Alen Gracanin

Professor X

Capstone Project

May 2024

Capstone: Development and Reflection of an Online Version of the Risk Board Game

I. Introduction

The project undertaken for this assignment was the development of the classic board game “Risk”, a turn-based strategy game. The main objective of the project was to create an interactive and online version of the game that could be played on a web browser with a focus on key elements such as territory control, strategic troop management and engaging battle mechanics.

The scope of the project included developing both the frontend and backend components of the game, implementing real-time multiplayer features, a chatroom for player communication and creating a game board with an interactive map for players.

Initially developed as part of a group assignment for a software engineering course at X university, this project has evolved further after course completion. This paper will reflect on various aspects of the project, including the development process, challenges encountered, solutions implemented, and key learnings gained. Additionally, it will explore the application’s overall functionality, potential improvements, areas for further enhancement, and discuss the updates made after the initial completion to advance the project into a fully deployable state.

II. Project Description

The project was developed to model the classic board game "Risk," incorporating all of its unique key features. The game contains an interactive map divided into territories where players compete for global domination by strategically deploying their troops, attacking opponents, and reinforcing their positions.

Risk is a turn-based game where players move through three different phases during their turn: draft, attack, and reinforce. During the draft phase, a player receives a number of troops to deploy on a territory they own. The number of troops received can be increased by owning all territories in a continent. The attack phase allows players to launch attacks against neighboring opponent territories. The reinforce phase allows players to move their troops between owned territories, provided there is a continuous path of player-owned territories connecting them. Troops cannot cross through opponent-owned territories.

The game features an interactive interface where players can click on territories to execute their actions, including real-time multiplayer functionality. The interface was designed to be visually appealing and engaging, with vibrant colors representing territories and players, indicating turn status and game state. Additionally, user interface elements indicate the game's current state, and a synchronized timer displays turn duration.

III. Development Process

3.1 Technologies Used

The project was developed using modern web technologies and development tools upon analysis of requirements for the project and considering individual familiarity with available technology the stack to use for development was decided upon. The frontend and backend were fully built with JavaScript as the main programming language. The technologies, frameworks and tools used include:

- **Frontend:** The frontend was built using the Vue.js framework, which provided a reactive and component-based architecture.
- **Backend:** The backend was developed using Node.js as the runtime environment, in combination with Express.js for routing, which allowed for handling of real-time events through WebSockets.
- **Libraries and Modules**: Various libraries and modules were used to help with project functionality. A notable library, Socket.IO was used to utilize WebSockets for bi-directional communication. A notable module used was `*crypto*` providing the `*randomUUID()*` method which was used for generation of secure game keys.
- **Version Control:** Git was used for version control allowing for collaboration and management of code changes.

3.2 Technical Architecture

The backend was organized around a central Game class that was responsible for managing the game state, player actions, and game flow. All game data was modeled using data structures for territories, players, and continents. Upon creating a new game, a new class instance is initialized with a UUID as the game key and stored in a dictionary-like object, allowing for

multiple concurrent games differentiated by their keys. The following code snippet demonstrates the core data structures used in the backend of the game:

Figure 3.2.1: Data structure overview

```
'https://ourgame.com/83930-dfnsd-sjfd90d9'

const game = {
    continents: [{}],
    territories: [{}],
    players: [{}],
    cards: [{}],
    currentTurn: 1, // playerID
    uuid: "",
}

const continent = {
    id: 0,
    name: "North America",
    territory_count: 9,
    player: undefined,
    bonus: 5
}

const territory = {
    id: 0,
    continent: 0,
    player: undefined,
    name: "Alaska",
    troops: 0,
    connections: [1, 2, 36]
}

const player = {
    id: 1,
    username: "",
    party_leader: false,
    color: "",
    alive: true,
    cards: [{}],
    troops: 0,
    deployable_troops: 0,
    territories: 0,
    turn_state: 1,
}
```

The frontend of the project was structured using Vue.js, which provided a component-based architecture for modular development and reusability. This approach allowed for improved structure and maintainability. Each feature of the interface; such as the lobby, chat, and game page, was broken down into components. Necessary data was passed between

components using Vue's refs and props. Additionally, Pinia was used for state management. This combination made it easy to create a dynamic interface that was constantly updated based on the state of the game or changing data which allowed for achieving the desired goals in player interactions, real-time updates, and dynamic UI rendering, overall contributing to a smooth and engaging user experience.

3.3 Challenges and Solutions

While in development, the project presented several challenges that required innovative solutions. One major challenge throughout the project was working with unfamiliar technologies. Going into the project, I had no experience developing a frontend with frameworks, so working with Vue.js was completely new to me, which posed issues during development. The solution involved extensive research, reading documentation, and utilizing online resources to learn common approaches to problems and tasks. Over time, I grasped the fundamentals and continued to learn deeper concepts, which increased development efficiency.

When developing a battle system, it was challenging to devise an algorithm that was fair and rewarded well-strategized attacks. The solution involved developing various functions that incorporated both randomness and strategy. We introduced randomness with dice roll simulations and considered troop counts for the territories involved in the battle. The troops continuously fight one-on-one until a territory has none remaining. The outcome of each battle is based on the higher dice roll. However, this approach led to true randomness, which was not ideal as it allowed attackers to succeed without worrying about having sufficient troop counts, while also punishing well-strategized attacks due to the same randomness.

To address this, we modified the probability to give defending troops a 60% chance of winning each one-on-one battle. This adjustment encouraged smarter decision-making from an attacking standpoint. To balance this advantage for attackers, we calculated the number of troops needed to guarantee a win based on the modified probability. If the attacker had this many troops but lost the battle, they would still capture the territory but only have one troop remaining. This approach balanced strategy in both attacking and defending. The function for this logic can be seen below.

```
Blitz(attacking_territory, defending_territory, attacking_troops) {
      const attacking_player = this.players[attacking_territory.player]
      const defending_player = this.players[defending_territory.player]
      let defending_troops = defending_territory.troops
      // Remove Attacking Troops From Current Territory
      attacking_territory.troops -= attacking_troops
      // Number of Troops To Guarantee A Win
      const attackingTroopsNeeded = Math.ceil(defending_troops / .4167) + 1
      const gauranteedAttackerWin = attacking_troops >= attackingTroopsNeeded
      let attacker_losses = 0
      let defender_losses = 0
      while(true) {
          const roll = this.RollDice(10000)
          console.log(`Blitz Roll: ${roll}`)
          // Defender Wins Roll
          if (roll > 4167) {
              attacking_troops--;
              attacker_losses++;
          }
          // Attacker Wins Roll
          else {
              defending_troops--;
              defender_losses++;
          }
          //  Attacker Wins
          if (defending_troops <= 0) {
```

```
            defending_territory.player = attacking_player.id
            defending_territory.troops = attacking_troops
            attacking_player.troops -= attacker_losses
            defending_player.troops -= defender_losses
            return
        }
        // Defender Wins
        if (attacking_troops <= 0) {
            if (gauranteedAttackerWin) {
                defending_territory.player = attacking_player.id
                defending_territory.troops = 1
                defending_player.troops -= defending_troops
                attacking_player.troops -= attacker_losses - 1
                return
            }
            // Adjust Troops
            defending_territory.troops -= defender_losses
            defending_player.troops -= defender_losses
            attacking_player.troops -= attacker_losses
            return
        }
    }
}
```

When working on the game logic, debugging and testing the code became an issue. Setting up a game and reaching the specific state needed to test the logic was very time-consuming. To address this, we began to develop tests for our code to simulate different game environments, allowing us to save time, prevent bugs and perfect game logic across various scenarios.

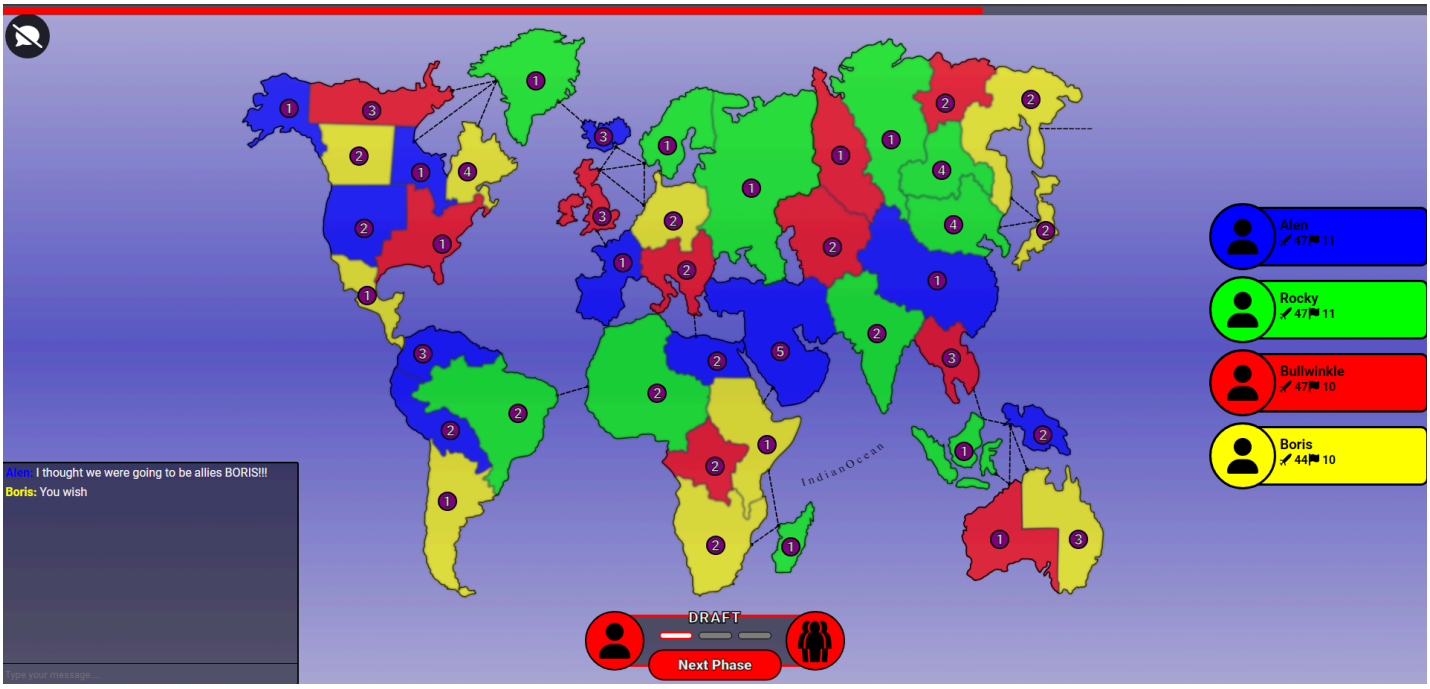
3.4 Post-Completion Updates

After the initial completion of the course, further development was done outside of the group and course to advance the game into a more polished state. One of the first key updates

focused on frontend, specifically the in-game interface. Player colors were improved to be more vibrant and distinct, timer and turn controller colors were synchronized to match the respective player turn, icons were added and changed to make the game more visually appealing, font updates to improve visual clarity in certain areas. Additionally, a hide chat feature was implemented to reduce clutter on screen specifically targeting responsiveness for smaller display devices.

For enhancing gameplay transparency and strategy, the game was missing a troop tracker display for other players to know troop count's of opponents across the map. This was implemented on player cards along with a territory tracker to boost the player's ability to make strategic decisions based on game data. The figure below shows the state of the game's interface with all of the changes implemented.

Figure 3.4.1: Game board



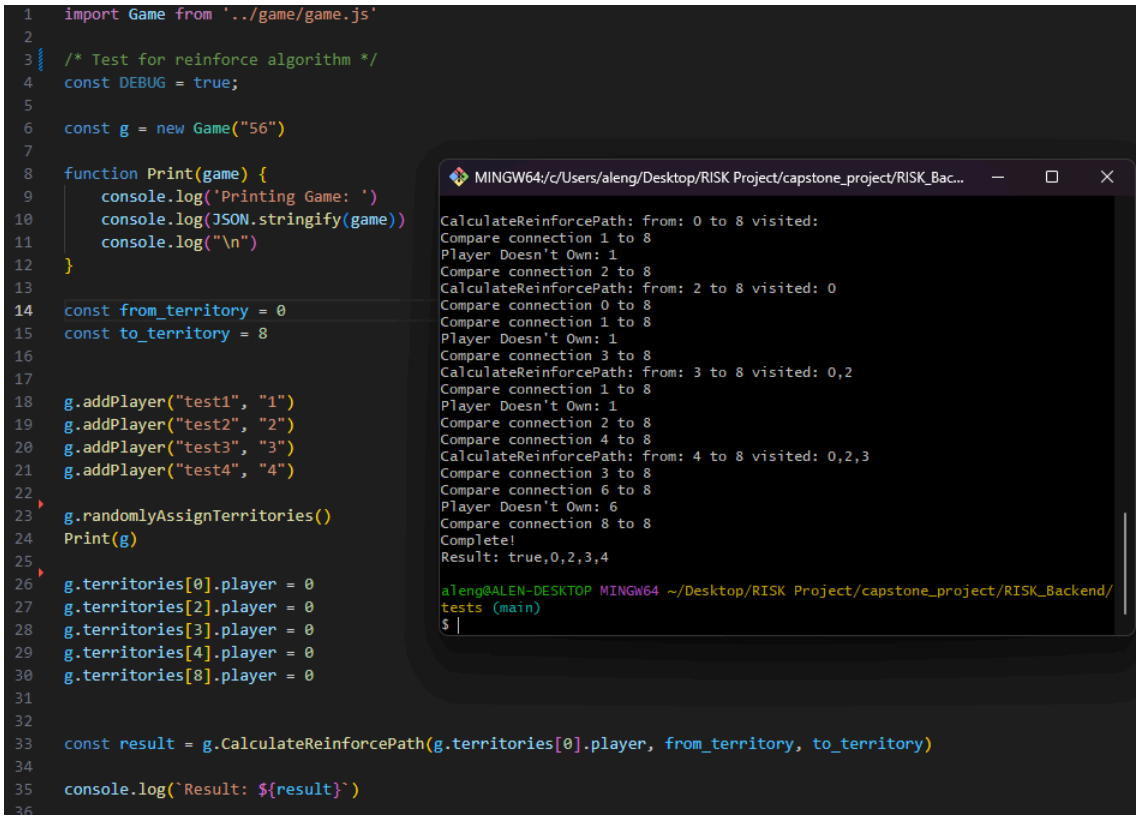
Furthermore, the reinforcement algorithm was refined and created to operate more consistently and efficiently. The algorithm was optimized to handle all reinforcement scenarios allowing for all valid connections to be traveled through. The updated algorithm, shown in the figure below, utilizes a breadth-first search approach to determine a valid path between the territories.

Figure 3.4.2: Reinforce BFS Algorithm

```
CalculateReinforcePath(player_id, from_territory, to_territory, visited=[]) {
    console.log(`CalculateReinforcePath: from: ${from_territory} to ${to_territory} visited: ${visited}`)
    visited.push(from_territory)
    const f_territory = this.territories[from_territory]
    let result = false
    for(let conn of f_territory.connections) {
        console.log(`Compare connection ${conn} to ${to_territory}`)
        if (this.territories[conn].player !== player_id) {
            console.log(`Player Doesn't Own: ${conn}`)
            continue;
        }
        if (conn == to_territory) {
            console.log(`Complete!`)
            return [true, visited]
        }
        const search = visited.find(element => element == conn)
        if (search == undefined) {
            result = this.CalculateReinforcePath(player_id, conn, to_territory, visited)
            if (result) break;
        }
    }
    return result
}
```

Accompanying this, I wrote a test case to evaluate its behavior in various custom scenarios to make debugging easier. In the figure below, a test case is shown calculating if Alaska to Greenland is a valid path for reinforcement along with the output.

Figure 3.4.3: Test case for reinforce



```
import Game from '../game/game.js'

/* Test for reinforce algorithm */
const DEBUG = true;

const g = new Game("56")

function Print(game) {
    console.log('Printing Game: ')
    console.log(JSON.stringify(game))
    console.log("\n")
}

const from_territory = 0
const to_territory = 8


g.addPlayer("test1", "1")
g.addPlayer("test2", "2")
g.addPlayer("test3", "3")
g.addPlayer("test4", "4")

g.randomlyAssignTerritories()
Print(g)

g.territories[0].player = 0
g.territories[2].player = 0
g.territories[3].player = 0
g.territories[4].player = 0
g.territories[8].player = 0


const result = g.CalculateReinforcePath(g.territories[0].player, from_territory, to_territory)

console.log(`Result: ${result}`)
```

```
MINGW64:/c/Users/aleng/Desktop/RISK Project/capstone_project/RISK_Bac...

CalculateReinforcePath: from: 0 to 8 visited:
Compare connection 1 to 8
Player Doesn't Own: 1
Compare connection 2 to 8
CalculateReinforcePath: from: 2 to 8 visited: 0
Compare connection 0 to 8
Compare connection 1 to 8
Player Doesn't Own: 1
Compare connection 3 to 8
CalculateReinforcePath: from: 3 to 8 visited: 0,2
Compare connection 1 to 8
Player Doesn't Own: 1
Compare connection 2 to 8
Compare connection 4 to 8
CalculateReinforcePath: from: 4 to 8 visited: 0,2,3
Compare connection 3 to 8
Compare connection 6 to 8
Player Doesn't Own: 6
Compare connection 8 to 8
Complete!
Result: true,0,2,3,4

aleng@ALEN-DESKTOP MINGW64 ~/Desktop/RISK Project/capstone_project/RISK_Backend/
tests (main)
$
```

IV. Project Deliverables

The project successfully developed a fully functional, multiplayer, web-based version of "Risk." The final product allows a user to host a game and wait in a lobby with a chatroom, where players can either be invited or join through a game list. Once a minimum of three players are connected, the host can start the game.

At the beginning of the game, each player receives twenty troops that are randomly deployed across all available territories. Each player is given a seventy-second turn, during which they go through three phases: draft, attack, and reinforce. This cycle repeats until a player loses all their territories, at which point they are removed from the game. The player who manages to own all territories on the map is declared the winner. The game will continue until a

winner is determined, or, in the case of inactivity among remaining players, the game will automatically shut down after six hours.

The following figures show the flow of the game from start to finish, highlighting all key features developed (See Appendix B1)

V. Reflective Conclusion

5.1 Reflection and Learning

Working on this project was a valuable learning experience that provided me with many insights into software engineering. I learned how to design and implement a full-stack web application from scratch, integrate real-time client-server communication, manage state across an application, and experience what it's like to be part of a development team. I further developed my knowledge of technologies and now have a deeper understanding of what a frontend JavaScript framework can do.

Breaking everything down into components and tackling such a large project taught me how to plan and approach overwhelming tasks. I learned that breaking things down into smaller, manageable parts makes complex projects easier to handle, and this is a process I will use for the rest of my career. Most importantly, this project taught me to be confident in my ability to learn new things. I learned that even when things are difficult, stressful, and challenging, they are still possible. This experience has shown me that the best way to grow my skills is through tackling new and unfamiliar challenges.

If I were to start the project over, I would focus more on initial planning, especially outlining how the game would work from start to finish. I feel a lot of time was wasted between tasks figuring out how the game should function, which could have been avoided with proper

preparation. This also includes frontend design, where using mockups to plan the layout would have saved a lot of time. I spent too much time tweaking CSS/HTML because I didn't like the structure after everything was done. With mockups, I could have focused on design separately from technical implementation. I would also have utilized testing earlier and more frequently to perfect game logic and ensure data structures were set up correctly. This approach would have prevented many bugs and made development more straightforward, resulting in cleaner code and increased productivity.

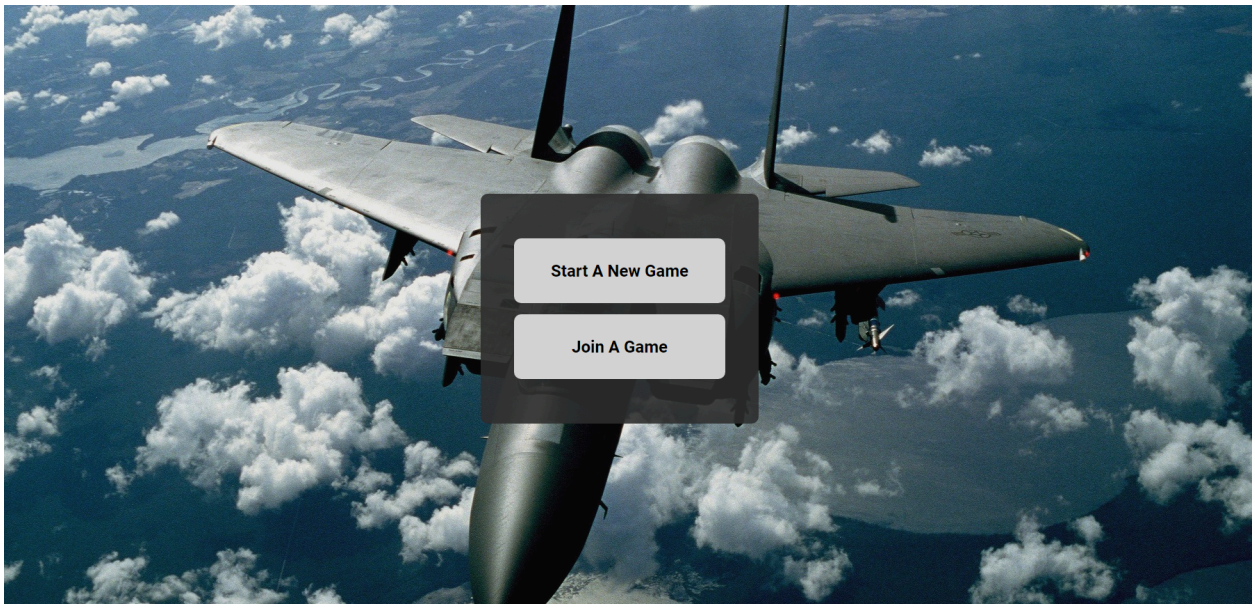
5.2 Future Enhancements

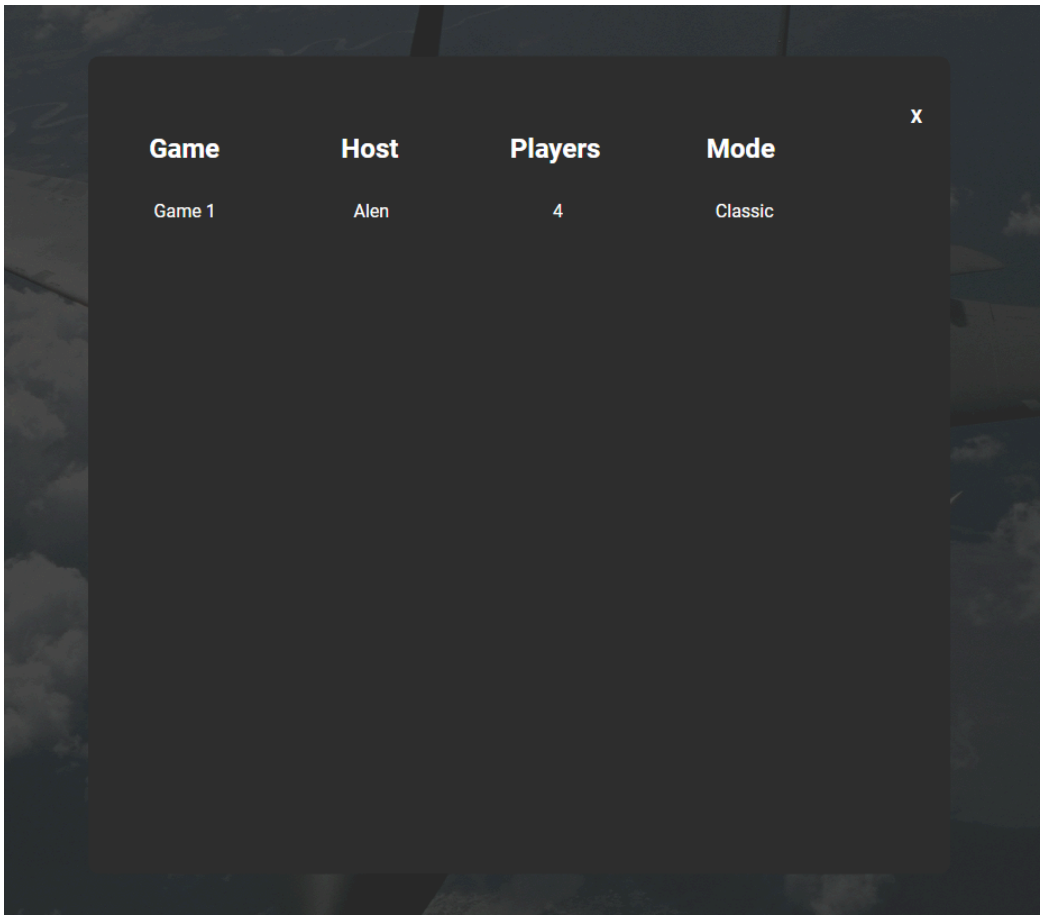
I believe the project is well set up for future enhancements and is scalable in almost every way. Moving forward, enhancements could include further refining the battle algorithm to incorporate more strategic depth and further reduce randomness. Animations could be added to the map, although it's currently tedious with the existing technology stack. Incorporating a game engine designed for web games, such as WebGL or Babylon.js, could improve the user experience by allowing more in-depth customization of the map. Additionally, incorporating AI opponents using an algorithm such as a Monte Carlo tree search could add an interesting and challenging dynamic to the game.

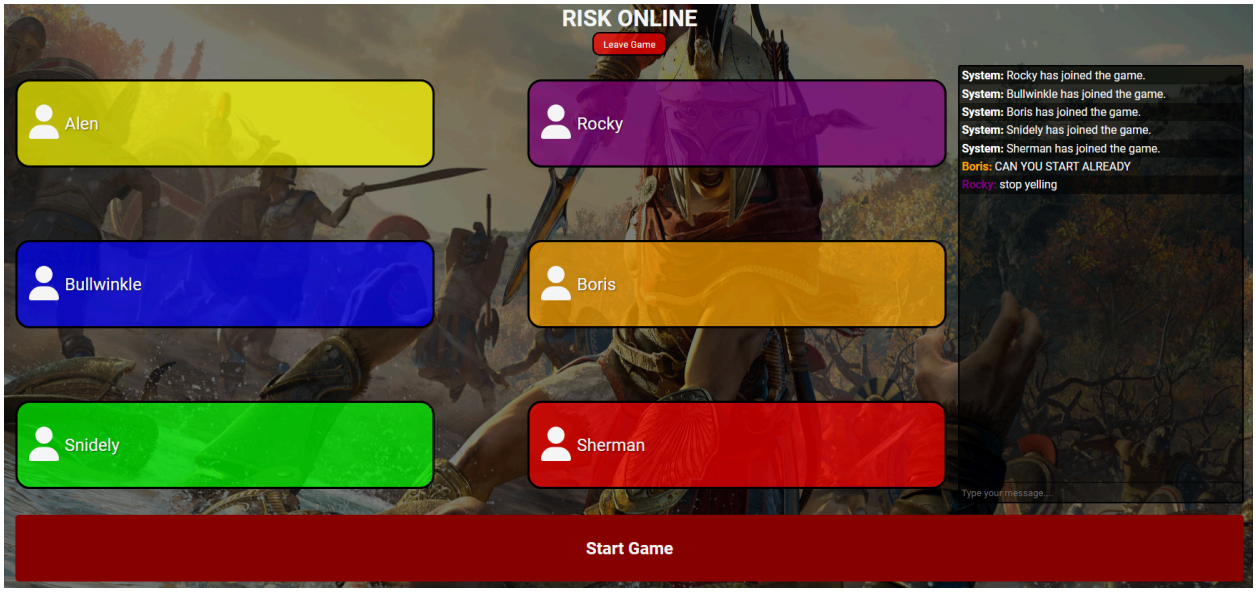
VI. Appendix

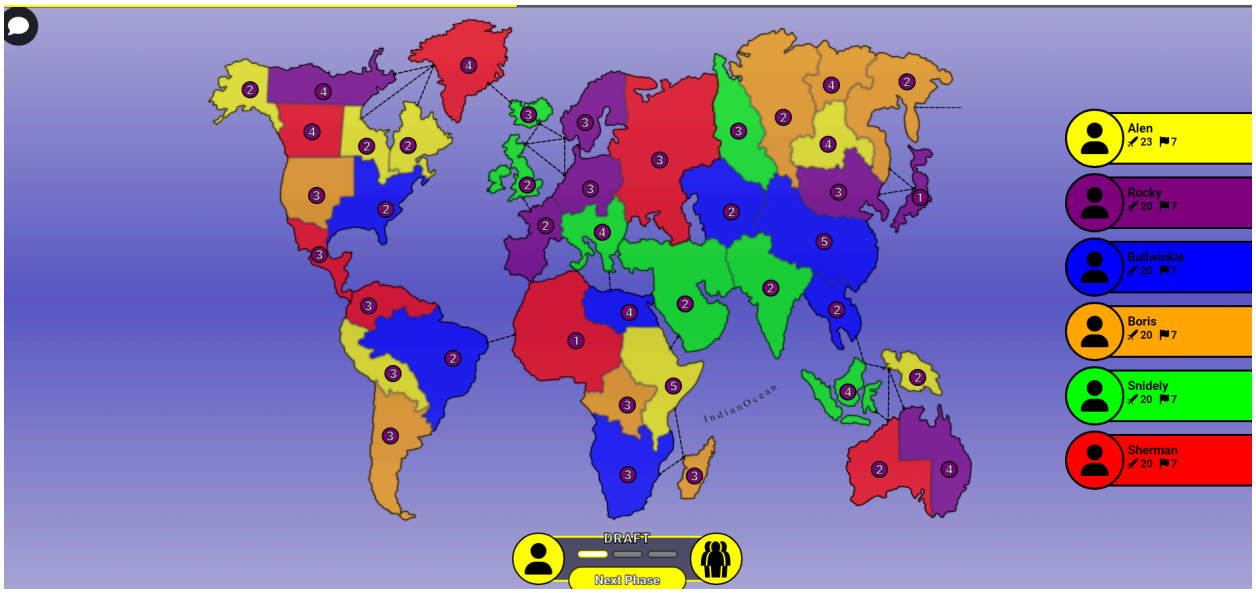
A.1 Source code: https://github.com/agracanin/capstone_project

B.1 Screenshots showing flow of the game as described in Section IV


Start A New Game
Join A Game

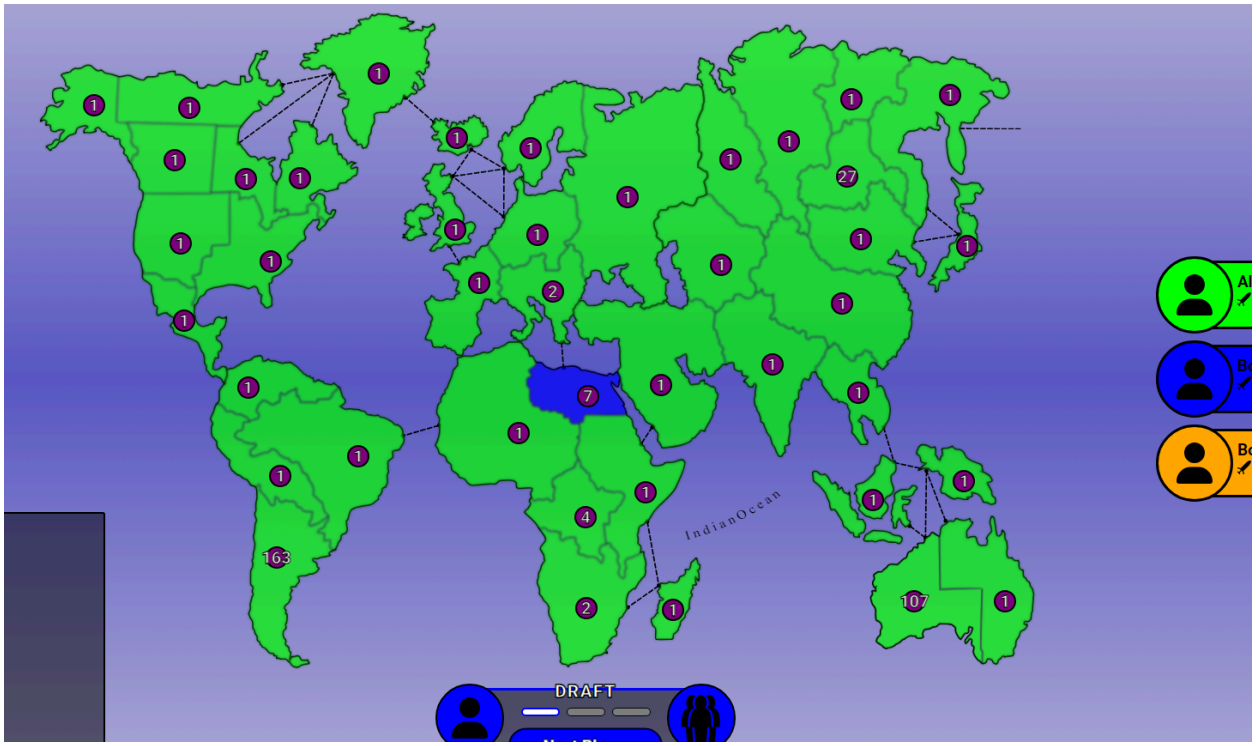

x
Game
Host
Players
Mode
Game 1
Alen
4
Classic

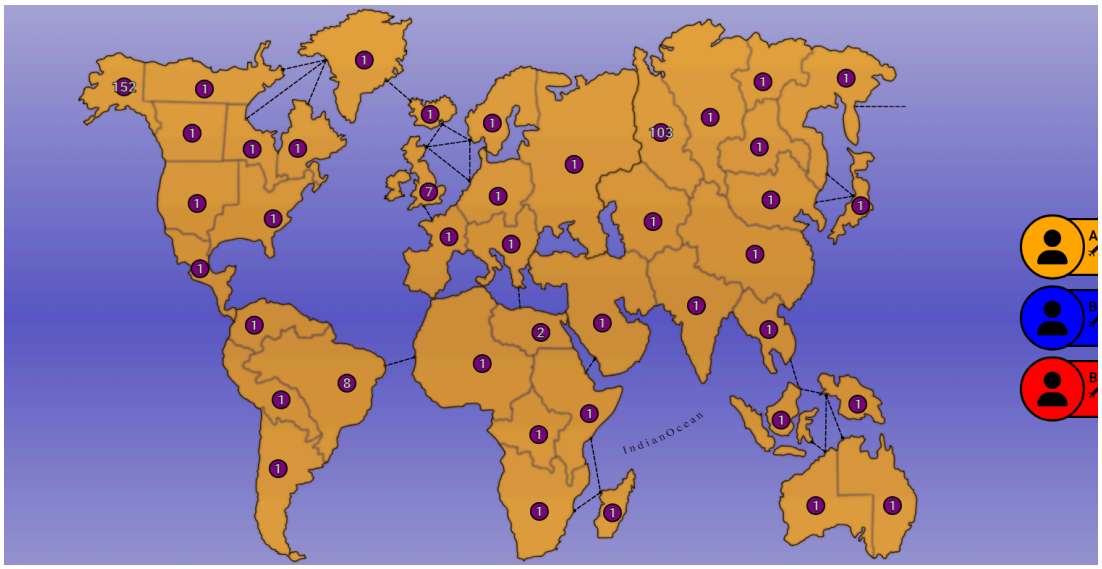

RISK ONLINE
Leave Game
Alen
Rocky
Bullwinkle
Boris
Snidely
Sherman
System: Rocky has joined the game.
System: Bullwinkle has joined the game.
System: Boris has joined the game.
System: Snidely has joined the game.
System: Sherman has joined the game.
Boris: CAN YOU START ALREADY
Rocky: stop yelling
Type your message...
Start Game


Alen
Rocky
Bullwinkle
Boris
Snidely
Sherman
DRAFT
Next Phase


Attacking Troops
4
5
1
Alen
Boris
Snidely
Sherman
ATTACK
Next Phase


3
2
3
1


IndianOcean
DRAFT


IndianOcean