PEER TO PEER BACKGAMMON

ARINA SAMOJLENKO, 202402637
MÁTÉ KORNIDESZ, 202100836
MIKKEL KATHOLM, 202107199

ALFA

PROJECT REPORT

December 2024

Advisor: Niels Olof Bouvin

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

PEER TO PEER BACKGAMMON

STUDENT ARINA, MÁTÉ & MIKKEL

Project Report

Department of Computer Science
Faculty of Natural Sciences
Aarhus University

December 2024

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

1 INTRODUCTION

2 ARCHITECTURE & IMPLEMENTATION

2.1 GENERAL ARCHITECTURE

The project consists of three main components: test

- **Client:** A React application that serves as the user interface and handles the game logic for the backgammon game. It uses Material-UI for theming and styling. The client allows users to register, log in, and play games. Key components include:
  - **Home Page** ('HomePage.js'): The landing page where users can choose to log in or register.
  - **Authentication Pages** ('SignIn.js', 'SignUp.js'): Forms that handle user authentication.
  - **Game Board** ('GameBoard.js'): Displays the backgammon board and manages game interactions.
- **Server:** An ASP.NET application written in C#. The server's primary role is to match players and facilitate the initial connection between them. Once two players are connected, the server disconnects, and the players establish a peer-to-peer connection.
- **Peer-to-Peer Connection:** After matchmaking, players communicate directly using PeerJS and WebRTC. This peer-to-peer connection handles real-time game data exchange without further server involvement.

This architecture minimizes server load by offloading game interactions to peer-to-peer connections, ensuring a responsive gaming experience. Security measures, such as cryptographic protocols, are implemented to prevent cheating and ensure fair play.

2.2 GAME LOGIC

Given that we are implementing standard backgammon, the game logic is relatively simple, there is however some basic rules and layout that is important to know and understand. Firstly the game is played on a board with 24 points, 12 on each side, the goal of the game is to move all of your pieces to the home board and then bear them off. The movement of a player piece is determined the roll of two dies, where a player can move one piece the sum of the two dies or two pieces the

value of each die, assuming the move is legal. A player can not move the piece if there is strictly more than one of the opponents pieces on the destination. If there is exactly one of the opponents pieces on the destination, the piece is hit and moved to the bar. The player must then move the piece from the bar to the opponents home board before moving any other pieces. When a player has moved all of their pieces to the home board, they can begin moving them off the board. The winner of the game is the player that first moves all of their pieces off the board.

As illustrated in figure 2.1, and briefly described above, the board is divided into four quadrants, where whites home board is the top right quadrant (numbers 19-24) and backs home board is the bottom right quadrant (numbers 1-6). This subsequently means that white and black move in opposite directions. White moves in a clockwise direction increasing in number, while black moves in a counter-clockwise direction decreasing in number. This is important to keep in mind when implementing the movement of the pieces and checking for legal moves.

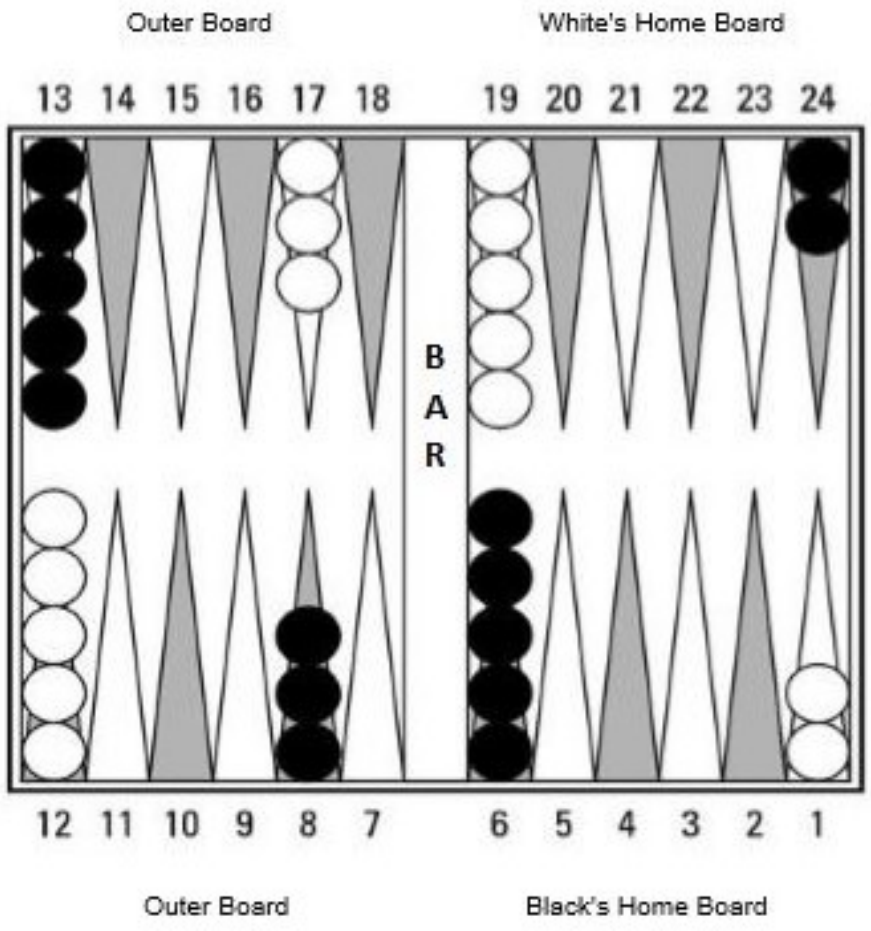

Figure 2.1: Backgammon board setup

2.3 NETWORKING

There are two types of networking used in this project, the first is the server that the players has to connect to in order to get matched with another player, the second is the peer-to-peer connection that is established between the two players once they are matched. As briefly mentioned in the general architecture section, the server is active at all times and is used to connect the players to each other, once two players are connected to each other the server disconnects from the server and from that point on the players are connected only to each other. Given that we want to use the server as little as possible, the statistics of the players are stored in the browser, as we will cover in section 2.5.1 this poses a minor security risk. There is also the possibility of a player clearing their browser cache and thus losing their statistics, this problem could be solved in many ways, one way could be to store the statistics in a central database, or as we would prefer, using a peer-to-peer database, it would even be possible to use a blockchain to store the statistics, it might be a bit overkill, but it is a possibility. As mentioned earlier we simply store the statistics in the browser, and thus none of the mentioned solutions are implemented, we are however aware of the problem and have thought about possible solutions.

2.4 USER INTERFACE

The user interface is primarily implemented using React. The first thing a player will see is the games home page where they can register or login. Once a player has registered their information is stored in a database, and they can log in. Once a player is logged in they are put in a lobby where they can either get matched with another player or go to a statistics page. Note that even though the statistics page is mentioned, and the pages does exist the functionality is not implemented.

When they are matched by the server with another player in their player group the game begins. Once the players are matched they are connected to each other using PeerJS and the game begins. The game is played on a board that is displayed to both players, the board is a standard backgammon board with 24 points, 12 on each side.

The whole user interface is designed to be intuitive and user-friendly. Everything the user will see is relatively simple in the design, partly because the game itself is rather simple, but also because we wanted to keep the design simple and clean.

The game board itself is implemented using the GameBoard component which uses various components to display the pieces and handle user interactions. The styling for the user interface is managed using

CSS and images, the images are used to display the pieces and the board itself.

2.5 IMPLEMENTATION

Our project uses multiple programming languages and technologies to implement that game and server connection to connect the players. The server part is written primary in C# and uses ASP.NET as the framework for setting up the server. The game itself is written in JavaScript and primarily uses the React library for the user interface.

2.5.1 Security

The major security concern in this project is ensuring that it is with high probability not possible for one of the players to cheat by convincing the opponent that the die he threw was different from what it actually was. In many instances like this there is a third party server that ensures that the parties playing the game is not cheating, but in our case where the game is distributed using a peer-to-peer network, we can not use a third party server as then it would not really be a peer-to-peer network. To solve this problem we have designed a cryptographic protocol that with high probability ensures that the players can not cheat. The protocol is as follows:

For simplicity, we note that the die has 6 sides and that the players are called A and B

1. A picks a random number $r_A \in \{1,2,3,4,5,6\}$ and a nonce $n_A \in \{0,1,\ldots 2^{128}-1\}$
2. A uses a cryptographic hash function H to calculate $h_A = H(r_A||n_A)$ where $||$ denotes concatenation
3. B picks a random number $r_B \in \{1,2,3,4,5,6\}$ and a nonce $n_B \in \{0,1,\ldots 2^{128}-1\}$
4. B uses a cryptographic hash function H to calculate $h_B = H(r_B||n_B)$ where $||$ denotes concatenation
5. A sends h_A to B
6. B sends h_B to A
7. Once A has received h_B and B has received h_A they send r_A, n_A and r_B, n_B respectively to the other player.
8. The players check if $H(r_A||n_A) = h_A$ and $H(r_B||n_B) = h_B$ if this is the case a fair die has been thrown and the protocol continues. If not cheating is suspected and the game is terminated.

9. Both players compute the value of the die as $(r_A + r_B \mod 6) + 1$

The protocol is designed to mimic a fair die, each player contributes with a random number for the die, as the number is random the opponent has no way of knowing what number is picked and thus has no way of influencing the outcome of the die. The nonce is used such that the opponent can not reuse or brute force the hash function to find the number that the opponent picked.

There is a small caveat to the protocol, if a player is able to brute force the hash function, he can cheat. This is however not really a problem as we are using a cryptographic hash function, which there is no efficient way of brute forcing, furthermore the protocol uses a nonce of 128 bits and a one bit number, which means that there is 2^{129} possible combinations for each die throw, which is infeasible to brute force. If further security is needed the nonce can simply be increased to a larger number, or slow hashing can be used.

A second security problem that is worth the at least have thought about is the possibility of a player to manipulate the number of games won. It is a problem that we have not taken security measures against, the main reason is that if a player is able to manipulate the number of games won, the player is really only cheating themselves. Given that players are matched based on the number of games won, the player will be matched with players that are better than them, and thus the cheating player will properly just lose the game. Furthermore, there is currently no way to profit monetarily from this game. Now that we have covered that there is no incentive to falsely increase the number of games won, a player might be lower the number of games won to get matched with players that are worse than them, this is not a new problem as a player can simply make a new account and get matched with players on the same level as the new account. This is a problem that is inherent to all games that are based on the number of games won, and is not something that we have taken measures against.

2.6 BACKLOG

The following is a list of features that we would have liked to implement, but did not have time to implement. Some of the features are not essential to the game, but would have been nice to have. The features are listed in no particular order.

- **Statistics:** A page where the player can see their statistics, such as number of games won, number of games lost, win rate etc.
- **Distributed Database:** As mentioned in the networking section, the statistics are stored in the browser, this poses a security risk,

and it would have been nice to have a distributed database to store the statistics.

- **Double Dice:** When a player throws a double, the player can move four times the value of the die, this is not implemented in the game.
- **Crawford rule:** The Crawford rule states that if a player is one point away from winning the game, the opponent can not double the game.
- **Handicap:** If a player has a higher win rate than the opponent, the opponent can remove a number of pieces from the board before the game begins.

3 EVALUATION & CONCLUSION

A AN APPENDIX

A.1 GITLAB REPOSITORY

The code for the project can be found at the following GitLab repository. Note that at the time of writing the repository is private and thus requires access to view. It is however expected that the repository will be made public at some point in the future.

https://gitlab.au.dk/au702308/building-the-iot

A.2 YOUTUBE DEMONSTRATION VIDEO

A demonstration video of the project can be found at the following YouTube link.

LinktoYouTubevideo