Department of Computer Engineering

CS319 Term Project

CS319-3C-CA

Fall / 2019

Final Implementation Report

Team

Talha Şen 21702020
Hakan Sivuk 21601899
Rafi Çoktalaş 21601537
Cevat Aykan Sevinç 21703201
Yusuf Nevzat Şengün 21601720

Instructor: Eray Tüzün

Teaching Assistants: Barış Ardıç, Alperen Çetin

Table of Contents

1. Introduction

At the beginning of iteration 2, we lacked four functionalities: bots, multi-player, trading and development cards. Through the semester the team split into three sub-teams: Talha and Hakan focused on multi-player, Yusuf and Cevat investigated core logic and Rafi developed bots. Considering single-player and multi-player:

- **Local 4-Player Single-Player**

Every functionality players can perform have been implemented. The base game has been delivered completely. Furthermore, two additional development cards have been introduced. Nevertheless, the bot functionality we promised could not be implemented before the deadline of iteration 2 due to its complexity and time restrictions.

- **4-Player MultiPlayer**

Multi-player is fully implemented. Every functionality players can perform in single-player, can also be performed in multi-player. Players in Bilkent wifi environment can connect our game server and play the full Catan experience.

2. Design Changes

For iteration 2, after the demo presentations of iteration 1, the team has focused on improving the design and each design decision that was taken in design report have been implemented in the project. There was not any error while the team was developing/improving the project according to the iteration 2 design. Our design has not changed and fully implemented. Moreover, for development cards, strategy design pattern is introduced.

3. Lessons Learnt

Iterative/Agile Development

A software project consists of sub systems and each sub system has their own tasks. There has to be a schedule for delivering these tasks. Otherwise, when the deadline draws close, a feature has to be implemented with nothing prior has ever been done. For each week in the iteration, a schedule for delivering these tasks must be explicitly determined.

More Design, Less Trouble

Although we have made several meetings to discuss about the design of the project, more hours could be spent on it. In those meetings, we prepared scenarios, use case diagrams and analysis class diagram, however we have ignored issues such as dynamic models, subsystem design, detailed class diagram. Especially lack of subsystem design and detailed object design taught us a crucial lesson: design properly then start to implement to save time and effort.

Communication

Each person in the group has a chance to communicate with other members almost everyday. Therefore, each person in the group can update others about the status of his tasks or show product of his work. However, one should also inform others about some details about his task regularly so that people who are using his part can implement compatible codes with it. The lesson we learned from this issue is that the importance of the way we communicate. Communication must be open.

Importance of Structure

Although we have decided to use some design patterns, we should be more determined to follow these patterns. We have seen that if the pattern determined is not followed carefully, things can get messy easily. The lesson we learned from this issue is that if the design pattern is determined, stick on it carefully.

Importance of Test

We have realized different bugs while playing our game. Some of them are crucial and should be fixed, otherwise these bugs can result in serious problems. This situation shows the importance and necessity of the test stage. The lesson we learned from this issue is that after and during an implementation, the implementation should be tested carefully.

4. User's Guide

4.1 MainMenu

Fig. 1. Main menu when the game opens.

- **1:** Opens player selection for single-player.
- **2:** Opens multi-player login for multi-player.
- **3:** Displays detailed information about the game.
- **4:** Exits the system.
- **5:** Adjusts the master volume of the game.

4.2 Help

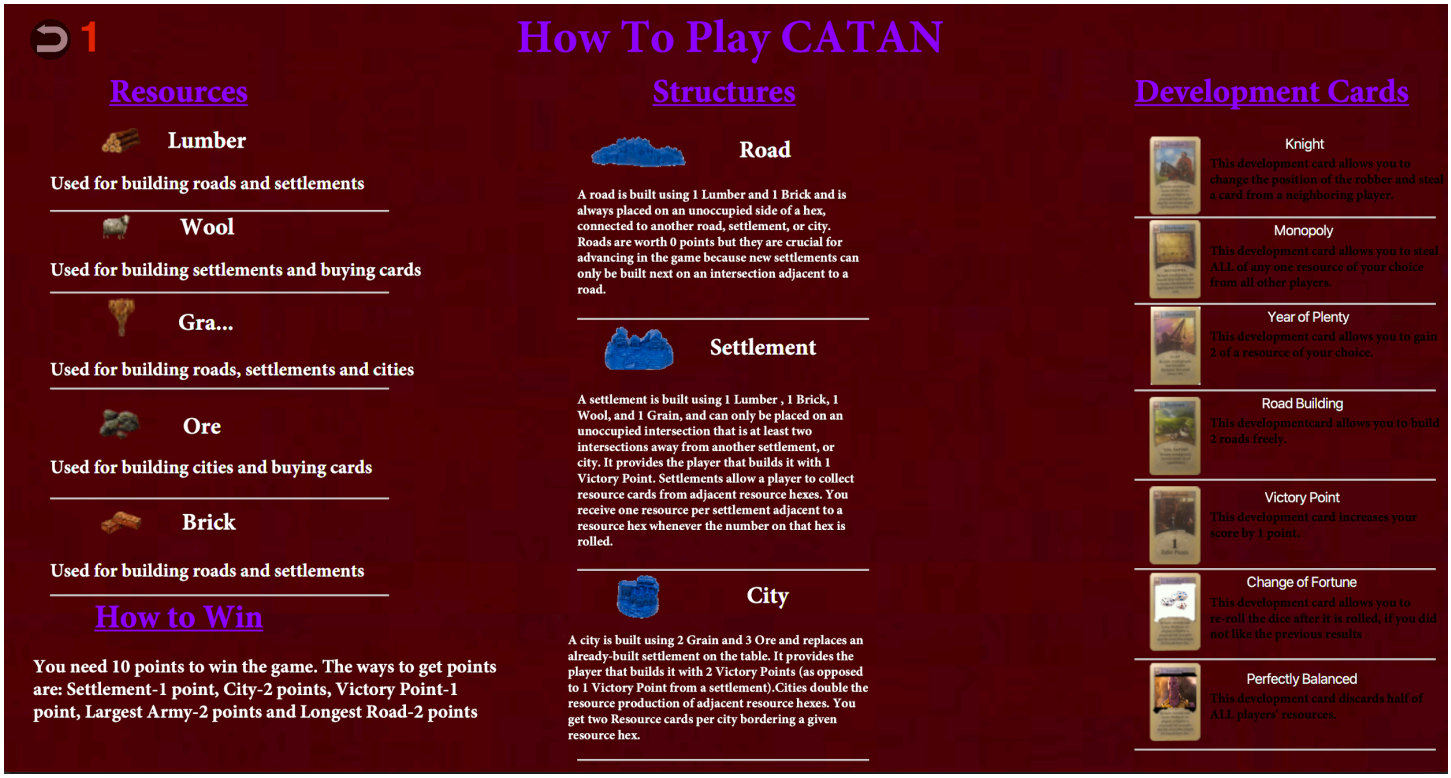

Fig. 2. Help accessed from main menu.

- ❏ **1:** Return to the main menu.

4.3 Player Local

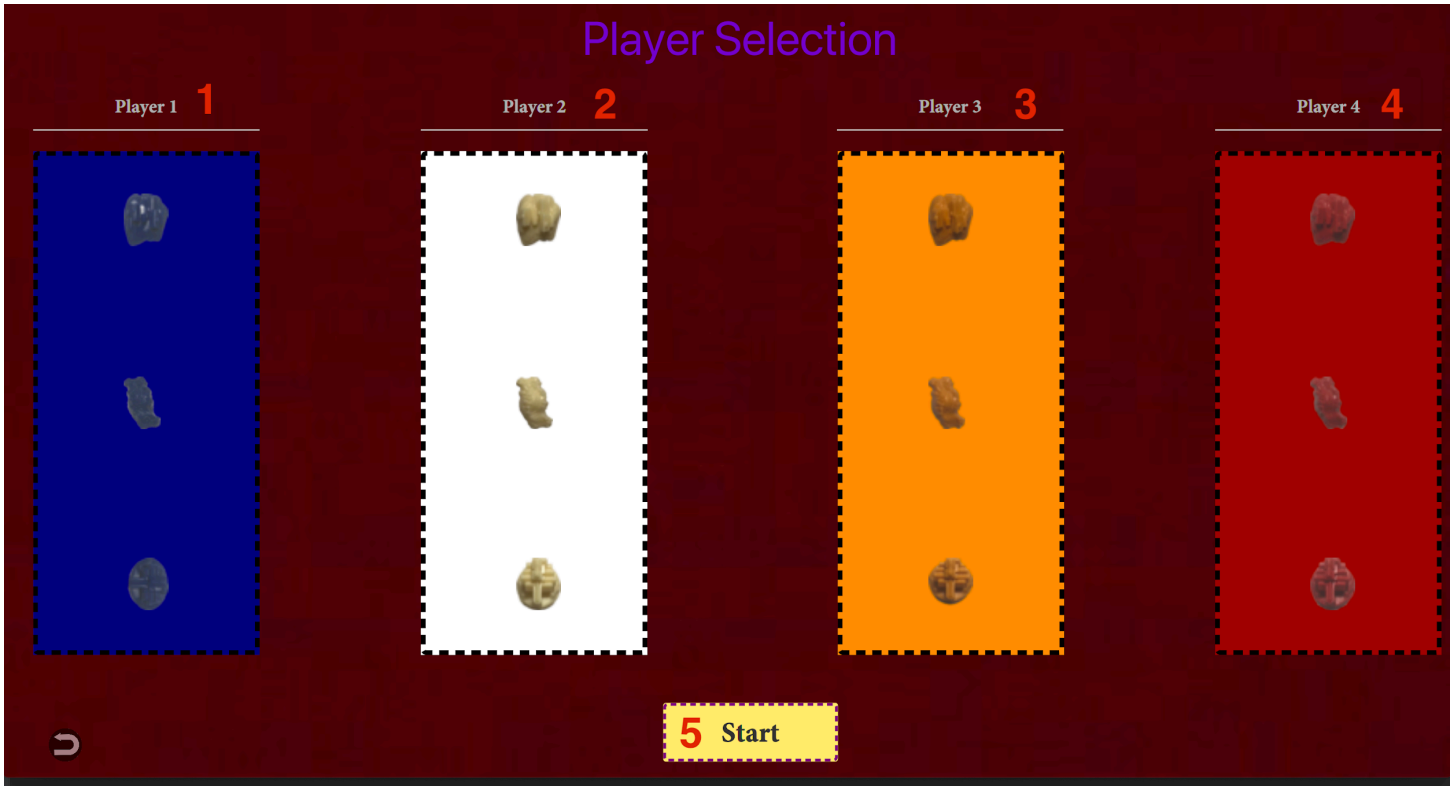

Fig. 3. Player selection for single-player.

- ❏ **1:** Player name for blue player, default is “BLUE”.
- ❏ **2:** Player name for white player, default is “WHITE”.
- ❏ **3:** Player name for orange player, default is “ORANGE”.
- ❏ **4:** Player name for red player, default is “RED”.
- ❏ **5:** Starts the single-player with the given or default player names.

4.4 Single-Player Game

Fig. 4, Single-player game.

- ❏ **1:** Area for game guide, legend.
- ❏ **2:** Opens the harbor and game trade options.
- ❏ **3:** Buys a development card for the player.
- ❏ **4:** Current player can see their information here.
- ❏ **5:** Gameboard.
- ❏ **6:** Status, displays information about the player actions.
- ❏ **7:** Other players' information, click to trade or hover over for detail information.
- ❏ **8:** Click to roll the dice.
- ❏ **9:** End the turn for the current player.
- ❏ **10:** Any development cards current player has will be displayed here.

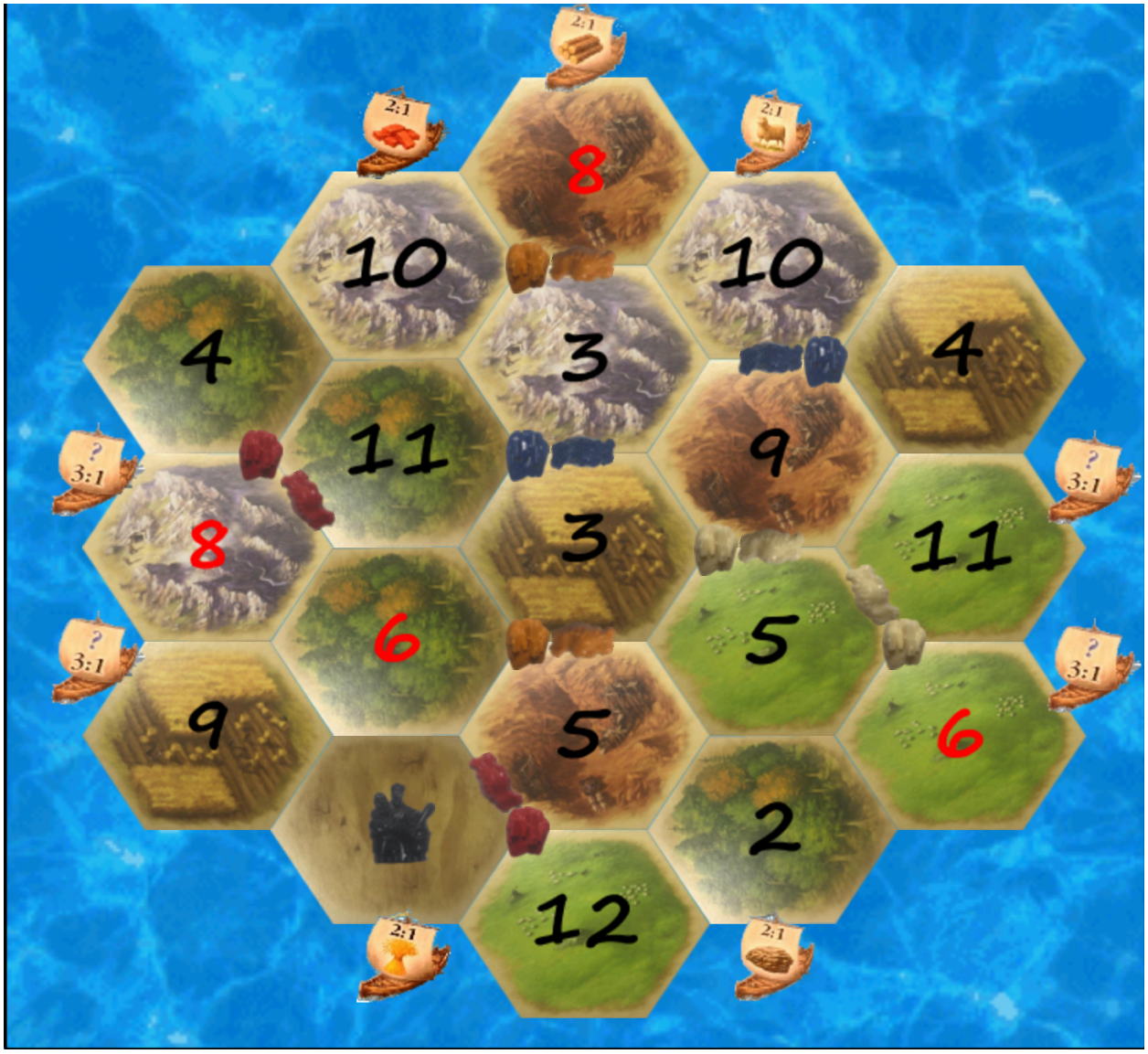

Fig. 5. Gameboard.

- **Interaction:** Corners of the hexagon are places where settlements/cities can be build if it is not conflicting. Sides of the hexagons are locations for roads. Players can interact with the board with their mouse. Simply click on anywhere on the gameboard and status bar will display if the intended action can be performed or not. Suitable places for constructing game pieces are highlighted when the mouse is hovered upon corners and sides. The robber can be dragged by clicking on the mouse on it and dragging it on to the selected hexagon before dropping it.

Figures 6 and 7. Highlighted settlement and built settlement.

Fig. 8. Detailed player Information when hovered over player circle.

- ❏ **1:** Displays the second in turn player's score and color.
- ❏ **2:** Displays the third in turn player's score and color.
- ❏ **3:** Displays the fourth in turn player's score and color.
- ❏ **4:** Displays the hovered player circle's detailed information.

Fig. 9. Trading popup when clicked on other player circle.

- ❏ **1:** Adjusts the resources that are offered by the current player.
- ❏ **2:** Adjusts the resources that are wanted from the other player.
- ❏ **3:** Sends trade offer to the other player.
- ❏ **4:** Cancels the trade (players can also click on other game areas to close popup).

Fig. 10. Trading with game popup when clicked on the trade with game button.

- ❏ **1:** Choose the resource for trade offer.
- ❏ **2:** Choose the resource to gain after the trading.
- ❏ **3:** If any harbor is owned, trade option for them will be available.
- ❏ **4:** Exits the trade popup (Players can click on elsewhere to exit too).

4.5 Multiplayer Login

Fig. 11. Multi-player login screen entered from play multi-player.

- ❏ **1:** Enter registered account username.
- ❏ **2:** Enter registered password for username.
- ❏ **3:** Create a new account.
- ❏ **4:** Click to login into multi-player for matchmaking.
- ❏ **5:** Enter registered account username.

4.6 Profile Creation

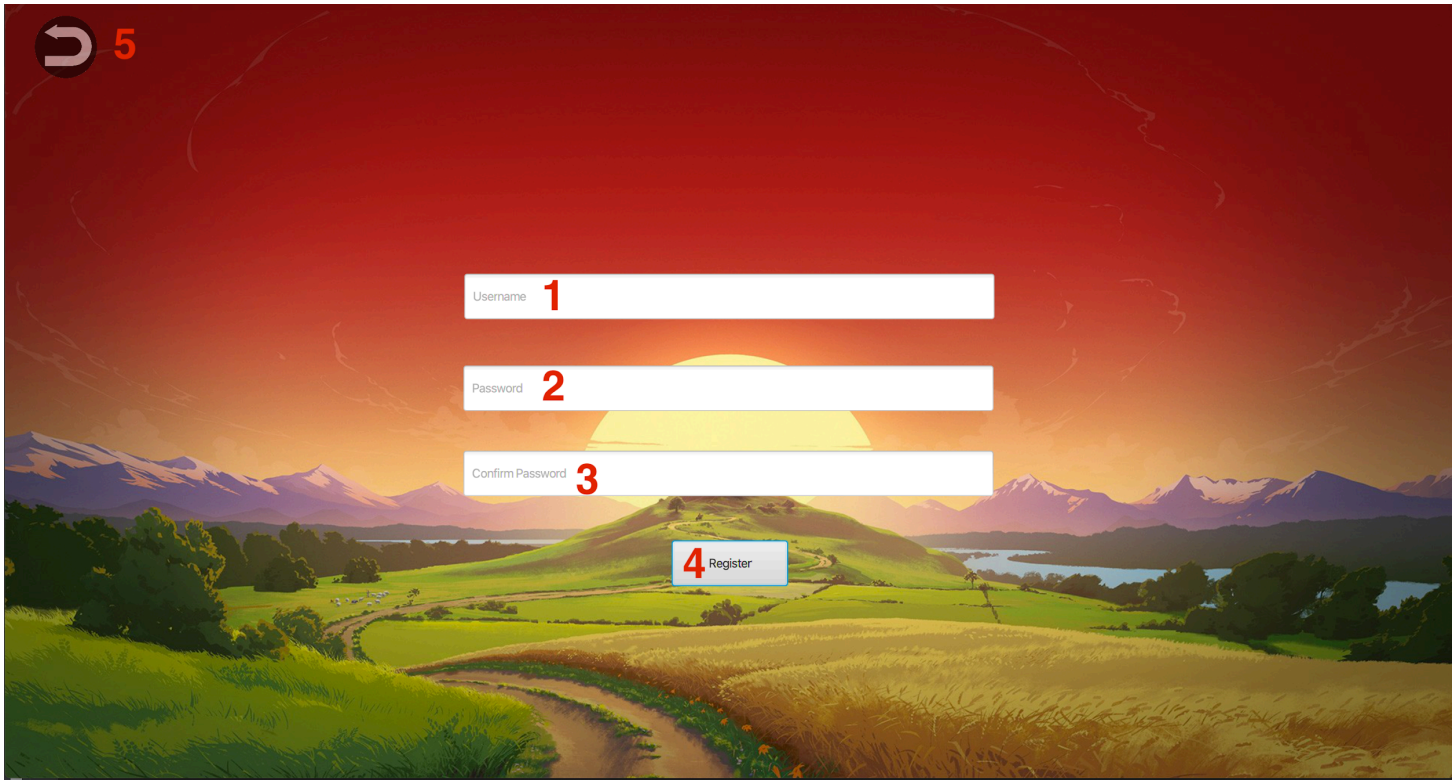

Fig. 12. Creating a profile for multi-player.

- **1:** Username for registering.
- **2:** Chosen password for the account.
- **3:** Password check field.
- **4:** Register the indicated account.
- **5:** Return to the login screen.

4.7 Matchmaking

Fig. 13. Searching for an online match 1.

- ❏ **1:** Finds online games when clicked.
- ❏ **2:** Returns to the login screen.

Fig. 14. Searching for an online match 2.

- ❏ **1:** Displays the number of players left to start an online game.

❑ **2:** Return to the start matchmaking.

4.8 Multi-Player Game

Fig. 15. Multi-player game.

Multi-player screen is not different than single-player screen with only two differences:

❑ **1:** Opens chat when hovered to communicate with other players.
❑ **2:** Always displays the local player's information.

Multi-player can only be played in Bilkent internet environment. Outside Bilkent, multi-player can be played with a VPN.

5. Build Instructions (System Requirements & Installation)

Since the project is developed in Java, it can run as a desktop application on MacOS and Windows platforms. The project was developed in IntelliJ IDEA (version 2019.1.3). Therefore this manual will refer to building the project on IntelliJ IDEA. Steps to build the project are as follows:

- Download the project folder from the Github. Once downloaded, open IntelliJ IDEA and choose to create a new project. Choose JavaFX application and select Java SDK as 14 or 11 as 13 may cause conflicts. Choose the downloaded CS319-3C-CA project folder as the source. Project name should be CS319-3C-CA after the selection. Click finish and click on yes everything that are prompted. Project must be opened by the IDEA now

- Download javaFX from this link: https://gluonhq.com/products/javafx/ (Download 11.0.2 for your specific operator)
- Put the javafx folder inside zip to your Catan project (place is your choice, but make sure it is okay)
- Open up Intellij
- Go to File > Project Structure > Global Libraries
- Click on the plus symbol on the upper left and click Java
- From the file selector window, go to the javafx folder you downloaded and put, go to lib and select every file except the src zip and the properties file (so select every executable file)
- Click OK on file selector
- Right click the module you just added and click "Add Module to the Project" (or something similar I don't remember exactly), click Apply on bottom right and then you are done with modules.
- For the files you want to run (In our case the GameStart, but you may have Test UI files you want to run) go to Run > Edit Configurations and select your main class. For its VM Options, copy paste this in there (except change the part I marked below): --module-path **PATH TO YOUR JAVAFX FOLDER**\javafx-sdk-11.0.2\lib --add-modules javafx.controls,javafx.fxml
- Click Apply then OK. You are done with JavaFx
- In Intellij go to File > Project Structure > Global Libraries and click the plus sign on the upper left and click Maven
- Copy and paste this to the text box:

```
<dependency>
 <groupId>io.github.typhon0</groupId>
 <artifactId>AnimateFX</artifactId>
 <version>1.2.1</version>
 <type>pom</type>
</dependency>
```

- Click OK, it will download it for you
- You are done with installing AnimateFX
- Open edit configuration of GameApplication
- At VM options, add javafx.media to the end of the line
- Click OK
- Go to https://jar-download.com/artifacts/io.socket/socket.io-client/1.0.0/source-code
- Download socket-io (1.0.0)
- In the zip files you just downloaded, there are 5 jar files, put them in a folder and put the folder in a place of your choice (make note of path)
- In Intellij, go to File > Project Structure
- Go to libraries
- Click the plus sign, select java and select all 5 jar files you downloaded.
- Click ok and add the jar files to the modules.
- Click apply and ok.
- Now, you should be able to build and run the project.

After the dependencies are installed, project can be run through the main method of the GameApplication class which can be found in the package: CS319-3C-CA.src.SceneManagement.GameApplciation. Simply right click Application class and choose “Run GameApplication.main().”

Server side of the project was implemented by node js and mongodb was used as a database. Therefore, in order to run the server you should have them on your computer. Steps for installing them as follows:

- You should first install node js on your computer. Go to the link https://nodejs.org/en/download/ and download the one proper for your operating system.
- Open the downloaded file and complete the installation by following steps.
- After installation, you can check if it is installed successfully by typing your terminal “node -v”. If it gives the version, it means you completed installation successfully.

- Now install mongodb.
- For Windows operating system:
    - Go to the link https://www.mongodb.com/download-center/community and download the setup file.
    - After it is downloaded, open it. You can follow the steps straightforward but be careful about choosing complete setup, running service as network service user, and not selecting to install MongoDB Compass.

- For Macos:
    - Install XCode which can be found on the App Store.
    - Install homebrew by typing the following command on terminal: /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" or you can find other options on official website: https://brew.sh/index_tr
    - Type this command: brew tap mongodb/brew
    - Type this command: brew install mongodb-community@4.2
    - You can check whether the installation is successfully completed by typing mongod --version on terminal.
    - Also go to your user directory and create mongodb/data/db directories. In other words, when you type pwd command in db directory you should see /Users/your_user_name/mongodb/data/db
    - When you want to run the server, you should run mongodb manually. You should run it by typing “mongod --dbpath=/Users/your_user_name/mongodb/data/db" command on terminal.

- Finally, installations are completed. Now come to the backend directory of the project on terminal.
- Type “npm install” command to install the necessary packages.
- After package installation is completed, you can run the server by typing “npm run start”. Now server is running.

6. Work Allocation Through Semester

❑ Talha Şen

At the start of the project I implemented the Card and Player classes for the game. I also designed the view and controller system for our MVC design at the start. I then started to implement both UIs and controllers. I implemented the UIs in fxml and css files and I implemented the controllers in a single controller class. After the first iteration, we had a design rework and we also had multiplayer going on so I split controllers to the respective scenes, singleplayer/multiplayer types and to their game related sub controllers. I implemented the additional controllers for single player then I started working for the multiplayer with Hakan. Hakan already implemented the backend so we prepared the frontend and edited the backend according to that. I designed and implemented multiplayer views and controllers. I also implemented the SoundController class which controls the whole music and effect sound flow of the game with Hakan. At the end, I mostly worked on bug fixes and tests for singleplayer, multiplayer, UIs and the server. For the analysis reports of iteration 1 and 2 I made the dynamic diagrams, and for the design report I made the ui and controller class diagrams, subsystem decomposition and in iteration 2 I made the server diagrams and subsystem decomposition with Hakan.

❑ Hakan Sivuk

At the beginning of the project I implemented some basic classes for the logic of the game. Also I had small roles in some parts of the design plans. Apart from these, my main work was on multiplayer part of the project. I made researches about server implementations, authentication, communication between client and the server throughout the game, database systems and so on. As a result of these researches, I designed backend server and server related parts of the frontend. By using these designs I implemented backend server and since Talha has experiences in terms of the logic of the game and UI, together we implemented frontend parts. Also we added sounds to the game with Talha. I prepared several kinds of diagrams for reports especially ones related to multiplayer or server part of the project. Through the end of the project, I worked on resolving mistakes and making the project bug free.

❑ Rafi Çoktalaş

At the beginning, I take place in the general low level design plan we follow throughout the project. For every modification in the design choices, I investigated the possible negative effects. Moreover, I implemented the structure classes.

Negative effects of the design choices result in a design rework. After the design rework, as all the choices are constructed with greater care, I started to design the AI algorithm. I have finished the algorithm right before the completion of the design rework. In the end, an unexpected amount of bugs we encounter pushed us to spend a great time on both analyzing and solving these bugs. Therefore, the bot player is put on the shelf. Thus, I also allocate my time to solve these bugs. The way in which I constructed the AI algorithm pave the way for me while I investigated the bugs as the AI algorithm requires me to look through all the possibilities each scenario has. Even though the AI did not become a feature, it proves to be beneficial in the bug fixing process.

❏ **Cevat Aykan Sevinç**

I was the project manager of group 3C-CA. I strictly followed deadlines and arranged meetings for the group. Furthermore, I helped my team with managing GitHub and almost always giving code reviews. Next, I played a great role in writing analysis, design and implementation report as I had a great understanding of the development of our project except multi-player development. I have been involved in the development of each single-player implementation. During implementation, my main responsibility was to connect the logic in single-player my teammates have provided. Later, I have also been involved in creating GUI for trading with players and game. Last, I have tested the game and fix bugs consist of core logic and GUI.

❏ **Yusuf Nevzat Şengün**

At the beginning of the project, I implemented the core classes of the logic according to our beginner level design that we have decided as a group. After every person in my team implemented their basic classes, I have merged them in order to maintain the different scenarios. After this time I behaved as the software architecture of my team. For integration of the code, I reworked on almost every class and made necessary changes. I followed, commented and gave idea about updates on model usage and operations. I also applied and implemented different design patterns and ideas to our project, such as mvc, facade, singleton and strategy design patterns and also I reworked on other parts of the project and redesigned these parts into more OOP related versions. I also implemented and debugged some features about UI and prepared some basic diagrams and wrote some parts for the reports. At the end, I have tested and debugged several parts of the project.