarding the regions

Constructors:

public MapManager : initializes *regions* from the configuration file (note that this does not set any owners for the regions).

Methods:

public TroopManager getTroopManager(int regionId) : returns the current TroopManager residing in a region

public void upgradeTroop(User user, int t) : upgrades all Troops with type *t* that belongs to the *user*

public void initializeRegion(User user, int regionId) : sets an owner for a region with default number of troops

public void addBonusTroop(int regionId, int type, int count) : adds bonus troops with type *t* to the region

public void setOwner(User user, int regionId, TroopManager troop) : changes the owner of a region to *user,* effectively also replaces the old TroopManager with the new one.

UserManager

Attributes

private int userNum;

Constructors

UserManager(String name[] , int color[]): creates new users with the given names and colors.

Methods

public void updateStatistics(int userId, int killCount, int deathCount) : updates the kill and death statistics of given user

public User getUser(int userId) : returns the corresponding User object with id *userId*

WarManager

Attributes

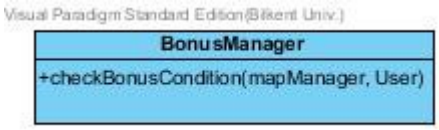
Constructors

WarManger(MapManager) : takes a MapManager to use in simulateWar method

Methods

public boolean simulateWar(int defenderId, int attackerId) : simulates a war between two sides and returns a boolean that indicates whether the attacker won or lost.

BonusManager



Attributes

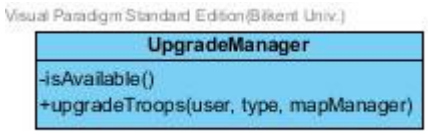
Constructors

BonusManager() : Default Constructor. Reads the configuration files and initializes bonus conditions and rewards.

Methods

boolean checkBonusCondition(Region[] region_list, User user) : iterates over the region list and determines whether the user deserves a bonus or not.

UpgradeManager



Attributes

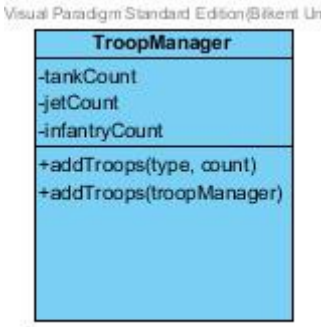
Constructors

UpgradeManager() : reads the configuration files and initializes the information regarding the upgrades.

Methods

public bool isAvailable(User u, int type) : checks whether the user can upgrade the type or not

TroopManager



Attributes

private int tankCount

private int infantryCount

private int jetCount

Constructors

TroopManager(int infantryCount = 1, tankCount = 0, jetCount = 0) : default constructor.

Methods

public void addTroops(int type, int count) : adds given number of troops

public void addTroops(TroopManager m) : adds an another TroopManager to the current one. Used in reinforcement

ReinforcementManager

Attributes

Constructors

Methods

public static void reinforce(int destinationId, TroopManager tm) : moves the given troops to the destination

SettingsManager Class

Methods:

public void changeVolumeLevel(int newVolumeLevel): This method takes an integer as parameter and changes the volume level of the game to this integer parameter.

SoundManager Class

Methods:

public void playSound(int soundID): This method takes an integer as parameter and this parameter is the id of the sound. After calling this method, requested sound with specific id will be played.

public void stopSound(): This method stops the sound which is currently being played.

5.3 Game Entities Subsystem

In this subsystem we have domain specific objects that we used.

User Class

Attributes:

private int color: This attribute of the User class holds the information of the color of the user. This color will paint the Regions of the user.

private string name: This attribute holds the user name of the user.

private int id: This attribute holds the user id which is a specific attribute for each user.

private int kill: This attribute holds the number of the soldiers which User has been killed.

private int death: This attribute holds the number of the soldiers which User has been lost during the wars.

private int numberOfRegion: This attribute is the number of the regions which belong to User.

private int experience: This attribute is the experience gained by the User during the wars and it will be used to make upgrades.

Methods:

public void updateExp(int experience): This method updates the experience of the user. It takes an integer as parameter and this parameter is the amount of the experience which will be given to the User.

public void updateStatistics(killCount, deathCount): This method updates the statistics of the User which are the amount of the kills and deaths User have been made. It takes two integer parameters and updates the kill and death attributes of the User.

Region Class

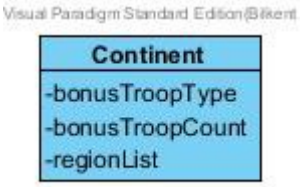
Attributes:

private int id: This attribute is the id of the region. It is unique for all of the Regions.

private int[] neighbor: This attribute is an integer array. In this integer array, there is the id's of the all neighbors of the Region.

private Point coordinate: This is an attribute for the MapManager class to draw the Region. This attribute holds the X and Y coordinates of the Region.

Continent Class



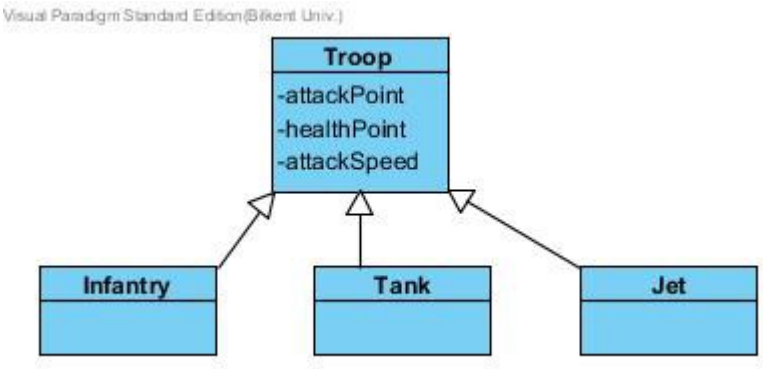
Attributes:

private string bonusTroopType: This attribute is to identify the type of the bonus troop when the User owns the all of the Regions on the Continent.

private int bonusTroopCount: This attribute is to determine the count of the bonus troop when the User owns the all of the Regions on the Continent.

private int[] regionList: This attribute is an integer array and this integer array holds the all of the id's of the Regions on this Continent.

Troop Class



Attributes:

private int attackPoint: This attribute determines the attack power of a Troop.

private int healthPoint: This attribute determines the health points of a Troop.

private int attackSpeed: This attribute determines the attack speed of a Troop. It will be used in the war simulations. If a Troop has a bigger attack speed it can hit more than other troops in one round.

- **Infantry Class**
- **Tank Class**
- **Jet Class**

These classes are the child classes of the Troop Class. They all have attackPoint, healthPoint and attackSpeed attributes.

6.Conclusion

The classic board game Risk is a good representative of the game Conquest, a turn based multiplayer strategy game, that invites to use their intelligence to dominate the world while having fun. It aims to deliver a reliable and fast gaming experiences while giving players to use different strategies against each other.

In this report the game Conquest, has been explored with all different accepts of it. Requirements have been extracted. Then these are analyzed to get a better engineering plan of the games system. After that system designed was examine in detail and finally object design has been added to report clarifying objects with design pattern applications, interface specifications and constraints.

To begin with, report starts with the general overlay of the game. Explaining the games aim, context and basic controls of the game. Games attack and defense mechanisms and basic game components are explained in detail.

According to games explanation a simple blue print of the game is extracted. This blueprint consists of functional and nonfunctional requirements, system constraints, some sample use case scenarios and some mockups of the basic screens and ui elements of Conquest.

After that Analysis part takes place. Class diagrams, Activity diagrams and Sequence diagrams are created and explained in detail. These different types of UML diagrams all give information about the game from different aspects of the system to form a unified system that consists all these features.

Then, object design is examined. Keeping system functionality, performance, coherence, extensibility and robustness in mind, system is reconsidered. And according to the need of the game appropriate design patterns are applied and explained in detail in the report.

This project has been a valuable experience towards basic object oriented design and working in groups. Through the timeline we have experienced some problems that we would experience in our real career. Helping us understand how a group of engineers should start a project from scratch and conclude in a timely and appropriately manner. Realizing how complex such a simple project can become easily it has gained us the problem solving ability which will be a key element that will shape our future careers.

7. References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] Wikipedia web site of the inspired "Risk" Game
https://en.wikipedia.org/wiki/Risk_(game)

[3]Webpage of the image of the Infantry Figure

http://hydra-media.cursecdn.com/heroesandgenerals.gamepedia.com/thumb/7/76/GE_Infantryman.png/144px-GE_Infantryman.png?version=8363820d90f27a3f3dce83922bbaf48a

[4]Webpage of the image of the Tank Figure
http://pngimg.com/upload/tank_PNG1310.png

[5]Webpage of the image of the Jet Figure

http://www.freelogovectors.net/wp-content/uploads/2011/07/jet-fighter-f-15.jpg

[6]Webpage of the image of the Map

http://www.americavstheworld.com/images/risk_big.jpg