

**Multilayer Grid Based Dexterity  
Training Game  
NEA: Georgiy Tnimov**

## Contents

0.1.	Abstract .....	3
0.2.	Problem Definition .....	3
0.3.	Client .....	4
0.3.1.	Client Synopsys (conclusion) .....	4
0.3.2.	Interview Notes .....	4
0.4.	Success Criteria .....	5
0.5.	Research .....	5
0.5.1.	Similar Solutions .....	5
0.5.1.1.	Tetris .....	5
0.5.1.2.	Tapp .....	6
0.5.1.3.	Other Dexterity Training Applications .....	6
0.5.2.	Multiplayer .....	6
0.5.3.	PRNG's (Pseudorandom Number Generators) .....	7
0.5.4.	Statistics(anti-cheat) .....	10
0.5.4.1.	Player timings .....	10
0.5.4.2.	Path optimality .....	10
0.6.	Prototyping .....	10
0.7.	Critical Path .....	13
0.8.	Objectives .....	13
0.8.1.	User Interface .....	13
0.8.2.	Server Side .....	14
0.9.	Documented Design .....	15
0.9.1.	Libraries Used .....	15
0.9.1.1.	Frontend Libraries .....	15
0.9.1.2.	Backend Libraries .....	16
0.9.2.	Iterative Design .....	17
0.9.3.	Algorithms .....	17
0.9.3.1.	Xoshiro256+ .....	17
0.9.3.2.	Sigmoid Function .....	18
0.9.3.3.	Manhattan Distance .....	18
0.9.3.4.	MergeSort .....	19
0.9.3.5.	Standard deviation .....	19
0.9.3.6.	Delayed Auto Shift .....	20
0.9.4.	Database Design and Queries .....	20
0.9.4.1.	User Authentication Queries .....	21
0.9.4.2.	User Registration Query .....	21
0.9.4.3.	Session Management .....	21
0.9.4.4.	Leaderboard Queries .....	22
0.9.4.5.	Game Submission .....	22
0.9.4.6.	Statistics Trigger .....	23
0.9.5.	Data Structures .....	23
0.9.5.1.	Circular Queue .....	23
0.9.5.2.	HashMap .....	23
0.9.5.3.	Option/Result Types .....	23
0.9.6.	Diagrams .....	24
0.9.7.	Frontend .....	24
0.9.8.	Backend .....	28
0.10.	Technical Solution .....	32

0.10.1. Code Contents .....	32
0.10.2. Skill table .....	34
0.10.3. Completeness of Solution .....	35
0.10.4. Code Quality .....	35
0.10.5. Source Code .....	36
0.10.5.1. Grid Component .....	36
0.10.5.2. Authentication .....	49
0.10.5.2.1. Frontend .....	49
0.10.5.2.2. Backend .....	51
0.10.5.3. Queue .....	53
0.10.5.4. Leaderboard .....	54
0.10.5.5. Server Routing .....	57
0.10.5.6. Singleplayer Game Management .....	58
0.10.5.7. Backend Error Handling .....	62
0.10.5.8. Database Models .....	63
0.10.5.9. Multiplayer game management .....	64
0.10.5.10. WASM .....	70
0.10.5.11. Settings Component .....	71
0.10.5.12. Layout and Styling .....	76
0.11. Testing .....	79
0.12. Evaluation .....	81
0.12.1. Overall Effectiveness .....	81
0.12.2. Evaluation Against Objectives .....	81
0.12.3. User Feedback .....	81
0.12.4. Response to Feedback .....	81
0.12.5. Future Improvements .....	81
1. Bibliography .....	82

## 0.1. Abstract

This project develops a multiplayer grid-based dexterity training game called DoubleTapp, designed to simultaneously test and improve the dexterity of both hands. Building on the existing single-cursor game Tapp, this implementation introduces dual-cursor gameplay requiring coordinated control using different keys for each hand. The system features both singleplayer and multiplayer modes with competitive elements, leaderboards, and server-side anti-cheat mechanisms.

The technical implementation uses Rust for the backend with the Axum framework for websocket connections and PostgreSQL for data persistence. The frontend is built with SvelteKit and Tailwind CSS, featuring customizable controls including Delayed Auto Shift (DAS) functionality. A custom implementation of the Xoshiro256+ PRNG algorithm ensures fairness across game instances.

## 0.2. Problem Definition

I plan to develop a game, which tests the dexterity of both hands, simultaneously. I believe its important that people can maintain their dexterity of both hands, and this game will help them do that. I also believe the game will be fun, and will be a good way to pass time. adding a competitive and multiplayer aspect to the game will also help with this.

I plan to develop this game using Rust and Svelte, as well as a websocket server, which will be used to communicate between the client and server.

## 0.3. Client

### 0.3.1. Client Synopsys (conclusion)

The Client is Alexander Tahiri, a software developer at Studio Squared and the developer of Tapp, a game based on a 4x4 grid, which consists of 12 inactive tiles, and 4 active tiles. Players use the mouse cursor to click on an active tile, which then deactivates that tile and activates a new, currently non-active tile. The objective of Tapp is to achieve as high a score as possible, without making any mistakes. The Client requires a derivative of this game, which tests simultaneous dexterity of both hands, additionally The Client wants to incorporate a competitive aspect to the game, which consists of a leaderboard section, allowing players to see their position within the rankings and a Tetris-99-esque game mechanic, where players compete to either achieve the highest score, or last the longest in a mass multiplayer format. The Client has specifically asked for the Catppuccin colour scheme to be used, The Client has sufficient computing power to host both the client, server and database, which will be provided free of charge.

### 0.3.2. Interview Notes

(all notes are paraphrased)

Q: What features are most important to you for DoubleTapp?

A: My main requirement is that the new game tests both hands simultaneously, and has replayability. Features such as users and leaderboards, along with a competitive aspect would be awesome.

Q: How many users do you expect to scale to?

A: I am estimating up to 50 concurrent users, and aim for small latencies.

Q: Any specific UI/GUI choices, and what platform should DoubleTapp support?

A: DoubleTapp should be a website, like the original Tapp, and it should use the Catppuccin color scheme.

Q: Any specific technologies you would like implemented?

A: I am a fan of Svelte, and would like to use Rust as the backend due to its fast speeds and growing technology base. Tapp doesn't have a database but SQL would be acceptable.

Q: DoubleTapp might have a cheating problem, would you like an anticheat?

A: An anticheat would be desirable. Due to Svelte being unobfuscated, a server-side anticheat might be best.

Q: What are your thoughts on monetization for DoubleTapp?

A: I'd prefer to keep it free to play. The focus should be on building a community rather than generating revenue at this stage.

Q: How important is cross-device compatibility?

A: The primary focus should be desktop browsers, but having it work reasonably well on tablets would be a nice bonus. I don't expect mobile phone support due to the dual-input nature.

Q: Any accessibility considerations you'd like to see implemented?

A: Customizable keybindings would be essential since this is a dexterity game. Also, ensuring the color scheme has sufficient contrast for visibility would be good.

## 0.4. Success Criteria

- game is completely functional
- server can handle 50 concurrent users
- average user rating is 4/5 or higher
- aesthetically pleasing UI
- useful UX
- easy to understand and customize settings

## 0.5. Research

### 0.5.1. Similar Solutions

There are a few similar products on the market that test dexterity in various ways. Understanding these existing solutions helps position DoubleTap in the competitive landscape and justify its development.

#### 0.5.1.1. Tetris

Tetris is one of the most recognized dexterity-based puzzle games worldwide. While it effectively tests hand-eye coordination and spatial reasoning, it differs from DoubleTap in several key ways:

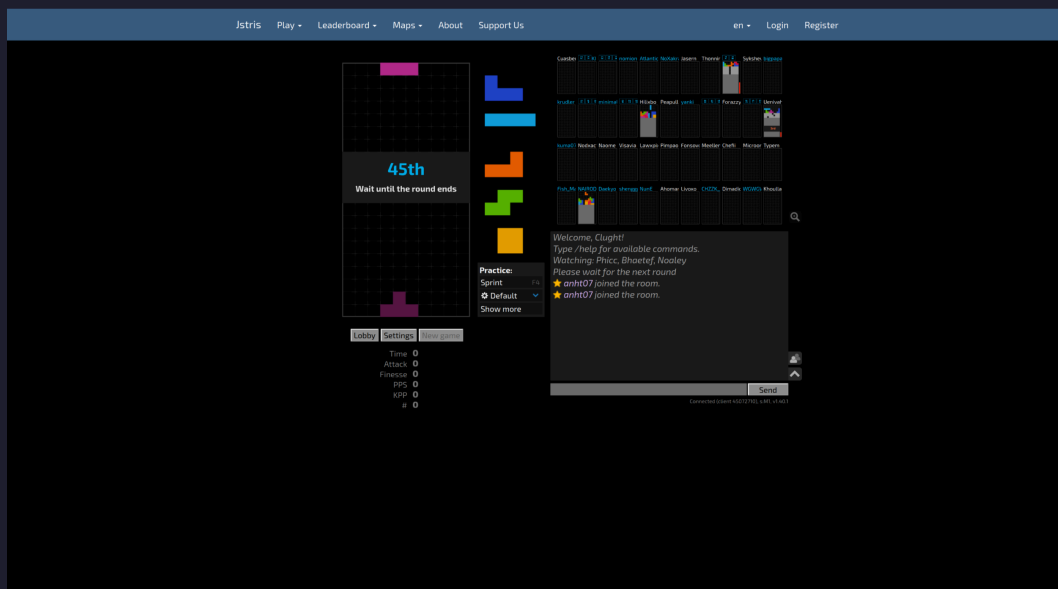


Figure 1: Tetris UI

- Tetris focuses primarily on single-hand dexterity, with players typically using one hand for directional controls and the other for occasional rotation/drop buttons
- It has a significant learning curve with complex strategies around piece placement and line clearing, i.e T-spins, Wall Kicks
- Players focus more on strategic planning of where to place pieces rather than pure dexterity training
- The modern competitive versions of Tetris (like Tetris 99) do incorporate multiplayer aspects, but interaction between players is indirect through “garbage lines”

Tetris has multiple useful features which I will be taking inspiration from, particularly Delayed Auto Shift (DAS)[1], which allows for precise control of pieces,

this allows for people to have more accurate control over their piece placement and allows for timing optimization

0.5.1.2. Tapp

Tapp, developed by Alexander Tahiri at Studio Squared, is the direct predecessor to DoubleTapp and shares the most similarities:

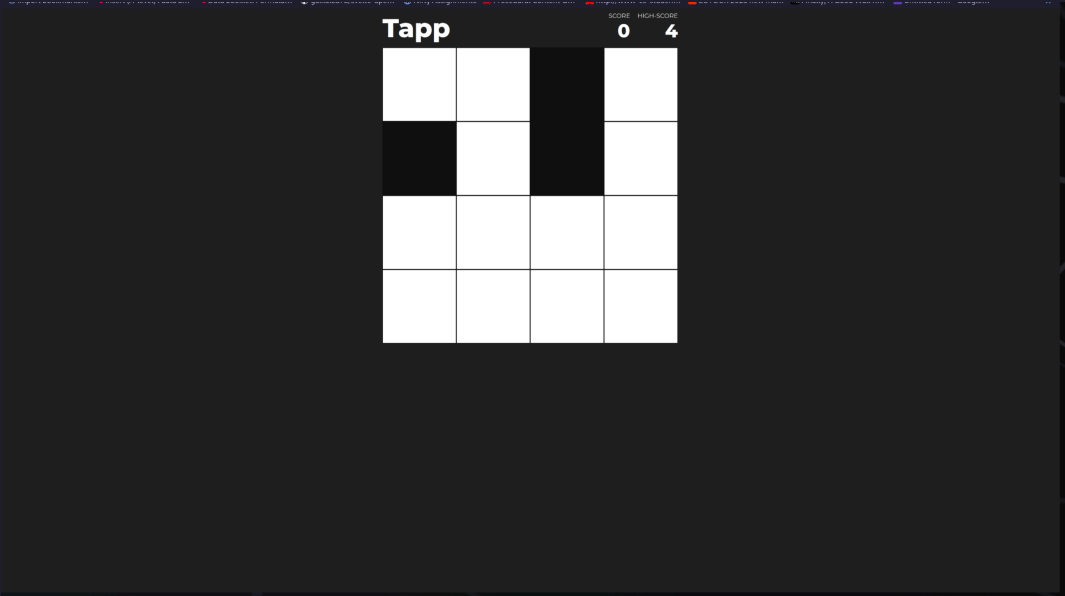


Figure 2: Tapp UI

- Uses a grid-based interface (4x4) with active and inactive tiles
- Tests dexterity through rapid target acquisition
- Focuses on score maximization without mistakes
- Simple, accessible gameplay with minimal learning curve

However, Tapp is limited to single-hand dexterity training, using only mouse input. It lacks the simultaneous dual-hand coordination that DoubleTapp aims to develop. Additionally, Tapp has no built-in multiplayer functionality or competitive leaderboard system.

0.5.1.3. Other Dexterity Training Applications

Various other applications exist for dexterity training, including:

- Typing games that test two-handed coordination but in a highly structured, predictable pattern (monkeytype, nitrotype)
- Rhythm games (like Dance Dance Revolution or osu!) that test reaction time and coordination but typically focus on timing rather than spatial navigation
- Aim trainers (for FPS games) that focus exclusively on mouse precision, although sometimes incorporate simultaneous dexterity, i.e counterstrafing, bhopping, edgebugging

0.5.2. Multiplayer

for implementing multiplayer, there are multiple solutions that work, i.e unidirectional HTTP requests, custom UDP handling, and websockets

Method	Pros	Cons
--------	------	------

HTTP [2]	<ul style="list-style-type: none"> <li>• Simple implementation</li> <li>• Reasonably performant</li> <li>• Easily Debuggable</li> <li>• widely supported</li> </ul>	<ul style="list-style-type: none"> <li>• Slow with many simultaneous users</li> <li>• Requires entire connection sequence for each request</li> <li>• relatively high latency</li> <li>• not designed for bidirectional communication</li> </ul>
UDP [3]	<ul style="list-style-type: none"> <li>• Very performant</li> <li>• allows for low level optimisations</li> <li>• minimal overhead</li> </ul>	<ul style="list-style-type: none"> <li>• susceptible to packet loss, and is not guaranteed to have data parity (important for doubletapp)</li> <li>• complex to implement, and difficult to interconnect with existing libraries without significant performance declines</li> <li>• often blocked by firewalls</li> <li>• no ordering guarantees</li> </ul>
Websockets [4]	<ul style="list-style-type: none"> <li>• allows for fast and safe data transmission</li> <li>• relatively complex to implement, as need to handle assignment of websockets to individual games</li> <li>• compatible with existing web server libraries</li> <li>• fully duplex, no need to reestablish connection sequence each request</li> </ul>	<ul style="list-style-type: none"> <li>• websockets don't recover when connections are terminated</li> <li>• some networks block the websocket protocol, limiting accessibility</li> <li>• high memory usage per connection compared to UDP/HTTP</li> </ul>

I have decided to use websockets, as they are a reasonable balance of complexity, performance, and ease of implementation, while still providing a high degree of reliability and safety.

### 0.5.3. PRNG's (Pseudorandom Number Generators)

after considering many PRNG's (pseudorandomnumber generators), for example ARC4 , seedrandom, ChaCha20, and discounting them due to performance issues / hardware dependent randomization, I decided on using the Xoshiro/Xoroshiro family of algorithms, which are based on the Linear Congruential Generators, which are a (now-obsolete) family of PRNG's, which use a linear multiplication combined with modulus operations, to create quite large non-repeating sequences, although quite slow and needing very large state. xoshiro generators use a much smaller state (between 128-512) bits, while still maintaining a large periodicity,

PRNG Algorithm	Pros	Cons
----------------	------	------

ARC4 (Alleged RC4)	<ul style="list-style-type: none"> <li>• Simple implementation</li> <li>• Fast for small applications</li> <li>• Variable key size</li> </ul>	<ul style="list-style-type: none"> <li>• Cryptographically broken</li> <li>• Biased output in early stream</li> <li>• Vulnerable to related-key attacks</li> </ul>
Seedrandom.js	<ul style="list-style-type: none"> <li>• Browser-friendly</li> <li>• Multiple algorithm options</li> <li>• Good for web applications</li> </ul>	<ul style="list-style-type: none"> <li>• JavaScript performance limitations</li> <li>• Depends on implementation quality</li> <li>• Not cryptographically secure by default</li> </ul>
ChaCha20	<ul style="list-style-type: none"> <li>• Cryptographically secure</li> <li>• Excellent statistical properties</li> <li>• Fast in software (no large tables)</li> <li>• Parallelizable</li> </ul>	<ul style="list-style-type: none"> <li>• Complex implementation</li> <li>• Overkill for non-security applications</li> <li>• Higher computational cost</li> </ul>
Xorshift	<ul style="list-style-type: none"> <li>• Extremely fast</li> <li>• Simple implementation</li> <li>• Good statistical quality</li> </ul>	<ul style="list-style-type: none"> <li>• Not cryptographically secure</li> <li>• Simpler variants have known weaknesses</li> <li>• Some states can lead to poor quality</li> </ul>
Linear Congruential Generator (LCG)	<ul style="list-style-type: none"> <li>• Simplest implementation</li> <li>• Very fast</li> <li>• Small state</li> </ul>	<ul style="list-style-type: none"> <li>• Poor statistical quality</li> <li>• Short period for 32-bit implementations</li> <li>• Predictable patterns</li> </ul>
Mersenne Twister	<ul style="list-style-type: none"> <li>• Very long period</li> <li>• Good statistical properties</li> <li>• Industry standard in many fields</li> </ul>	<ul style="list-style-type: none"> <li>• Large state (2.5KB)</li> <li>• Not cryptographically secure</li> <li>• Slow initialization</li> </ul>
Xoshiro256+/++	<ul style="list-style-type: none"> <li>• Excellent speed</li> <li>• Great statistical properties</li> <li>• Small state (256 bits)</li> <li>• Fast initialization</li> </ul>	<ul style="list-style-type: none"> <li>• Not cryptographically secure</li> <li>• Newer algorithm (less scrutiny)</li> <li>• Some variants have issues with specific bits</li> </ul>



PCG (Permuted Congruential Generator)	<ul style="list-style-type: none"> <li>• Excellent statistical properties</li> <li>• Small state</li> <li>• Good performance</li> <li>• Multiple variants available</li> </ul>	<ul style="list-style-type: none"> <li>• More complex than basic PRNGs</li> <li>• Not cryptographically secure</li> <li>• Relatively new</li> </ul>
---------------------------------------	--	---

PRNG Algorithm	Estimated Time	Cycle Length	State Size	Performance
ARC4	Medium	$10^{100}$	256 bits	Moderate
seedrandom.js	Medium	(multiple selectable algorithms)	Varies by algorithm	Moderate (JS limited)
ChaCha20	High	$2^{256}$	384 bits	High for crypto
Xorshift	Very Low	$2^{128} - 1$	128-256 bits	Very High
Linear Congruential Generator (LCG)	Extremely Low	Up to $2^{32}$	32-64 bits	Extremely High
Mersenne Twister	Medium	$2^{19937} - 1$	2.5 KB (19937 bits)	Moderate
Xoshiro256+/+	Very Low	$2^{256} - 1$	256 bits	Very High
PCG (Permuted Congruential Generator)	Low	$2^{128}$ or more	64-128 bits	High

after testing, xoshiro256+ has provided the best results, in terms of speed and simplicity of implementation, while still providing a high degree of randomness, and a large cycle length, which is important for a game such as DoubleTapp, where we want to ensure that the game is fair and that the same seed will not be repeated for a long time. additionally the math behind Xoshiro is layered in complexity, and really interesting, which has led me to want to implement it

#### 0.5.4. Statistics(anti-cheat)

for the anticheat,I will be comparing the consistency of player movement timings, and the optimality of their paths, to approximately determine if they are using any forms of cheating, be it a bot, or a human using external software.

##### 0.5.4.1. Player timings

for player timings, I will be using the standard deviation of the player's move timings, and comparing it to a sampled standard deviation based on my own move timings, a high standard deviation indicates that the player is more human, as different grid positions require different amounts of thought to move optimally

##### 0.5.4.2. Path optimality

for calculating optimal paths, there are a few different algorithms that can be used, each having different time and space complexities, it is important that the algorithm calculates the optimal path, not a close approximation, as this will be used to detect potential cheaters. performance is inherently critical for this part, as it will be run on every "submission" of a move, and will need to be done concurrently.

Algorithm	Time Complexity	Space Complexity
A-Star	$O(b^d)$	$O(b^d)$
Dijkstra's	$O(V + E)$	$O(V)$
Manhattan Distance	$O(1)$	$O(1)$

overall, manhattan distance is the best option for this project, as at max the grid would be 6x6, in which using A-star would be overkill, and manhattan distance is the fastest, while djikstra's is the slowest, and would be too slow for the game.

#### 0.6. Prototyping

A rudimentary prototype has been made, which tested out multiple different input methods for simultaneous inputs, which has finalized in a "cursor"-based system, where you have two cursors controlled by Wasd-like movement, with each set of controls representing their respective cursor, additionally it has been decided that both cursors need to be on individual Tiles, to prevent copying movements on each hand. this prototype also implemented server-side move verification, making it more difficult to cheat. Finally, the UI design of the prototype will be used in later iterations of the project. the prototype has no game verification, but contains the core gameplay mechanics, and the UI design.

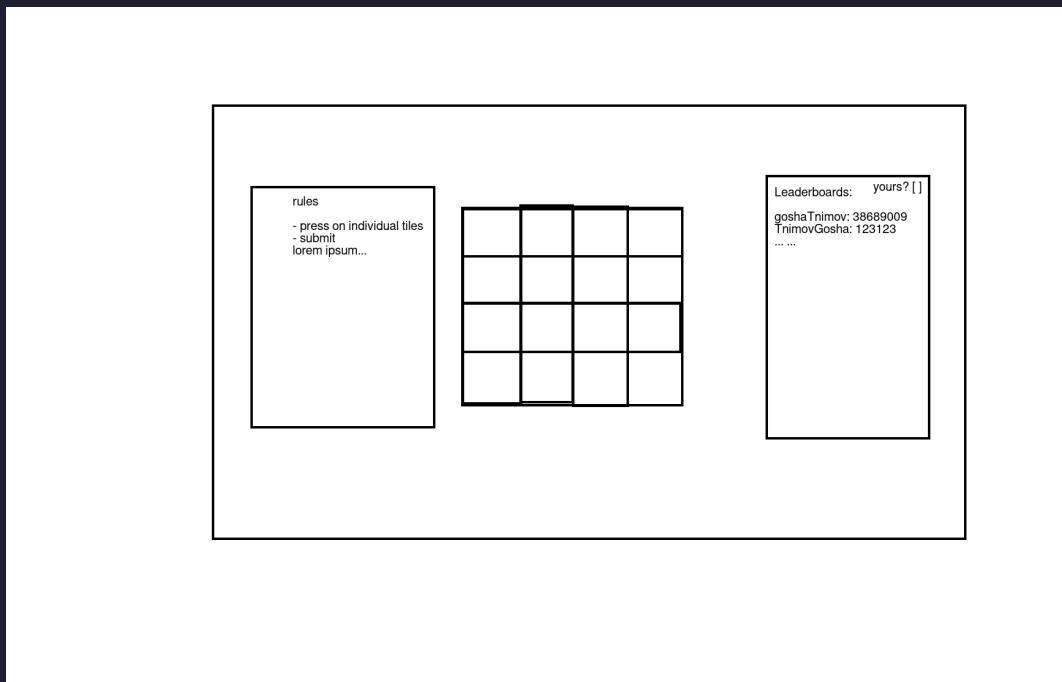


Figure 3: Initial Doubletapp WireFrame UI

this was the initial UI design sketch, which shows the general layout of the game

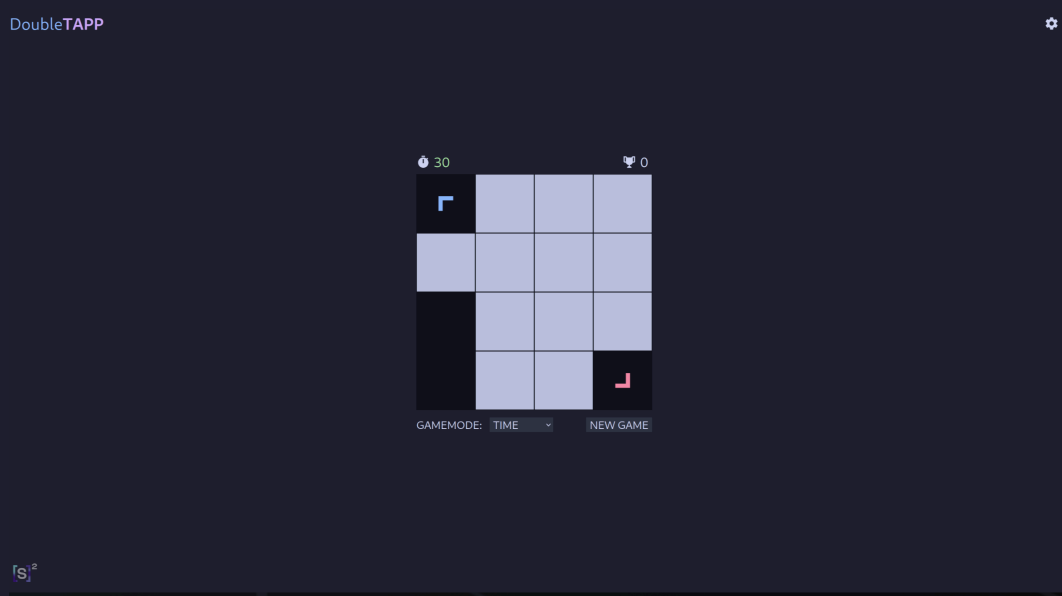


Figure 4: Initial Doubletapp WireFrame UI

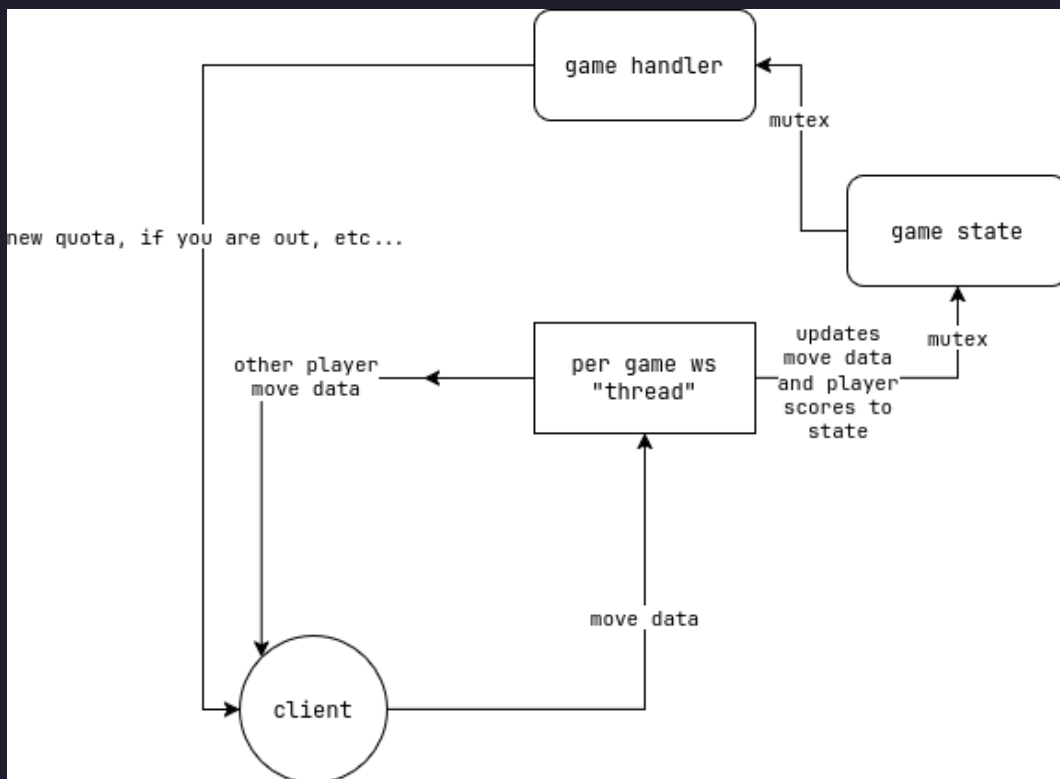


Figure 5: Game Handler Prototype Flowchart - Early design of the game processing pipeline

## 0.7. Critical Path



Figure 6: Intended Critical Path

## 0.8. Objectives

### 0.8.1. User Interface

- 1 user can interact with the grid
  - 1.1 user can move both cursors using keyboard on the grid
  - 1.2 user can "submit" moves using a keybind
  - 1.3 user can reset game (in single player) via a keybind
- 2 user can change gamemode (singleplayer,multiplayer) on the main page
  - 2.1 user can change grid size (4x4,5x5,6x6) in singleplayer
  - 2.2 in singleplayer, user can change time limit (30,45,60)
- 3 user can access settings
  - 3.1 user can modify keybinds for each action in the game
  - 3.2 user can change DAS
  - 3.3 user can change ARR
  - 3.4 user can log out of account
  - 3.5 user can reset all keybinds to a sane default
- 4 user can play the game
  - 4.1 on game start, user sees cursors are positioned on opposing sides of the board
  - 4.2 on game start, user sees the starting active tiles
  - 4.3 user can view current game score
  - 4.4 in singleplayer, user sees time remaining

- 4.5 in multiplayer, user can see time remaining for current quota, players remaining and current score
- 4.6 user is notified of their position in the multiplayer game
- 4.7 user can "submit" their move
  - 4.7.1 user can interactively see if the move was valid via a colour interaction which flashes green or red depending on if the move was valid, a valid move is when the two cursors are on two active grid tiles within the grid boundary and they are distinct active tiles
  - 4.7.2 on successful submit, user sees two new tiles become active, which were previously inactive and are not on current cursor location
- 4.8 cursors are rendered via two different colours, with the two cursors being visually distinct but symmetrically consistent
- 5 user can see statistics post singleplayer game end
  - 5.1 user views their score
  - 5.2 user views if their score was validated by the server
  - 5.3 user views their leaderboard position
  - 5.4 user can copy their game statistics to the clipboard for sharing
  - 5.5 if user is logged in and not marked as a cheater, user can view their game in the statistics page
  - 5.6 user has the option to start a new game from the results menu
- 6 user can view leaderboard
  - 6.1 user can view leaderboards, in a paginated format
- 7 user can play the multiplayer gamemode
  - 7.1 user can see the other players movements on other grids in the game
  - 7.2 user can see their remaining score quota for each 5 second interval period
  - 7.3 after a user has been eliminated by not reaching the quota, the user can view their position in the game
- 8 user can log in to the application
  - 8.1 user can login or signup depending on their requirements
  - 8.2 user is shown error codes depending on if account already exists or their login details are incorrect

## 0.8.2. Server Side

- 1 User CRUD
  - 1.1 simple user authentication
    - 1.1.1 simple verification of authenticity, i.e password hashing & username uniqueness check
- 2 Database Schema
  - 2.1 contains user table
  - 2.2 contains game table, which stores all real authenticated games (not including moves)
  - 2.3 contains linked user statistics table
- 3 Game Verification
  - 3.1 server verifies all moves are valid
  - 3.2 server verifies that move positioning is within human bounds, i.e ratio of "optimal moves" and timing distribution
  - 3.3 server verifies that game was submitted within the time limit (with a grace period)
- 4 Multiplayer implementation
  - 4.1 server can communicate actions bidirectionally with client

- 4.2 each move is verified by the server
- 4.3 low latency communication between server and client
- 4.4 client can distinguish between types of messages recieved

## 0.9. Documented Design

### 0.9.1. Libraries Used

#### 0.9.1.1. Frontend Libraries

Name	Version	Reason	Link
Svelte	4.2.7	Reactive UI framework with minimal boilerplate, used for the frontend to provide a performant, easily maintainable UI/UX	<a href="https://svelte.dev">svelte.dev</a>
SvelteKit	2.0.0+	Full-stack framework built on Svelte, allowing for simplification of operations between the frontend and the backend	<a href="https://kit.svelte.dev">kit.svelte.dev</a>
Tailwind CSS	3.4.4	css library, which allows you to define your css classes embedded in the html, allowing for a more readable and quickly iterable codebase	<a href="https://tailwindcss.com">tailwindcss.com</a>
Tailwind Catppuccin	0.1.6	Client-requested color scheme	<a href="#">GitHub</a>
Svelte Material Icons	3.0.5	Icon library for Svelte, MIT licensed	<a href="#">npm</a>
UUID	11.0.4	frontend library for generating UUID's, used for game management	<a href="#">npm</a>
Xoshiro WASM	Local	Custom WASM implementation of Xoshiro256+	in code
TypeScript	5.0.0+	Typed JavaScript for better development	<a href="https://typescriptlang.org">typescriptlang.org</a>
Vite	5.0.3	Modern frontend build tool, used in frontend to allow for fast development and optimized production builds	<a href="https://vitejs.dev">vitejs.dev</a>

Vite Plugin WASM	3.4.1	Vite plugin for WebAssembly integration	<a href="#">npm</a>
------------------	-------	---	---------------------

#### 0.9.1.2. Backend Libraries

Name	Version	Reason	Link
Axum	0.7.5	Modern Rust web framework with WebSocket support, one of the fastest web frameworks currently available, asynchronous and type-safe	<a href="#">GitHub</a>
Axum-Extra	0.9.4	Extension crate for Axum with additional features like cookie handling and typed headers	<a href="#">GitHub</a>
Tokio	1.39.2	Asynchronous runtime for Rust, required by axum and used for thread handling in websockets	<a href="#">tokio.rs</a>
SQLx	0.8.0	Async SQL toolkit with compile-time checked queries, used for database operations, inherently supports pooling and multithreading.	<a href="#">GitHub</a>
Serde	1.0.205	Serialization framework for structured data, allows for parsing JSON and other data formats into Rust objects, speeding up development time and reducing the amount of code needed to be written	<a href="#">serde.rs</a>
Serde_json	1.0.128	JSON implementation for Serde, used for parsing and generating JSON data in WebSocket communication	<a href="#">GitHub</a>
Bcrypt	0.17.0	Password hashing library, used before storing passwords in the database, salted and performant, although slightly outdated	<a href="#">crates.io</a>
Tower-HTTP	0.5.2	HTTP middleware stack, baseline from axum, used for low level websocket handling	<a href="#">GitHub</a>
UUID	1.7.0	Library for generating UUIDs, used for game management	<a href="#">crates.io</a>
ULID	1.1.3	Sortable identifier generation, used for game management	<a href="#">crates.io</a>



Validator	0.20.0	Data validation library, used for validating user input	<a href="https://crates.io">crates.io</a>
Chrono	0.4.37	Date and time library with timezone support, used for handling timestamps and durations to verify games	<a href="https://crates.io">crates.io</a>
SCC	2.1.11	Concurrent collections for server applications, performant asynchronous hashmaps	<a href="https://crates.io">crates.io</a>
Silly-RNG	0.1.0	Custom RNG implementation, used for the game, based on xoshiro-wasm	Local package
Cookie	0.18.1	HTTP cookie parsing and cookie jar management, used for session handling	<a href="https://crates.io">crates.io</a>
Dotenvy	0.15.7	Loads environment variables from .env files, used for configuration management	<a href="https://crates.io">crates.io</a>
Futures	0.3.31	Async programming primitives, used for handling asynchronous websocket operations	<a href="https://crates.io">crates.io</a>
Rand	0.8.5	Random number generation utilities, used for game seeding	<a href="https://crates.io">crates.io</a>
Thiserror	2.0.11	Error handling library that simplifies custom error types, used for robust error management	<a href="https://crates.io">crates.io</a>
Tracing-subscriber	0.3.18	Utilities for implementing and composing tracing subscribers, used for logging and diagnostics	<a href="https://crates.io">crates.io</a>

## 0.9.2. Iterative Design

## 0.9.3. Algorithms

### 0.9.3.1. Xoshiro256+

xoshiro256+ is my chosen RNG, as it is performant and has a relatively low state size, allowing for many concurrent games to be played on a single machine, it is also very simple to implement, and has a relatively high cycle length, allowing for a more consistent game experience, it is also very fast, and has a low memory footprint, making it a perfect fit for the game. xoshiro256+ has a time complexity of  $O(1)$ , and a space complexity of  $O(1)$ , as it only requires a single pass through the seed array, and a single pass through the result array, which is constant time, and constant space, as the size of the seed and result arrays are constant.

```

// output is generated before the "next" cycle
let result = self.seed[0].wrapping_add(self.seed[3]);
// shifting prevents guessing from linearity
let t = self.seed[1] << 17;
// these 4 xor operations simulate a matrix transformation
self.seed[2] ^= self.seed[0];
self.seed[3] ^= self.seed[1];
self.seed[1] ^= self.seed[2];
self.seed[0] ^= self.seed[3];
// last xor is just a xor
self.seed[2] ^= t;
// the rotation ensures that all bits in the seed eventually interact, allowing
for much higher periodicity (cycles before you get an identical number, which in the case
of xoshiro256+ is  $2^{256} - 1$ )
self.seed[3] = Xoshiro256plus::rol64(self.seed[3], 45);
// gets the first 53 bits of the result, as only the first 53 bits are guaranteed
to be unpredictable for xoshiro256+, for the other variations i.e ++,*,** they are
optimized for all the bits to be randomized, but as xoshiro256+ is optimized for floating
points, which we require
(result >> 11) as f64 * (1.0 / (1u64 << 53) as f64)

```

### 0.9.3.2. Sigmoid Function

the sigmoid function is a function, that maps any real input onto a S shaped curve, which is bound between values, in my case i am bounding the output of the Xoshiro256+ float to be between 0..11, which allows me to easily use it to generate the "next" state of the game, allowing for a more natural distribution of numbers, as well as a more consistent distribution of numbers, which allows for a more consistent game experience.

```

// simple function, but incredibly useful
function sigmoid(x):
    return 1.0 / (1.0 + exp(-x))

```

### 0.9.3.3. Manhattan Distance

the manhattan distance is a distance metric, which is the sum of the absolute differences of their Cartesian coordinates, in my case i am using it to calculate the distance between the cursors, which allows for a more accurate calculation of the distance between the cursors, which allows for a more accurate game experience. the time complexity of the manhattan distance is  $O(1)$ , as it only requires a single pass through the coordinates, and a single pass through the result, which is constant time, and constant space, as the size of the coordinates and result are constant.

```

fn manhattan_distance(x1: f64, y1: f64, x2: f64, y2: f64) -> f64 {
    (x1 - x2).abs() + (y1 - y2).abs()
}

```

#### 0.9.3.4. MergeSort

mergesort is a sorting algorithm, which works by the divide and conquer principle, where it breaks down the array into smaller and smaller arrays, till it gets to arrays of length 2, which it then subsequently sorts from the ground up, returning a sorted array in  $O(n \log(n))$  time complexity &  $O(n)$  space complexity

```
function merge_sort(array):
    if length of array <= 1:
        return array

    mid = length of array / 2
    left = merge_sort(subarray from start to mid)
    right = merge_sort(subarray from mid to end)

    return merge(left, right)

function merge(left, right):
    result = empty list
    left_index = 0
    right_index = 0

    while left_index < length of left and right_index < length of right:
        if left[left_index] <= right[right_index]:
            append left[left_index] to result
            left_index = left_index + 1
        else:
            append right[right_index] to result
            right_index = right_index + 1

    append remaining elements from left starting at left_index to result
    append remaining elements from right starting at right_index to result

    return result
```

#### 0.9.3.5. Standard deviation

the algorithm for standard deviation is as follows:

$$\sigma = \sqrt{\frac{(\sum(x) - \mu)^2}{N}}$$

where  $N$  is the number of elements in the array,  $x_i$  is the  $i$ th element in the array, and  $\mu$  is the mean of the array.

which can be implemented quite neatly in rust, using iterators, and their respective methods.

```
fn std_dev(arr: &[T]) -> T {
    let sum = arr.iter().sum::<T>();
    let mean = sum / arr.len() as T;
    let variance = arr.iter().map(|x| (x - mean).powi(2)).sum::<T>() / arr.len() as T;
    return variance.sqrt()
}
```

#### 0.9.3.6. Delayed Auto Shift

Delayed auto shift (DAS for short) is a technique implemented in tetris, where you wait for a period of time before starting to move the pieces, while the key is being held down, bypassing the operating systems repeat rate. This is useful for optimizing movements in games similar to DoubleTapp, or tetris, people can customize their DAS and their ARR(auto repeat rate) to be optimal for their own reaction time, so if they need to move a piece they can move it to the corners very quickly, but only after X time has passed, instead of the OS default of 1 second for delay and 100ms per repeat, in my algorithm I used the provided javascript api's of setTimeout and setInterval, wrapped inside an asynchronous function to allow for multiple consecutive inputs, I separately handle keyDown and keyUp events, where on key down the interval is added to an array of intervals (thanks to javascripts type safety), in which the interval is cleared when an OS keyUP is detected, this comes with caveats as there are operating systems which send these events at different times, which can introduce some uncertainty. But due to the timings being customizable, this isn't much of a problem.

```
// Example for one direction, repeated for others
case $state.keycodes.wU:
    if (dasIntervals[0] == false) {
        dasIntervals[0] = setTimeout(() => {
            dasIntervals[0] = setInterval(() => {
                wcursorY = Math.max(wcursorY - 1, 0);
                if ($state.gameMode === 'multiplayer') {
                    ws.send(JSON.stringify({
                        type: 'Move',
                        data: { player_id: `${temp_id}`, action: 'CursorBlueUp' }
                    }));
                }
                moves.push(['CursorBlueUp', Date.now() - lastActionTime]);
                lastActionTime = Date.now();
            }, $state.das);
        }, $state.dasDelay);
    }
}
```

#### 0.9.4. Database Design and Queries

Figure 7: Entity Relationship Model - Database schema showing relationships between game entities

ERM

Centre Number: 22147

20

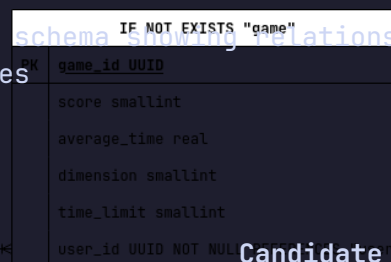
Candidate Number: 9255

IF NOT EXISTS "user"

PK id UUID

IF NOT EXISTS "session"

PK ssid UUID



#### 0.9.4.1. User Authentication Queries

```
SELECT id, password FROM "user" WHERE username = $1
```

this query is quite simple, it just selects the id and password from the user table, where the username is the same as the one provided, as the password is hashed before being stored, this method is secure. additionally it is run on the server side, preventing any XSS attacks, or SQL injections.

#### 0.9.4.2. User Registration Query

```
INSERT INTO "user" (id, username, password) VALUES ($1, $2, $3)
```

another simple query, it just inserts the id, username and password into the user table, again, the password is hashed before being stored, this method is secure.

#### 0.9.4.3. Session Management

```
INSERT INTO session (ssid, user_id, expiry_date)
VALUES ($1, $2, NOW() + INTERVAL '7 DAYS')
```

another simple query, although it adds expiry date to the session, preventing ugly rust code

```
SELECT u.id, u.username, u.admin, u.cheater
FROM "user" u
INNER JOIN session s ON u.id = s.user_id
WHERE s.ssid = $1 AND s.expiry_date > NOW()
```

this query is quite pretty, it looks for all sessions that fit the ssid, and then checks if the expiry date is greater than the current date, if it is, then the user is authenticated, and the user id, username, and admin status is returned.

#### 0.9.4.4. Leaderboard Queries

```
-- Get global leaderboard
SELECT "game".score, "user".username
FROM "game"
JOIN "user" ON "game".user_id = "user".id
WHERE dimension = $1
AND time_limit = $2
ORDER BY score
OFFSET ($3 - 1) 100
FETCH NEXT 100 ROWS ONLY
-- Get user's personal scores
SELECT "game".score, "user".username
FROM "game"
JOIN "user" ON "game".user_id = "user".id
WHERE dimension = $1
AND time_limit = $2
AND "user".id = $4
ORDER BY score
OFFSET ($3 - 1) 100
FETCH NEXT 100 ROWS ONLY
```

these queries use postgresSQL's pagination function, which allows the leaderboards to be paginated, instead of loading all the data into memory, which would be very slow and inefficient. additionally the queries are very readable, I selected 100 rows as it is a good balance and takes up about a page of space.

#### 0.9.4.5. Game Submission

```
INSERT INTO "game" (game_id, score, average_time, dimension, time_limit, user_id)
VALUES ($1, $2, $3, $4, $5, $6)
```

self explanatory.

#### 0.9.4.6. Statistics Trigger

```
CREATE OR REPLACE FUNCTION update_statistics_on_game_insert()
RETURNS TRIGGER AS $$
BEGIN
UPDATE user_statistics
SET
games_played = games_played + 1,
highest_score = GREATEST(highest_score, NEW.score)
WHERE user_id = NEW.user_id;
UPDATE statistics
SET
total_timings = total_timings + NEW.average_time,
total_score = total_score + NEW.score,
games_played = games_played + 1;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER game_insert_trigger
AFTER INSERT ON game
FOR EACH ROW EXECUTE FUNCTION update_statistics_on_game_insert();
```

this trigger is used to update both user statistics, and global statistics, when a game is submitted, it is inserted into the game table, and then the trigger is called to update the statistics. a game is only submitted when it is verified and guaranteed to be a valid game, so the statistics do not include cheaters. additionally you do have to be logged in to submit a game, so the statistics are only updated for logged in users.

#### 0.9.5. Data Structures

##### 0.9.5.1. Circular Queue

A queue is a data structure following the FIFO (first in first out) principle, where you use a sized array, along with variables to store the capacity, front & back of the array, when a file is queued, the file is put onto the index of the back of the array, and then the back index is added to % capacity unless the back becomes equal to the front, in which the queue returns an error instead, this allows for a non resizable array, which allows a set amount of elements to be queued, but not more than the size of the array, allowing for efficient memory management

##### 0.9.5.2. HashMap

A hash table (colloquially called a hashmap) is an array that is abstracted over by a “hashing” function, which outputs an index based on an output, usually the hash function aims to be as diverse as possible, but you can also write special hash functions that are more efficient for your given data types.

##### 0.9.5.3. Option/Result Types

an Optional type, is a simple data structure that allows for beautiful error handling, an Option type wraps the output data, allowing for the error to be handled before trying to manipulate data, i.e in a Some(data) or None, where None means that the data was nonexistent, or we can use a result type to handle errors down the stack, where

we can pass the error with `Err(e)` and `Ok(d)`, so if one part of the function layer breaks we can know exactly where it errored and softly handle the error if needed

#### 0.9.6. Diagrams

#### 0.9.7. Frontend

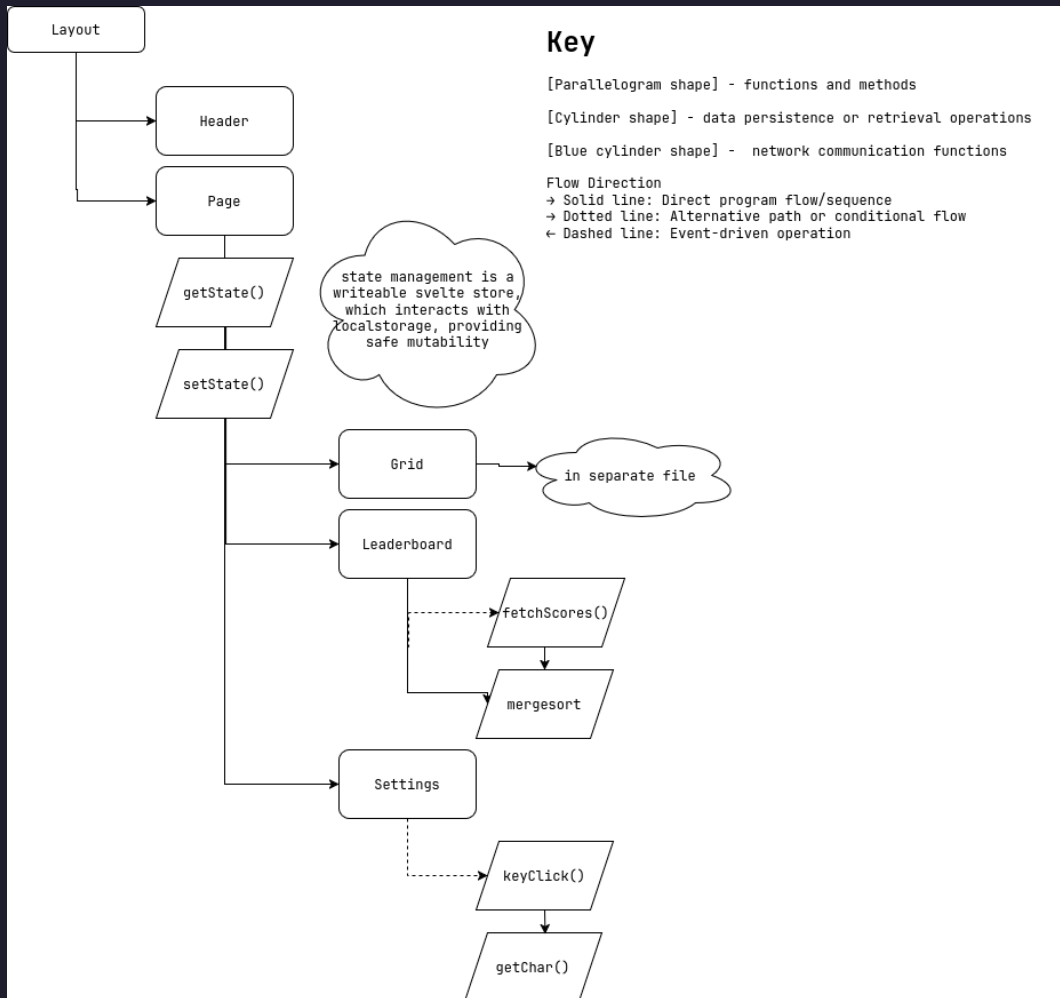


Figure 8: Client Component and Flow diagram



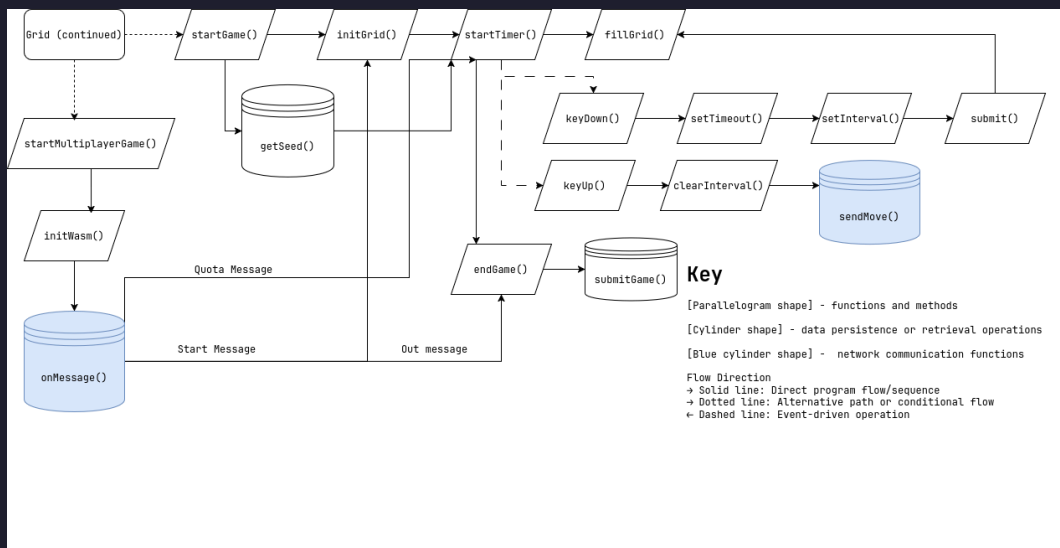


Figure 9: Grid Component

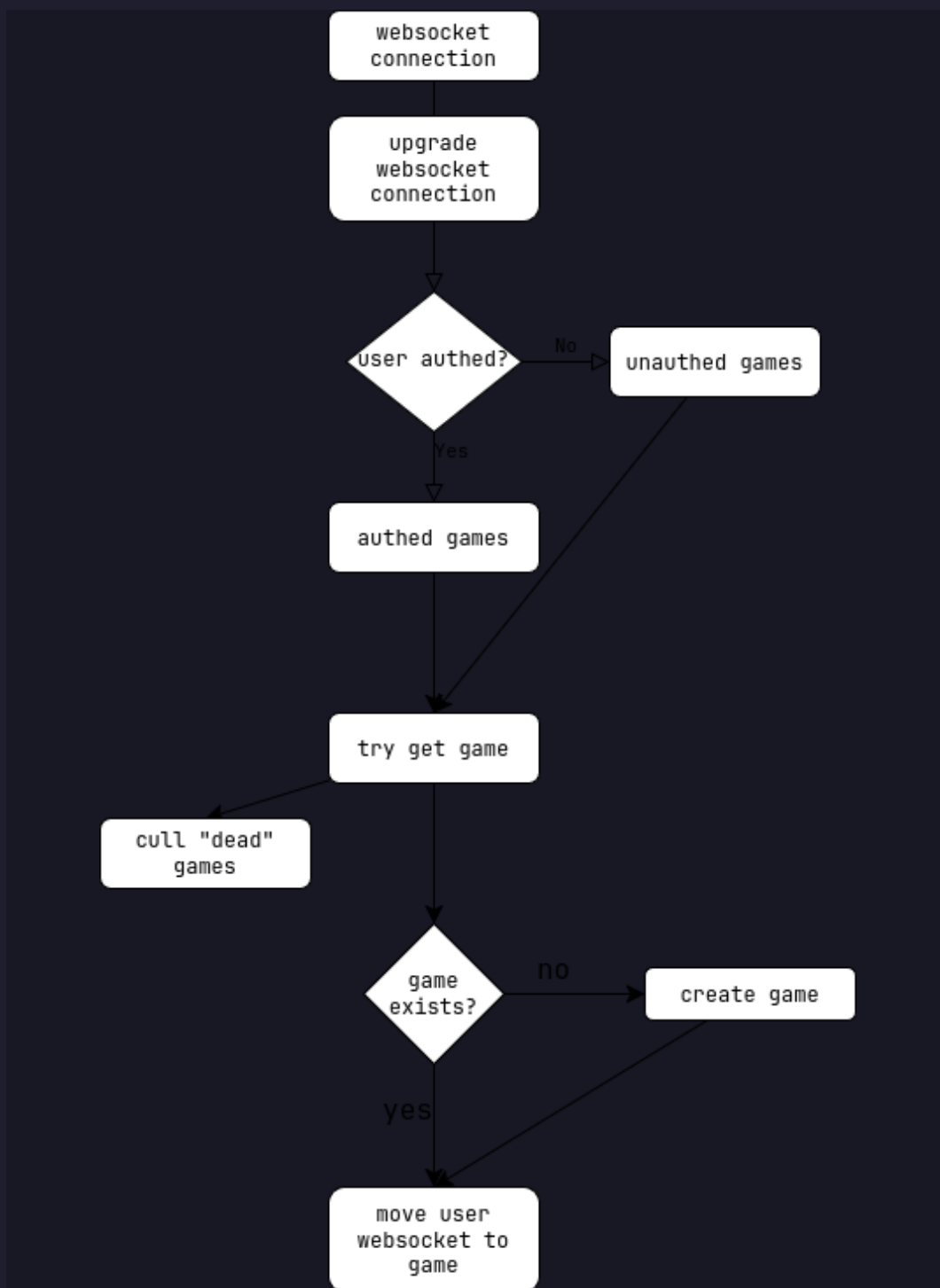


Figure 10: Game Handler Flowchart

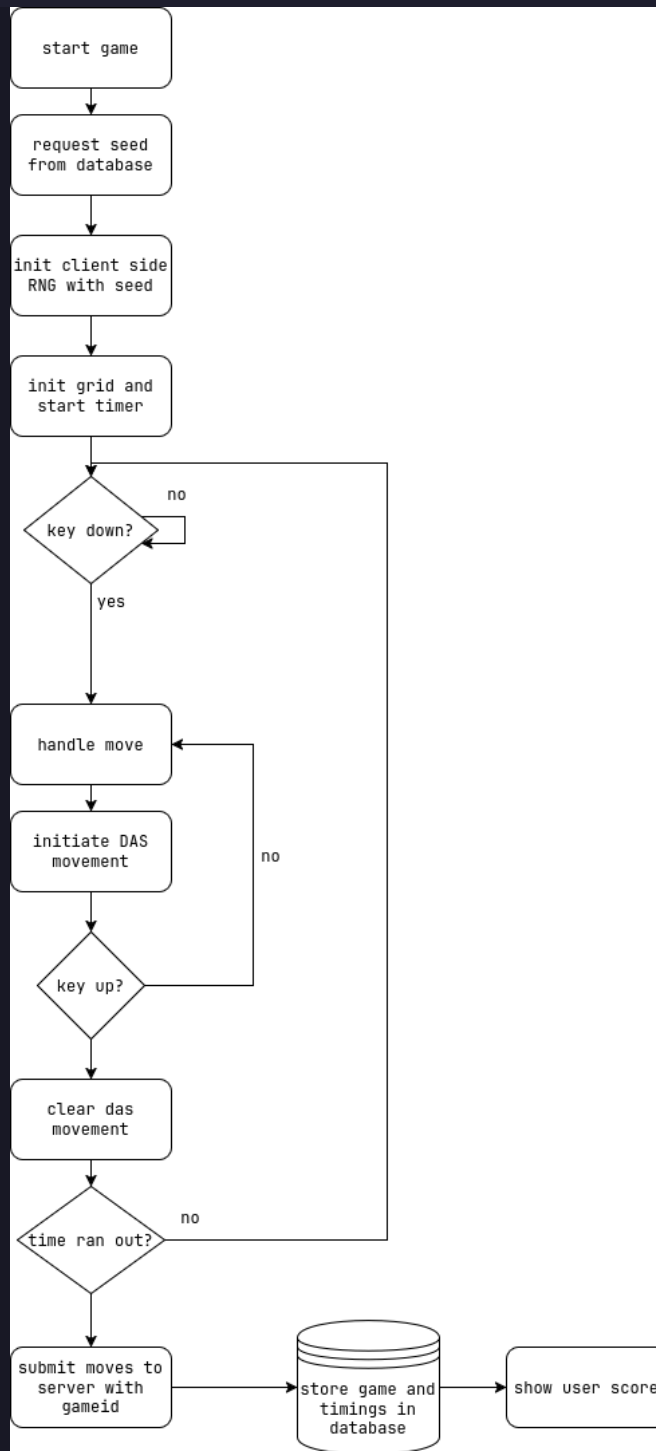


Figure 11: Singleplayer Game Flowchart

### 0.9.8. Backend

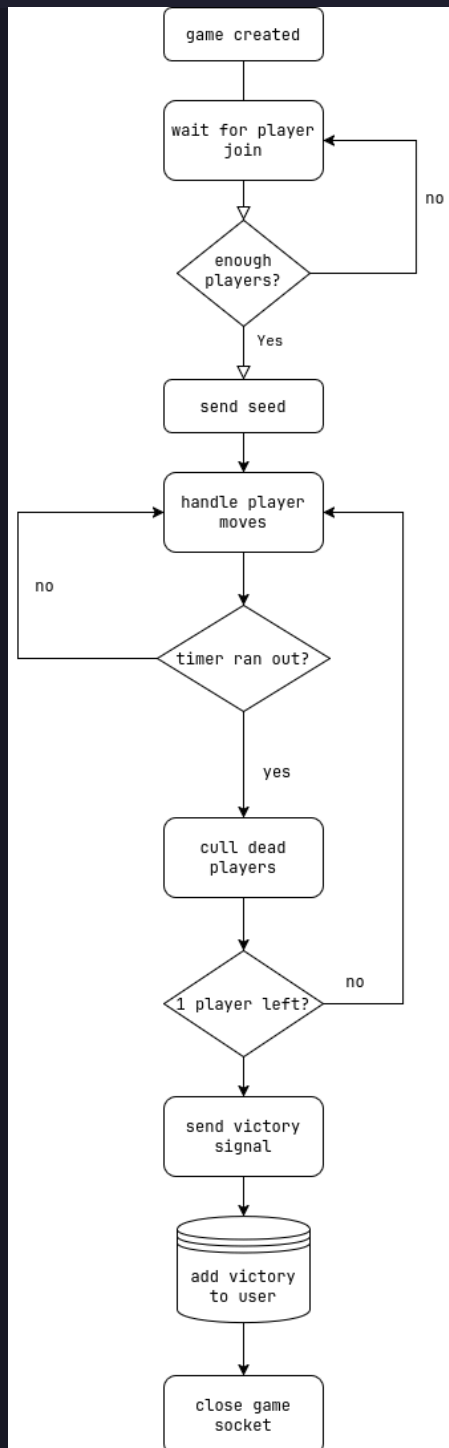


Figure 12: Multiplayer Game Flowchart

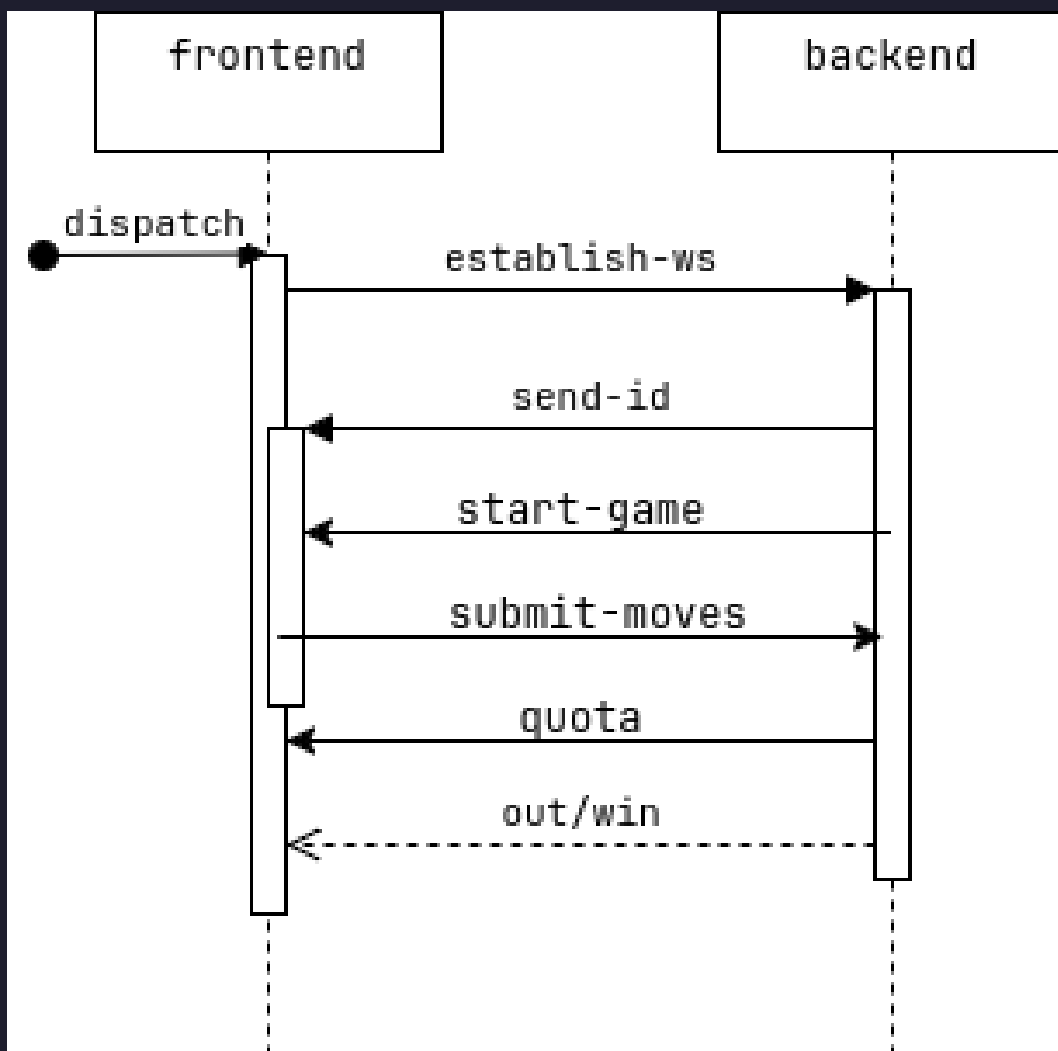


Figure 13: WebSocket message diagram

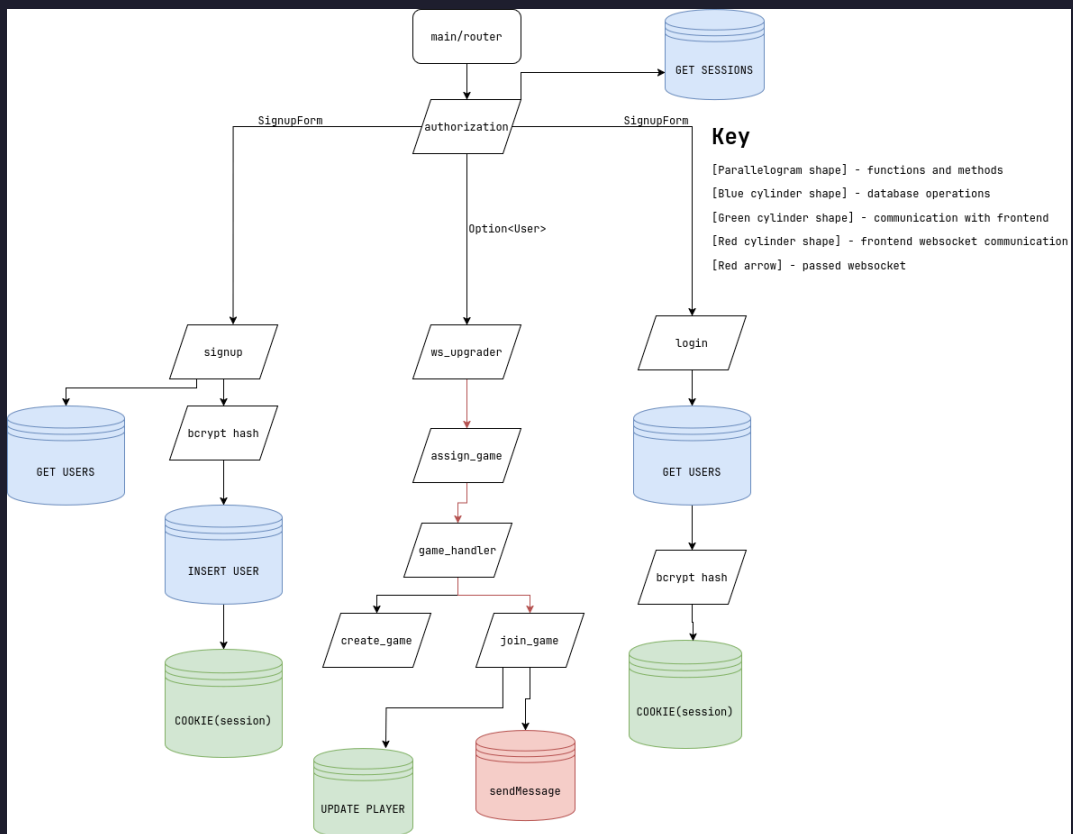


Figure 14: Backend Multiplayer Flowchart

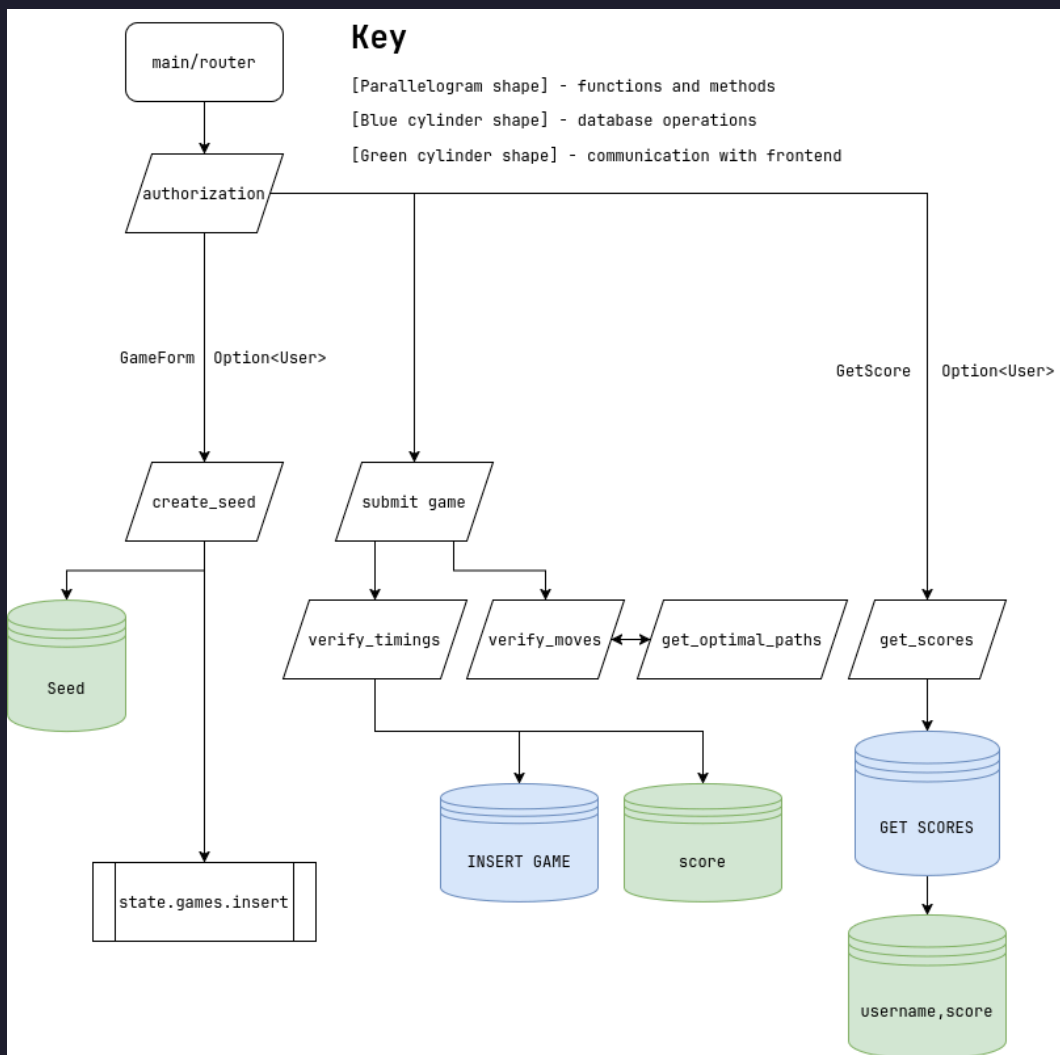


Figure 15: Backend Flowchart

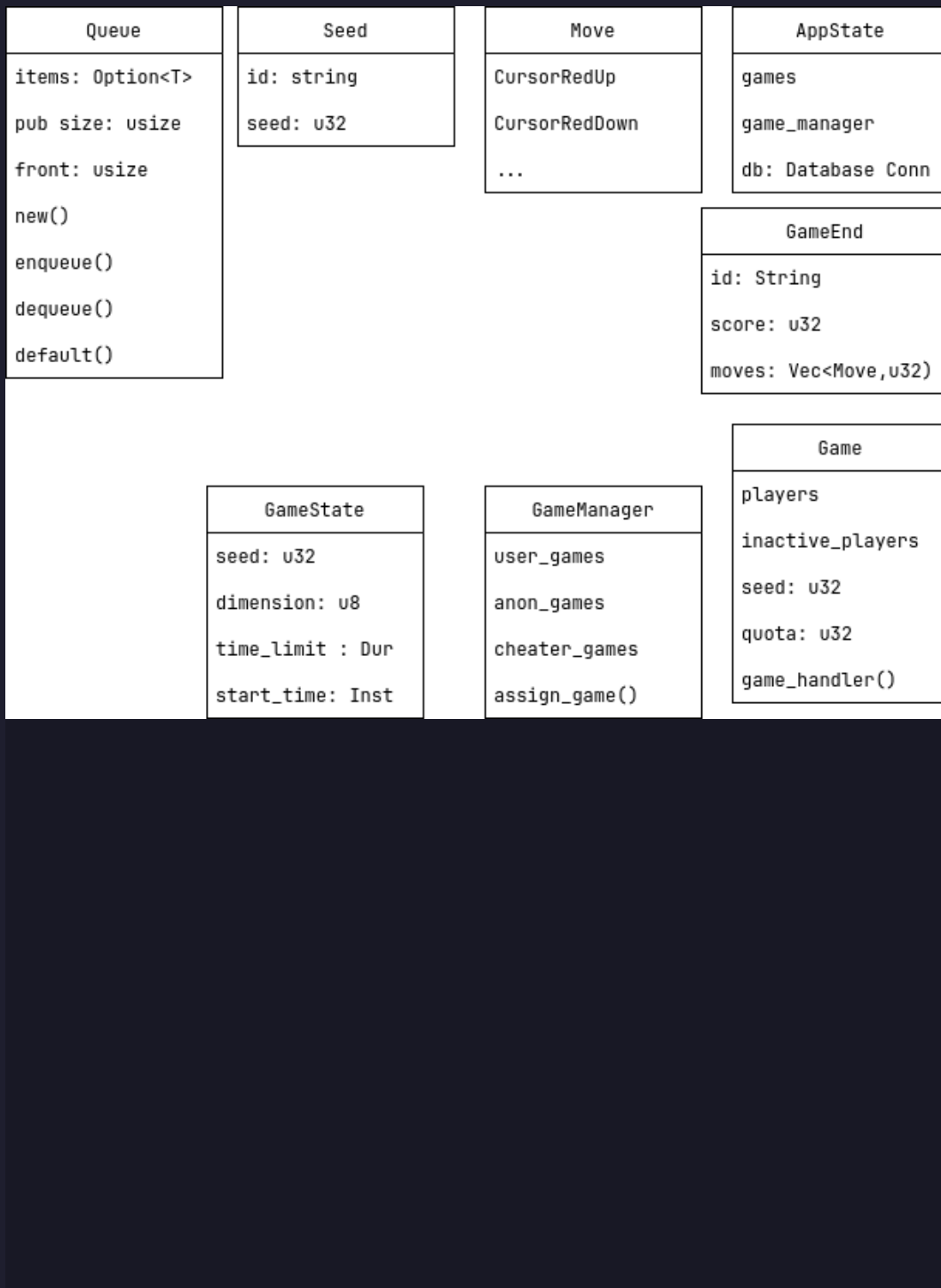


Figure 16: Class Diagram

## 0.10. Technical Solution

### 0.10.1. Code Contents

Component	Description	Path/Location
-----------	-------------	---------------



<u>Grid</u>	Core game grid display and interaction component, handles cursor movement, tile activation, and game state	/src/lib/Grid.svelte
<u>Authentication</u>	User registration, login, and session management	src/routes/signup/+page.svelte, backend/src/misc.rs
<u>Leaderboard</u>	Leaderboard component, displays the leaderboard	src/routes/leaderboard/+page.svelte
<u>Settings</u>	Settings component, displays the settings	src/routes/settings/+page.svelte
<u>Singleplayer Game Management</u>	Singleplayer game management component, handles the singleplayer game	backend/src/main.rs
<u>Multiplayer Game Management</u>	Multiplayer game management component, handles the multiplayer game	backend/src/game.rs
<u>Database Models</u>	Database models, defines the database schema	backend/src/models.rs
<u>Server Routing</u>	Server routing, defines the server routes	backend/src/main.rs
<u>Backend Error Handling</u>	Backend error handling, handles errors in the backend	backend/src/error.rs
<u>WASM</u>	WASM implementation, used for the PRNG	xoshiro-wasm/src/lib.rs, pkg/*
<u>Queue</u>	Queue implementation, used to manage game states	backend/src/misc.rs

### 0.10.2. Skill table

Group	Skill	Description	Link/(s)
A	Complex Data Models	Interlinked tables in database, along with complex queries	<a href="#">Database Models</a> , <a href="#">Authentication</a> , <a href="#">Leaderboard</a>
A	Hash Tables	Hashmaps used to map ULID's to games and user websockets	<a href="#">Multiplayer Game Management</a> , <a href="#">Singleplayer Game Management</a>
A	Queue	Circular queue used to manage game states	<a href="#">Queue</a>
A	Hashing	Hash function used to hash passwords	<a href="#">Authentication</a>
A	Complex Mathematical Model	Implementation of a PRNG	<a href="#">WASM</a>
A	Complex Mathematical Model	MergeSort implementation for leaderboard	<a href="#">Leaderboard</a>
A	Complex Control Model	Websocket Future Pattern Matching, (scheduling/pattern matching)	<a href="#">Multiplayer Game Management</a>
A	Complex OOP model	game handler class, grid class, user class, etc.	<a href="#">Game Handler</a> , <a href="#">Grid</a> , <a href="#">Multiplayer Game Management</a> , <a href="#">Database Models</a>
A	Complex client-server model	complex HTTP request handling, including deserializing and parsing JSON objects	<a href="#">Server Routing</a> , <a href="#">Backend Error Handling</a> , <a href="#">Authentication</a> , <a href="#">Singleplayer Game Management</a>
A	Complex client-server model	Websocket handling, including sending and receiving messages, and transfer of websockets between threads	<a href="#">Multiplayer Game Management</a>

A	Complex client-server model	Authentication Middleware	<u>Authentication</u>
B	Simple Mathematical Model	Game Timing and Score Calculation	<u>Game Handler</u>

### 0.10.3. Completeness of Solution

### 0.10.4. Code Quality

my coding style follows rust's programming principles, i.e error handling through result and option types, and a focus on readability and maintainability, i.e i use descriptive variable names, and i try to comment my code to explain why behind the code, i also try to use meaningful variable names, and i try to keep functions small and focused, i.e single responsibility.

for error handling i use result and option types, i try to handle errors in the frontend and backend, and i try to use meaningful error messages, and i try to keep the code clean and readable, allowing for easier debugging and maintenance, i use the thiserror crate to define custom a custom error type, `AppError`, which is used to handle all errors in the backend, i also use the axum crate to handle errors in the backend, additionally `AppError` implements `IntoResponse`, which allows for handling of errors with constructing HTTP and websocket responses.

one particular example of performance optimizations is in the Multiplayer Game Management section, I use the `tokio::select` macro to handle the websocket messages and game states, this allows for the websocket messages and game states to be handled concurrently, and the `tokio::sync::mpsc` crate to send the websocket to the game handler thread, this allows for the websocket to be sent to the game handler thread without blocking the main thread, the `tokio::select` macro brings great improvements to performance, as it is non-blocking and only runs when there is an available event.

additionally I have used rust, which is a systems programming language with performance on-par with c++ and alternatives, and used libraries known for high performance. particularly `axum`, which is currently the 8 fastest web framework, per the [techempower framework](<https://www.techempower.com/benchmarks/#hw=ph&test=composite&section=data-r23>) benchmark, and `tokio`, which is a high-performance asynchronous runtime for Rust. `svelte` is also known for performance, and is one of the fastest frontend framework for building user interfaces.

additionally i use `scc` Hashmaps, instead of rust's standard library hashmaps, which perform better in concurrent environments, and are more memory efficient.

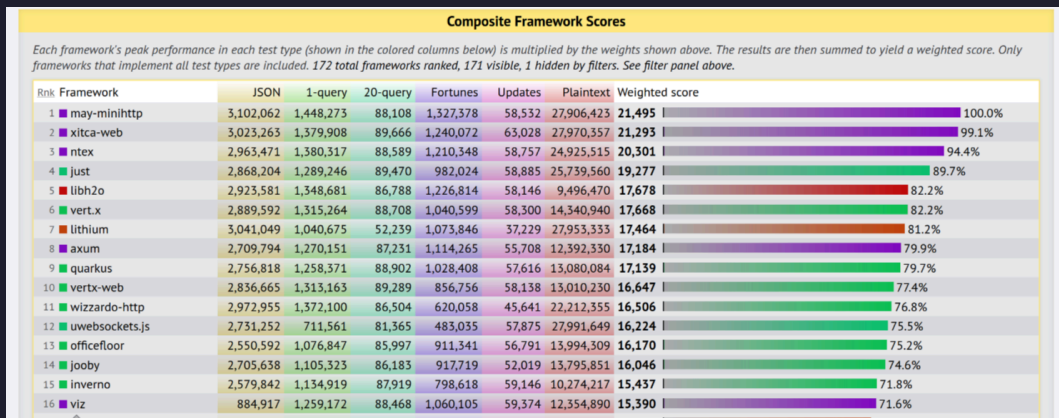


Figure 17: TechEmpower Framework Benchmark

## 0.10.5. Source Code

### 0.10.5.1. Grid Component

rust

```

1 <script lang="ts">
2   import Clock from 'svelte-material-icons/Timer.svelte';
3   import Trophy from 'svelte-material-icons/Trophy.svelte';
4   import Dice from 'svelte-material-icons/Dice5.svelte';
5   import Meow from 'svelte-material-icons/ViewGrid.svelte';
6   import Party from 'svelte-material-icons/PartyPopper.svelte';
7   import { browser } from '$app/environment';
8   import { getContext, onMount } from 'svelte';
9   import { json } from '@sveltejs/kit';
10  import { v4 as uuidv4 } from 'uuid';
11  import { Xoshiro256plus } from 'xoshiro';
12
13  async function initWasm() {
14    rng = new Xoshiro256plus(BigInt(69));
15  }
16
17  let rng: Xoshiro256plus;
18  if (browser) {
19    initWasm().catch(console.error);
20  }
21  export let showModal;
22  let state: any = getContext('state');
23  let scoreboard: any = 0;
24  let end = true;
25  let interval: any;
26  let dasIntervals = Array(8).fill(0);
27  let gameStarted = false;
28  let gameId = 0;
29  let time = $state.timeLimit;
30  let score = 0;
31  let quota = 0;
32  let playersLeft = 0;
33  let moves: any = [];
34  let grid = Array(Math.pow($state.size, 2)).fill(false);
35  let cGrid = Array(Math.pow($state.size, 2)).fill('neutral');
36  let wcursorX = 0;
37  let wcursorY = 0;
38  let acursorX = $state.size - 1;

```

```

39 let acursorY = $state.size - 1;
40 let lastActionTime = 0;
41 let temp_id: String = '';
42 let ws: WebSocket;
43 const initGrid = () => {
44   gameStarted = false;
45   wcursorX = 0;
46   wcursorY = 0;
47   acursorX = $state.size - 1;
48   acursorY = $state.size - 1;
49
50   grid = Array(Math.pow($state.size, 2)).fill(false);
51 };
52 const endGame = () => {
53   score = 0;
54   time = $state.timeLimit;
55   wcursorX = 0;
56   wcursorY = 0;
57   acursorX = $state.size - 1;
58   acursorY = $state.size - 1;
59   moves = [];
60   clearInterval(interval);
61
62   // Close WebSocket if in multiplayer mode
63   if ($state.gameMode === 'multiplayer' && ws) {
64     ws.close();
65     temp_id = '';
66   }
67
68   initGrid();
69 };
70 const startGame = () => {
71   switch ($state.gameMode) {
72     case 'timer':
73       gameStarted = true;
74       startTimer();
75       break;
76     case 'multiplayer':
77       startMultiplayerGame();
78     // case 'pulse':
79     // case 'endless':
80   }
81 };
82 const startMultiplayerGame = () => {
83   if (ws) {
84     ws.close();
85   }
86   ws = new WebSocket('/ws/game');
87   ws.onopen = (e) => {
88     console.log('WebSocket opened');
89   };
90   ws.onmessage = (e) => {
91     const data = e.data;
92
93     try {
94       const message = JSON.parse(data);
95       switch (message.type) {
96         case 'Start':

```

```

97     console.log('Game starting with seed:', message.data);
98     gameStarted = true;
99     rng = new Xoshiro256plus(BigInt(message.data));
100    time = 5;
101    wcursorX = 0;
102    wcursorY = 0;
103    acursorX = $state.size - 1;
104    acursorY = $state.size - 1;
105    interval = setInterval(() => {
106        time -= 1;
107        if (time <= 0) {
108            clearInterval(interval);
109        }
110    }, 1000);
111    fillGrid($state.size);
112    break;
113 case 'Quota':
114     console.log(
115         'Quota update:',
116         message.data.quota,
117         'players left:',
118         message.data.players_left
119     );
120     quota = message.data.quota;
121     playersLeft = message.data.players_left;
122     time = 5;
123     score = 0;
124     clearInterval(interval);
125     interval = setInterval(() => {
126         time -= 1;
127         if (time <= 0) {
128             clearInterval(interval);
129         }
130     }, 1000);
131     break;
132 case 'Move':
133     console.log('Received move:', message.data);
134     break;
135 case 'Out':
136     console.log('player out placed', message.data);
137     end = false;
138     scoreboard = message.data;
139     ws.close();
140     break;
141 case 'Win':
142     console.log("you won!!!!", message.data);
143     end = false;
144     scoreboard = 1;
145     ws.close();
146     break;
147 case 'ID':
148     console.log('Received ID:', message.data);
149     temp_id = message.data;
150     break;
151 case 'Ping':
152     console.log('Received ping');
153     break;
154 default:

```

```

155         console.log('Unknown message type:', message);
156     }
157 } catch (err) {
158     console.error('Failed to parse message:', err);
159 }
160 };
161 ws.addEventListener('close', (e) => {
162     ws.close();
163     temp_id = '';
164 });
165 };
166 const startTimer = async () => {
167     let data = { dimension: $state.size, time_limit: $state.timeLimit };
168     await fetch('/api/get-seed', {
169         method: 'POST',
170         headers: {
171             'Content-Type': 'application/json'
172         },
173         body: JSON.stringify(data)
174     })
175     .then((res) => {
176         return res.json();
177     })
178     .then((data) => {
179         rng = new Xoshiro256plus(BigInt(data.seed));
180         gameId = data.id;
181     });
182
183     wcursorX = 0;
184     wcursorY = 0;
185     acursorX = $state.size - 1;
186     acursorY = $state.size - 1;
187
188     fillGrid($state.size);
189     time = $state.timeLimit;
190     interval = setInterval(async () => {
191         time -= 1;
192         if (time == 0) {
193             end = false;
194             await fetch('/api/submit-game', {
195                 method: 'POST',
196                 headers: {
197                     'Content-Type': 'application/json'
198                 },
199                 body: JSON.stringify({ id: gameId, moves: moves, score: score })
200             })
201             .then((res) => {
202                 return res.json();
203             })
204             .then((data) => {
205                 scoreboard = data;
206             })
207             .catch((err) => console.error('wahrt'));
208             moves = [];
209             clearInterval(interval);
210         }
211     }, 1000);
212 };

```

```

213 const submit = (time: any) => {
214   if (!gameStarted && $state.gameMode === 'timer') {
215     lastActionTime = Date.now();
216     startGame();
217     return;
218   }
219   if ($state.gameMode === 'timer') {
220     moves.push(['Submit', time]);
221   } else if ($state.gameMode === 'multiplayer') {
222     ws.send(
223       JSON.stringify({ type: 'Move', data: { player_id: `${temp_id}`, action: 'Submit' } })
224     );
225   }
226   if (end) {
227     let wIndex = wcursorX * $state.size + wcursorY;
228     let aIndex = acursorX * $state.size + acursorY;
229     let wStatus = grid[wIndex];
230     let aStatus = grid[aIndex];
231     if (wStatus && aStatus && (wcursorX !== acursorX || wcursorY !== acursorY)) {
232       cGrid[wIndex] = 'correct';
233       cGrid[aIndex] = 'correct';
234       let count = 0;
235       while (count < 2) {
236         let x = Math.floor(rng.next() * $state.size);
237         let y = Math.floor(rng.next() * $state.size);
238         if (
239           !grid[x * $state.size + y] &&
240           (wIndex !== x * $state.size + y || aIndex !== x * $state.size + y)
241         ) {
242           grid[x * $state.size + y] = true;
243           count += 1;
244         }
245       }
246       grid[wIndex] = false;
247       grid[aIndex] = false;
248       score += 1;
249     } else {
250       if (wStatus && aStatus) {
251         cGrid[wIndex] = 'incorrect';
252       } else if (wStatus) {
253         cGrid[aIndex] = 'incorrect';
254         cGrid[wIndex] = 'correct';
255       } else if (aStatus) {
256         cGrid[wIndex] = 'incorrect';
257         cGrid[aIndex] = 'correct';
258       } else {
259         cGrid[wIndex] = 'incorrect';
260         cGrid[aIndex] = 'incorrect';
261       }
262       score = 0;
263     }
264     setTimeout(() => {
265       cGrid[wIndex] = 'neutral';
266       cGrid[aIndex] = 'neutral';
267     }, 150);
268   }
269 };
270 const onKeyUp = (e: any) => {

```



```

271     let i = 0;
272     switch (e.key) {
273         case $state.keycodes.wU:
274             i = 0;
275             break;
276         case $state.keycodes.wD:
277             i = 1;
278             break;
279         case $state.keycodes.wL:
280             i = 2;
281             break;
282         case $state.keycodes.wR:
283             i = 3;
284             break;
285         case $state.keycodes.aU:
286             i = 4;
287             break;
288         case $state.keycodes.aD:
289             i = 5;
290             break;
291         case $state.keycodes.aL:
292             i = 6;
293             break;
294         case $state.keycodes.aR:
295             i = 7;
296             break;
297     }
298     clearInterval(dasIntervals[i]);
299     dasIntervals[i] = false;
300 };
301 const onKeyDown = (e: any) => {
302     if (!gameStarted && $state.gameMode === 'multiplayer') {
303         return;
304     }
305     const timeDiff = Date.now() - lastActionTime;
306     switch (e.key) {
307         case $state.keycodes.wU:
308             if (dasIntervals[0] == false) {
309                 dasIntervals[0] = setTimeout(() => {
310                     dasIntervals[0] = setInterval(() => {
311                         wcursorY = Math.max(wcursorY - 1, 0);
312                         if ($state.gameMode === 'multiplayer') {
313                             ws.send(
314                                 JSON.stringify({
315                                     type: 'Move',
316                                     data: { player_id: `${temp_id}`, action: 'CursorBlueUp' }
317                                 })
318                             );
319                         }
320                         moves.push(['CursorBlueUp', Date.now() - lastActionTime]);
321                         lastActionTime = Date.now();
322                     }, $state.das);
323                 }, $state.dasDelay);
324             }
325             wcursorY = Math.max(wcursorY - 1, 0);
326             if ($state.gameMode === 'multiplayer') {
327                 ws.send(
328                     JSON.stringify({

```

```

329         type: 'Move',
330         data: { player_id: `${temp_id}`, action: 'CursorBlueUp' }
331     })
332 );
333 }
334 moves.push(['CursorBlueUp', timeDiff]);
335 lastActionTime = Date.now();
336 break;
337 case $state.keycodes.wD:
338     if (dasIntervals[1] == false) {
339         dasIntervals[1] = setTimeout(() => {
340             dasIntervals[1] = setInterval(() => {
341                 wcursorY = Math.min(wcursorY + 1, $state.size - 1);
342                 if ($state.gameMode === 'multiplayer') {
343                     ws.send(
344                         JSON.stringify({
345                             type: 'Move',
346                             data: { player_id: `${temp_id}`, action: 'CursorBlueDown' }
347                         })
348                     );
349                 }
350                 moves.push(['CursorBlueDown', Date.now() - lastActionTime]);
351                 lastActionTime = Date.now();
352             }, $state.das);
353             }, $state.dasDelay);
354     }
355     wcursorY = Math.min(wcursorY + 1, $state.size - 1);
356     if ($state.gameMode === 'multiplayer') {
357         ws.send(
358             JSON.stringify({
359                 type: 'Move',
360                 data: { player_id: `${temp_id}`, action: 'CursorBlueDown' }
361             })
362         );
363     }
364     moves.push(['CursorBlueDown', timeDiff]);
365     lastActionTime = Date.now();
366     break;
367 case $state.keycodes.wL:
368     if (dasIntervals[2] == false) {
369         dasIntervals[2] = setTimeout(() => {
370             dasIntervals[2] = setInterval(() => {
371                 wcursorX = Math.max(wcursorX - 1, 0);
372                 if ($state.gameMode === 'multiplayer') {
373                     ws.send(
374                         JSON.stringify({
375                             type: 'Move',
376                             data: { player_id: `${temp_id}`, action: 'CursorBlueLeft' }
377                         })
378                     );
379                 }
380                 moves.push(['CursorBlueLeft', Date.now() - lastActionTime]);
381                 lastActionTime = Date.now();
382             }, $state.das);
383             }, $state.dasDelay);
384     }
385     wcursorX = Math.max(wcursorX - 1, 0);
386     if ($state.gameMode === 'multiplayer') {

```

```

387     ws.send(
388         JSON.stringify({
389             type: 'Move',
390             data: { player_id: `${temp_id}`, action: 'CursorBlueLeft' }
391         })
392     );
393 }
394 moves.push(['CursorBlueLeft', timeDiff]);
395 lastActionTime = Date.now();
396 break;
397 case $state.keycodes.wR:
398     if (dasIntervals[3] == false) {
399         dasIntervals[3] = setTimeout(() => {
400             dasIntervals[3] = setInterval(() => {
401                 wcursorX = Math.min(wcursorX + 1, $state.size - 1);
402                 if ($state.gameMode === 'multiplayer') {
403                     ws.send(
404                         JSON.stringify({
405                             type: 'Move',
406                             data: { player_id: `${temp_id}`, action: 'CursorBlueRight' }
407                         })
408                     );
409                 }
410                 moves.push(['CursorBlueRight', Date.now() - lastActionTime]);
411                 lastActionTime = Date.now();
412             }, $state.das);
413             }, $state.dasDelay);
414     }
415     wcursorX = Math.min(wcursorX + 1, $state.size - 1);
416     if ($state.gameMode === 'multiplayer') {
417         ws.send(
418             JSON.stringify({
419                 type: 'Move',
420                 data: { player_id: `${temp_id}`, action: 'CursorBlueRight' }
421             })
422         );
423     }
424     moves.push(['CursorBlueRight', timeDiff]);
425     lastActionTime = Date.now();
426     break;
427 case $state.keycodes.aU:
428     if (dasIntervals[4] == false) {
429         dasIntervals[4] = setTimeout(() => {
430             dasIntervals[4] = setInterval(() => {
431                 acursorY = Math.max(acursorY - 1, 0);
432                 if ($state.gameMode === 'multiplayer') {
433                     ws.send(
434                         JSON.stringify({
435                             type: 'Move',
436                             data: { player_id: `${temp_id}`, action: 'CursorRedUp' }
437                         })
438                     );
439                 }
440                 moves.push(['CursorRedUp', Date.now() - lastActionTime]);
441                 lastActionTime = Date.now();
442             }, $state.das);
443             }, $state.dasDelay);
444     }

```

```

445     acursorY = Math.max(acursorY - 1, 0);
446     if ($state.gameMode === 'multiplayer') {
447         ws.send(
448             JSON.stringify({
449                 type: 'Move',
450                 data: { player_id: `${temp_id}`, action: 'CursorRedUp' }
451             })
452         );
453     }
454     moves.push(['CursorRedUp', timeDiff]);
455     lastActionTime = Date.now();
456     break;
457 case $state.keycodes.aD:
458     if (dasIntervals[5] == false) {
459         dasIntervals[5] = setTimeout(() => {
460             dasIntervals[5] = setInterval(() => {
461                 acursorY = Math.min(acursorY + 1, $state.size - 1);
462                 if ($state.gameMode === 'multiplayer') {
463                     ws.send(
464                         JSON.stringify({
465                             type: 'Move',
466                             data: { player_id: `${temp_id}`, action: 'CursorRedDown' }
467                         })
468                     );
469                 }
470                 moves.push(['CursorRedDown', Date.now() - lastActionTime]);
471                 lastActionTime = Date.now();
472             }, $state.das);
473             }, $state.dasDelay);
474     }
475     acursorY = Math.min(acursorY + 1, $state.size - 1);
476     if ($state.gameMode === 'multiplayer') {
477         ws.send(
478             JSON.stringify({
479                 type: 'Move',
480                 data: { player_id: `${temp_id}`, action: 'CursorRedDown' }
481             })
482         );
483     }
484     moves.push(['CursorRedDown', timeDiff]);
485     lastActionTime = Date.now();
486     break;
487 case $state.keycodes.aL:
488     if (dasIntervals[6] == false) {
489         dasIntervals[6] = setTimeout(() => {
490             dasIntervals[6] = setInterval(() => {
491                 acursorX = Math.max(acursorX - 1, 0);
492                 if ($state.gameMode === 'multiplayer') {
493                     ws.send(
494                         JSON.stringify({
495                             type: 'Move',
496                             data: { player_id: `${temp_id}`, action: 'CursorRedLeft' }
497                         })
498                     );
499                 }
500                 moves.push(['CursorRedLeft', Date.now() - lastActionTime]);
501                 lastActionTime = Date.now();
502             }, $state.das);

```

```

503     }, $state.dasDelay);
504 }
505 acursorX = Math.max(acursorX - 1, 0);
506 if ($state.gameMode === 'multiplayer') {
507     ws.send(
508         JSON.stringify({
509             type: 'Move',
510             data: { player_id: `${temp_id}`, action: 'CursorRedLeft' }
511         })
512     );
513 }
514 moves.push(['CursorRedLeft', timeDiff]);
515 lastActionTime = Date.now();
516 break;
517 case $state.keycodes.aR:
518     if (dasIntervals[7] == false) {
519         dasIntervals[7] = setTimeout(() => {
520             dasIntervals[7] = setInterval(() => {
521                 acursorX = Math.min(acursorX + 1, $state.size - 1);
522                 if ($state.gameMode === 'multiplayer') {
523                     ws.send(
524                         JSON.stringify({
525                             type: 'Move',
526                             data: { player_id: `${temp_id}`, action: 'CursorRedRight' }
527                         })
528                     );
529                 }
530                 moves.push(['CursorRedRight', Date.now() - lastActionTime]);
531                 lastActionTime = Date.now();
532             }, $state.das);
533         }, $state.dasDelay);
534     }
535     acursorX = Math.min(acursorX + 1, $state.size - 1);
536     if ($state.gameMode === 'multiplayer') {
537         ws.send(
538             JSON.stringify({
539                 type: 'Move',
540                 data: { player_id: `${temp_id}`, action: 'CursorRedRight' }
541             })
542         );
543     }
544     moves.push(['CursorRedRight', timeDiff]);
545     lastActionTime = Date.now();
546     break;
547 case $state.keycodes.submit:
548     submit(timeDiff);
549     lastActionTime = Date.now();
550     break;
551 case $state.keycodes.reset:
552     end == false ? (end = true) : '';
553     endGame();
554     break;
555 }
556 };
557 const fillGrid = (count: number) => {
558     let placed = 0;
559     while (placed < count) {
560         let x = Math.floor(rng.next() * $state.size);

```

```

561     let y = Math.floor(rng.next() * $state.size);
562     if (grid[x * $state.size + y] == false) {
563         grid[x * $state.size + y] = true;
564         placed += 1;
565     }
566 }
567 };
568 initGrid();
569 </script>
570
571 <div class="">
572   {#if end}
573   <div class="flex flex-row text-3xl text-text justify-between py-2">
574     <div class="flex flex-row items-center">
575       <Clock />
576       <div class="px-2 {time < 3 ? (time < 2 ? 'text-red' : 'text-peach') : 'text-green'}">
577         {time}
578       </div>
579     </div>
580     <div class="flex flex-row items-center">
581       <Trophy />
582       <div class="px-2">
583         {$state.gameMode === 'multiplayer' ? `${score}/${quota} (${playersLeft})` : score}
584       </div>
585     </div>
586   </div>
587   <div class="flex flex-col items-center">
588     <div class="relative w-fit h-fit">
589       {#if $state.gameMode === 'multiplayer' && (!temp_id || !gameStarted)}
590       <div
591         class="absolute top-0 left-0 right-0 bottom-[4.5rem] flex items-center justify-center
592         z-10 bg-base/80"
593       >
594         {#if !temp_id}
595         <button
596           class="px-4 py-2 rounded-lg transition-colors duration-300 bg-lavender text-mantle
597           hover:bg-rosewater"
598           on:click={startMultiplayerGame}
599         >
600           Join Game
601         </button>
602         {/if}
603         <div class="text-text text-3xl flex flex-col items-center gap-4">
604           <div class="flex items-center gap-2">Waiting for players...</div>
605         </div>
606       </div>
607     </div>
608     <!-- svelte-ignore ally-autofocus -->
609     <div class="w-fit h-fit flex flex-col" autofocus>
610       {#each Array($state.size) as _, col}
611       <div class="w-fit h-fit flex flex-row">
612         {#each Array($state.size) as _, row}
613         <div
614           id={grid[row * $state.size + col]}
615           class="{cGrid[row * $state.size + col] === 'correct'
616             ? 'bg-green'
617             : cGrid[row * $state.size + col] === 'incorrect'

```

```

617         ? 'bg-red'
618         : grid[row * $state.size + col]
619         ? 'bg-crust'
620         : 'bg-text'}
621
622     w-32 h-32 border-crust border flex items-center justify-center transition-colors
duration-100"
623     >
624     <div
625         class="h-8 w-8 {row == wcursorX && col == wcursorY
626         ? 'border-t-blue border-l-blue border-t-8 border-l-8'
627         : ''} {row == acursorX && col == acursorY
628         ? 'border-b-red border-r-red border-b-8 border-r-8'
629         : ''}"
630     />
631 </div>
632 {/each}
633 </div>
634 {/each}
635 <div class="text-text flex flex-row text-2xl py-4 justify-between">
636     <div class="flex flex-row">
637         <select
638             id="gamemodes"
639             name="modes"
640             class="bg-surface0 px-2"
641             bind:value={$state.gameMode}
642         >
643             <label for="gamemodes" class="pr-4"> GAMEMODE: </label>
644             <option value="timer"> TIME </option>
645             <option value="multiplayer"> MULTIPLAYER </option>
646             <option value="endless"> ZEN </option>
647         </select>
648     </div>
649     <select
650         id="size"
651         name="sizes"
652         class="bg-surface0 px-2"
653         bind:value={$state.size}
654         on:change={() => {
655             endGame();
656         }}
657     >
658         <option value={4}> 4x4 </option>
659         <option value={5}> 5x5 </option>
660         <option value={6}> 6x6 </option>
661     </select>
662     <select
663         id="time"
664         name="times"
665         class="bg-surface0 px-2 {$state.gameMode == 'timer'
666         ? 'bg-surface0'
667         : 'bg-surface0/0 text-crust/0'}"
668         bind:value={$state.timeLimit}
669         on:change={() => {
670             time = $state.timeLimit;
671             endGame();
672         }}
673     >

```

```

674         <option value={30}> 30s </option>
675         <option value={45}> 45s </option>
676         <option value={60}> 60s </option>
677     </select>
678     <button class="bg-surface0 px-2" on:click={endGame}> RESET </button>
679 </div>
680 </div>
681 </div>
682 </div>
683 { :else }
684 <div class="text-text flex align-right flex-col w-96">
685     <div class="text-5xl py-2 font-bold flex items-center border-b-4 border-b-surface0">
686         <Party class="mr-4" /> game ended
687     </div>
688     <div class="text-4xl py-2 flex items-center justify-between">
689         score: {score}
690         <div class="text-overlay1">
691             { #if $state.gameMode === 'multiplayer' && scoreboard > 0 }
692             Position: #{scoreboard}
693             { :else }
694             #{scoreboard}
695             { /if }
696         </div>
697     </div>
698     <div class="flex-col items-center text-3xl justify-between pb-2">
699         <div class="flex items-center my-1">
700             <Dice /> gamemode:
701             <div class="ml-1 text-overlay1"> { $state.gameMode } </div>
702         </div>
703         <div class="flex items-center my-1">
704             <Meow /> size:
705             <div class="ml-1 text-overlay1"> { $state.size } x { $state.size } </div>
706         </div>
707         <div class="flex items-center my-1">
708             { #if $state.gameMode === 'timer' }
709             <Clock /> time:
710             <div class="ml-1 text-overlay1"> { $state.timeLimit } s </div>
711             { /if }
712         </div>
713     </div>
714     <button
715         class="text-2xl h-12 my-2 bg-blue/80 hover:bg-blue border-rosewater transition-colors
duration-150 font-bold"
716         on:click={() => {
717             end = true;
718             endGame();
719         }}
720     >
721     submit score?
722 </button>
723 <button
724     class="text-2xl h-12 my-2 bg-mauve/80 hover:bg-mauve border-rosewater transition-colors
duration-150 font-bold"
725     on:click={() => {
726         end = true;
727         endGame();
728     }}
729     >

```



```

730     play again?
731     </button>
732 </div>
733   {/if}
734 </div>
735
736 <svelte:window on:keydown={onKeyDown} on:keyup={onKeyUp} />
737
738
739
740
741

```

## 0.10.5.2. Authentication

### 0.10.5.2.1. Frontend

```

1 <script lang="ts">
2   import { goto, invalidateAll } from "$app/navigation";
3   let isSignup = true;
4   let error = "";
5
6   async function handleSubmit(e: SubmitEvent) {
7     e.preventDefault();
8     let data = new URLSearchParams(new FormData(e.target as HTMLFormElement));
9     let path = isSignup ? "/api/user/signup" : "/api/user/login";
10    const res = await fetch(path, {
11      method: "POST",
12      headers: {
13        "Content-Type": "application/x-www-form-urlencoded",
14      },
15      body: data,
16    });
17    if (res?.ok) {
18      await invalidateAll();
19      goto("/");
20    } else {
21      switch (res?.status) {
22        case 409:
23          error = "Username or email already exists";
24          break;
25        case 401:
26          error = "Invalid credentials";
27          break;
28        case 404:
29          error = "not found";
30          break;
31        default:
32          error = "An unknown error occurred";
33          break;
34      }
35    }
36  }
37 </script>
38
39 <div class="min-h-screen w-screen flex items-center justify-center bg-base">
40   <div class="w-full max-w-md mx-4 bg-mantle rounded-xl shadow-xl p-8">

```

```

41     {#if isSignup}
42     <div class="flex flex-col gap-8">
43       <div class="text-center">
44         <h1 class="text-3xl font-medium text-lavender mb-2">Create Account</h1>
45       </div>
46       <form on:submit={handleSubmit} class="flex flex-col gap-6">
47         <div class="flex flex-col gap-4">
48           <input
49             type="text"
50             name="username"
51             placeholder="Username"
52             class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
focus:border-lavender transition-colors"
53           />
54           <input
55             type="password"
56             name="password"
57             placeholder="Password"
58             class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
focus:border-lavender transition-colors"
59           />
60         </div>
61         {#if error}
62         <div class="bg-red/10 border border-red/20 text-red px-4 py-3 rounded-lg text-sm">
63           {error}
64         </div>
65         {/if}
66         <button
67           type="submit"
68           class="w-full px-4 py-3 rounded-lg font-medium bg-lavender text-mantle hover:bg-
rosewater transition-colors"
69         >
70           Sign Up
71         </button>
72       </form>
73       <button
74         on:click={() => (isSignup = false)}
75         class="text-subtext0 hover:text-text transition-colors pt-2"
76       >
77         Already have an account? Login here
78       </button>
79     </div>
80   {/else}
81   <div class="flex flex-col gap-8">
82     <div class="text-center">
83       <h1 class="text-3xl font-medium text-lavender mb-2">Welcome Back!</h1>
84     </div>
85     <form on:submit={handleSubmit} class="flex flex-col gap-6">
86       <div class="flex flex-col gap-4">
87         <input
88           type="username"
89           name="username"
90           placeholder="Username"
91           class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
focus:border-lavender transition-colors"
92         />
93         <input
94           type="password"

```

```

95         name="password"
96         placeholder="Password"
97         class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
focus:border-lavender transition-colors"
98     />
99     </div>
100     {#if error}
101         <div class="bg-red/10 border border-red/20 text-red px-4 py-3 rounded-lg text-sm">
102             {error}
103         </div>
104     {/if}
105     <button
106         type="submit"
107         class="w-full px-4 py-3 rounded-lg font-medium bg-lavender text-mantle hover:bg-
rosewater transition-colors"
108     >
109         Login
110     </button>
111 </form>
112 <button
113     on:click={() => (isSignup = true)}
114     class="text-subtext0 hover:text-text transition-colors pt-2"
115 >
116     Don't have an account? Sign up here
117 </button>
118 </div>
119 {/if}
120 </div>
121 </div>

```

#### 0.10.5.2.2. Backend

```

1 pub struct SignForm {
2     pub(crate) username: String,
3     #[validate(length(min = 8))]
4     pub(crate) password: String,
5 }
6
7 #[axum::debug_handler]
8 pub async fn signup(
9     State(state): State<Arc<AppState>>,
10    headers: HeaderMap,
11    Form(details): Form<SignForm>,
12 ) -> Result<CookieJar, AppError> {
13     let mut conn = state.db.acquire().await?;
14     let jar = CookieJar::from_headers(&headers);
15     let exists: Option<i64,> = sqlx::query_as("SELECT 1 FROM \"user\" WHERE username = $1")
16         .bind(&details.username)
17         .fetch_optional(&mut *conn)
18         .await?;
19
20     if exists.is_some() {
21         return Err(AppError::Status(StatusCode::CONFLICT));
22     }
23
24     let hashed = bcrypt::hash(details.password, bcrypt::DEFAULT_COST)?;
25     let user_id = uuid::Uuid::new_v4();

```

```

26
27     sqlx::query!(
28         "INSERT INTO \"user\" (id, username, password) VALUES ($1, $2, $3)",
29         user_id,
30         details.username,
31         hashed
32     )
33     .execute(&mut *conn)
34     .await?;
35
36     let session_id = uuid::Uuid::new_v4();
37     sqlx::query!(
38         "INSERT INTO session (ssid, user_id, expiry_date) VALUES ($1, $2, NOW() + INTERVAL '7
39         DAYS')",
40         session_id,
41         user_id
42     )
43     .execute(&mut *conn)
44     .await?;
45     Ok(jar.add(
46         Cookie::build(("session", session_id.to_string()))
47             .path("/")
48             .build(),
49     ))
50 }
51
52 pub async fn login(
53     State(state): State<Arc<AppState>>,
54     jar: CookieJar,
55     Form(details): Form<SignForm>,
56 ) -> Result<CookieJar, AppError> {
57     let mut conn = state.db.acquire().await?;
58
59     let user: Option<(uuid::Uuid, String)> =
60         sqlx::query_as("SELECT id, password FROM \"user\" WHERE username = $1")
61             .bind(&details.username)
62             .fetch_optional(&mut *conn)
63             .await?;
64
65     let (user_id, hashed) = user.ok_or(AppError::Status(StatusCode::UNAUTHORIZED))?;
66
67     if !bcrypt::verify(details.password, &hashed)? {
68         return Err(AppError::Status(StatusCode::UNAUTHORIZED));
69     }
70
71     let session_id = uuid::Uuid::new_v4();
72     sqlx::query!(
73         "INSERT INTO session (ssid, user_id, expiry_date) VALUES ($1, $2, NOW() + INTERVAL '7
74         DAYS')",
75         session_id,
76         user_id
77     )
78     .execute(&mut *conn)
79     .await?;
80     Ok(jar.add(
81         Cookie::build(("session", session_id.to_string()))

```

```

82         .path("/")
83         .build(),
84     ))
85 }
86
87 #[axum::debug_middleware]
88 pub async fn authorization(
89     State(state): State<Arc<AppState>>,
90     headers: HeaderMap,
91     mut request: Request,
92     next: Next,
93 ) -> Result<Response, AppError> {
94     let jar = CookieJar::from_headers(&headers);
95     let user = if let Some(cookie) = jar.get("session") {
96         if let Ok(session_id) = uuid::Uuid::parse_str(cookie.value()) {
97             let mut conn = state.db.acquire().await?;
98             sqlx::query_as!(
99                 crate::models::UserExt,
100                 r#"
101                 SELECT u.id, u.username, u.admin, u.cheater
102                 FROM "user" u
103                 INNER JOIN session s ON u.id = s.user_id
104                 WHERE s.ssid = $1 AND s.expiry_date > NOW()
105                 "#,
106                 session_id
107             )
108             .fetch_optional(&mut *conn)
109             .await?
110         } else {
111             None
112         }
113     } else {
114         None
115     };
116
117     request.extensions_mut().insert(user);
118     let response = next.run(request).await;
119     Ok(response)
120 }

```

### 0.10.5.3. Queue

```

1  #[derive(Debug)]
2  pub struct Queue<T> {
3      items: [Option<T>; 64],
4      pub size: usize,
5      front: usize,
6  }
7
8  impl<T> Queue<T> {
9      pub fn new() -> Self {
10         Self {
11             items: std::array::from_fn(|_| None),
12             size: 0,
13             front: 0,
14         }
15     }

```

```

16
17     pub fn enqueue(&mut self, item: T) -> bool {
18         if self.size == self.items.len() {
19             return false;
20         }
21         let rear = (self.front + self.size) % self.items.len();
22         self.items[rear] = Some(item);
23         self.size += 1;
24         true
25     }
26
27     pub fn dequeue(&mut self) -> Option<T> {
28         if self.size == 0 {
29             return None;
30         }
31         let item = self.items[self.front].take();
32         self.front = (self.front + 1) % self.items.len();
33         self.size -= 1;
34         item
35     }
36 }
37
38 impl<T> Default for Queue<T> {
39     fn default() -> Self {
40         Self::new()
41     }
42 }
43
44
45
46

```

#### 0.10.5.4. Leaderboard

```

1 <script lang="ts">
2   import { onMount } from 'svelte';
3
4   let dimension = 4;
5   let timeLimit = 30;
6   let leaderboard: Array<[string, number]> = [];
7   let currentPage = 1;
8   let userOwned = false;
9   onMount(() => {
10     fetchScores();
11   });
12
13   async function fetchScores() {
14     const res = await fetch(`/api/get_scores`, {
15       method: 'POST',
16       headers: { 'Content-Type': 'application/json' },
17       body: JSON.stringify({ page: currentPage, dimension, time_limit: timeLimit, user_scores:
18         userOwned }),
19     });
20     const data = await res.json();
21     leaderboard = mergesort(data);
22   }

```

```

23 function mergesort(arr: Array<[string, number]>): Array<[string, number]> {
24   if (arr.length < 2) return arr;
25   const mid = Math.floor(arr.length / 2);
26   const left = mergesort(arr.slice(0, mid));
27   const right = mergesort(arr.slice(mid));
28   return merge(left, right);
29 }
30
31 function merge(left: Array<[string, number]>, right: Array<[string, number]>): Array<[string,
number]> {
32   let result = [];
33   while (left.length && right.length) {
34     if (left[0][1] > right[0][1]) {
35       result.push(left.shift());
36     } else {
37       result.push(right.shift());
38     }
39   }
40   return [...result, ...left, ...right];
41 }
42 //TODO make the dimension and timelimit a select
43 </script>
44
45 <div class="min-h-screen bg-mantle text-text p-8">
46   <div class="text-3xl font-bold mb-6">Leaderboards</div>
47   <div class="flex gap-4 items-center mb-6">
48     <div class="font-semibold">Dimension:</div>
49     <select id="size" name="dimension" class="bg-surface0 px-2" bind:value={dimension}>
50       <option value={4}> 4x4 </option>
51       <option value={5}> 5x5 </option>
52       <option value={6}> 6x6 </option>
53     </select>
54     <div class="font-semibold">Time Limit:</div>
55     <select id="size" name="dimension" class="bg-surface0 px-2" bind:value={timeLimit}>
56       <option value={30}> 30s </option>
57       <option value={45}> 45s </option>
58       <option value={60}> 60s </option>
59     </select>
60     <div class="font-semibold">Personal Bests:</div>
61     <input type="checkbox" bind:checked={userOwned} class="bg-surface0 px-2">
62     <button on:click={fetchScores} class="px-4 py-1 rounded bg-green text-text font-semibold
hover:bg-sky">Refresh</button>
63   </div>
64   <table class="min-w-full border-collapse">
65     <thead class="bg-[#1e2030]">
66       <tr>
67         <th class="px-4 py-2 border border-text">Username</th>
68         <th class="px-4 py-2 border border-text">Score</th>
69       </tr>
70     </thead>
71     <tbody>
72       {#each leaderboard as [user, score]}
73       <tr class="hover:bg-text">
74         <td class="px-4 py-2 border border-text">{user}</td>
75         <td class="px-4 py-2 border border-text">{score}</td>
76       </tr>
77     </tbody>
78   </table>

```

```

79 </table>
80 <div>
81   <button on:click={() => {currentPage -= 1; fetchScores()}} disabled={currentPage === 1} > back
   </button>
82   {currentPage}
83   <button on:click={() => {currentPage += 1; fetchScores()}} > next </button>
84 </div>
85 </div>

```

rust

```

1
2 #[derive(Serialize, Deserialize, sqlx::FromRow)]
3 pub struct Score {
4     username: String,
5     score: Option<i16>,
6 }
7
8 #[derive(Serialize, Deserialize)]
9 pub struct GetScore {
10     page: u32,
11     dimension: u8,
12     time_limit: u8,
13     user_scores: bool,
14 }
15
16 #[axum::debug_handler]
17 pub async fn get_scores(
18     State(state): State<Arc<AppState>>,
19     Extension(user): Extension<Option<UserExt>>,
20     Json(data): Json<GetScore>,
21 ) -> Result<Json<Vec<(String, usize)>>, AppError> {
22     let query_string = if data.user_scores && user.is_some() {
23         r#"
24         SELECT "game".score, "user".username
25         FROM "game"
26         JOIN "user" ON "game".user_id = "user".id
27         WHERE dimension = $1
28         AND time_limit = $2
29         AND "user".id = $4
30         ORDER BY score
31         OFFSET ($3 - 1) * 100
32         FETCH NEXT 100 ROWS ONLY
33         "#
34     } else {
35         r#"
36         SELECT "game".score, "user".username
37         FROM "game"
38         JOIN "user" ON "game".user_id = "user".id
39         WHERE dimension = $1
40         AND time_limit = $2
41         ORDER BY score
42         OFFSET ($3 - 1) * 100
43         FETCH NEXT 100 ROWS ONLY
44         "#
45     };
46     let user_id = match user.is_some() {
47         true => user.unwrap().id,
48         false => uuid::Uuid::new_v4(),

```



```

49     };
50     let res: Vec<(String, usize)> = sqlx::query_as::<_, Score>(query_string)
51         .bind(data.dimension as i32)
52         .bind(data.time_limit as i32)
53         .bind(data.page as i32)
54         .bind(user_id)
55         .fetch_all(&mut *state.db.acquire().await?)
56         .await?
57         .iter()
58         .map(|x| (x.username.clone(), x.score.unwrap() as usize))
59         .collect();
60     Ok(Json(res))
61 }

```

#### 0.10.5.5. Server Routing

rust

```

1 [tokio::main]
2 async fn main() {
3     // basic initialization
4     dotenvy::dotenv().ok();
5
6     let database_url = std::env::var("DATABASE_URL").expect("DB_URL must be set");
7
8     let pool = PgPool::connect(&database_url).await.unwrap();
9
10    tracing_subscriber::fmt::init();
11
12    let state = Arc::new(AppState {
13        games: Mutex::new(HashMap::new()),
14        game_manager: GameManager {
15            user_games: Arc::new(Mutex::new(Queue::<(
16                ulid::Ulid,
17                tokio::sync::mpsc::Sender<WebSocket>,
18            )>::new()))),
19            cheater_games: Arc::new(Mutex::new(Queue::<(
20                ulid::Ulid,
21                tokio::sync::mpsc::Sender<WebSocket>,
22            )>::new()))),
23            anon_games: Arc::new(Mutex::new(Queue::<(
24                ulid::Ulid,
25                tokio::sync::mpsc::Sender<WebSocket>,
26            )>::new()))),
27        },
28        db: pool,
29    });
30    let app = Router::new()
31        .route("/get-seed", post(create_seed))
32        .route("/submit-game", post(submit_game))
33        .route("/game", any(ws_upgrader))
34        .route("/get_scores", post(misc::get_scores))
35        .route("/user/signup", post(misc::signup))
36        .route("/user/login", post(misc::login))
37        .layer(middleware::from_fn_with_state(
38            state.clone(),
39            misc::authorization,
40        ))
41        .with_state(state)

```

```

42     .layer(CorsLayer::permissive())
43     .layer(TraceLayer::new_for_http());
44
45     let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
46     axum::serve(listener, app).await.unwrap();
47 }

```

#### 0.10.5.6. Singleplayer Game Management

```

1  /// enum representing all possible moves done by the client
2  #[repr(u8)]
3  #[derive(Serialize, Deserialize, Debug, PartialEq, Eq, Clone, Copy)]
4  pub enum Move {
5      CursorRedUp,
6      CursorRedDown,
7      CursorRedLeft,
8      CursorRedRight,
9      CursorBlueUp,
10     CursorBlueDown,
11     CursorBlueLeft,
12     CursorBlueRight,
13     Submit,
14 }
15 #[derive(Serialize, Deserialize)]
16 pub struct GameForm {
17     dimension: u8,
18     time_limit: u8,
19 }
20
21 #[derive(Debug, Copy, Clone)]
22 pub struct GameState {
23     seed: u32,
24     dimension: u8,
25     time_limit: Duration,
26     start_time: Instant,
27 }
28
29 #[derive(Serialize)]
30 pub struct Seed {
31     id: String,
32     seed: u32,
33 }
34 #[derive(Serialize, Deserialize, Debug)]
35 pub struct GameEnd {
36     id: String,
37     score: u32,
38     //u32 is time difference in ms
39     moves: Vec<(Move, u32)>,
40 }
41 }
42
43 /// creates a new seed using the implemented splitmix and xoshiro256+ algorithms from sillyrng
44 #[axum::debug_handler]
45 pub async fn create_seed(
46     State(state): State<Arc<AppState>>,
47     Json(form): Json<GameForm>,
48 ) -> (StatusCode, Json<Seed>) {

```

```

49     let game_id = ulid::Ulid::new();
50     let seed = rand::random::<u32>();
51     let game_state = GameState {
52         seed,
53         dimension: form.dimension,
54         time_limit: Duration::from_secs(form.time_limit.into()),
55         start_time: Instant::now(),
56     };
57
58     println!(
59         "Creating game {} with dimension {} and time limit {}s",
60         game_id, form.dimension, form.time_limit
61     );
62
63     state.games.lock().await.insert(game_id, game_state);
64
65     let res = Json(Seed {
66         id: game_id.to_string(),
67         seed,
68     });
69
70     (StatusCode::OK, res)
71 }
72
73 #[axum::debug_handler]
74 pub async fn submit_game(
75     State(state): State<Arc<AppState>>,
76     Extension(user): Extension<Option<UserExt>>,
77     Json(game): Json<GameEnd>,
78 ) -> Result<(StatusCode, Json<u32>), AppError> {
79     println!(
80         "Received submission for game {} with {} moves",
81         game.id,
82         game.moves.len()
83     );
84
85     let id = ulid::Ulid::from_string(&game.id).unwrap();
86     let lock = state.games.lock().await;
87     let mut conn = state.db.acquire().await?;
88     let details = lock.get(&id).unwrap();
89
90
91     let elapsed = Instant::now().duration_since(details.start_time);
92     if elapsed > details.time_limit + Duration::from_secs(3) {
93         println!("Game {} exceeded time limit ({}s + 3s)", game.id, details.time_limit.as_secs());
94         return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
95     }
96
97     let time = verify_timings(game.moves.iter().map(|(&_, m)| *m).collect(), state.clone()).await;
98
99     if !time.0 {
100         println!("Rejected game {} due to suspicious timings", game.id);
101         return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
102     }
103
104     let score = match verify_moves(
105         game.moves.iter().map(|(m, _)| *m).collect(),
106         details.dimension,
107         details.seed,

```

```

107     )
108     .await
109     {
110         Ok(s) => s,
111         Err(e) => {
112             println!("{:?}", e);
113             // TODO anomalous game pushing
114             return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
115         }
116     };
117     if score == game.score {
118         if let Some(u) = user {
119             println!(
120                 "Game {} submitted with score {}, user exists : {}",
121                 game.id,
122                 score,
123                 u.clone().username
124             );
125             sqlx::query!("INSERT INTO
126 \"game\" (game_id,score,average_time,dimension,time_limit,user_id) VALUES ($1,$2,$3,$4,$5,
127 $6)",uuid::Uuid::new_v4(),score as i32,time.1, details.dimension as i32,30,u.id).execute(&mut
128 *conn).await?;
129         }
130         Ok((StatusCode::OK, Json(score)))
131     } else {
132         Ok((StatusCode::NOT_ACCEPTABLE, Json(0)))
133     }
134 }
135
136 pub async fn verify_moves(moves: Vec<Move>, size: u8, seed: u32) -> Result<u32, String> {
137     //this is assuming we start at 0,0 and size,size (should be a client side force, now enforced)
138     let mut rng = sillyrng::Xoshiro256plus::new(Some(seed as u64));
139     let mut grid: Vec<bool> = vec![false; (size * size) as usize];
140     let mut blue_coords: (u8, u8) = (0, 0);
141     let mut red_coords: (u8, u8) = (size - 1, size - 1);
142     let mut score = 0;
143     let mut distance = 0;
144     let mut anomalous_distances = 0;
145     let mut optimal_distance = 0;
146     let mut count = 0;
147     while count < size {
148         let x: u8 = (rng.next() * size as f64).floor() as u8;
149         let y: u8 = (rng.next() * size as f64).floor() as u8;
150         if grid[(x * size + y) as usize] == false {
151             grid[(x * size + y) as usize] = true;
152             count += 1;
153         }
154     }
155     for i in moves.iter() {
156         match i {
157             Move::CursorRedUp => {
158                 red_coords.1 = (red_coords.1 as i8 - 1).max(0) as u8;
159                 distance += 1;
160             }
161             Move::CursorRedDown => {
162                 red_coords.1 = (red_coords.1 + 1).min(size - 1);
163                 distance += 1;
164             }
165         }
166     }
167 }

```

```

162 Move::CursorRedLeft => {
163     red_coords.0 = (red_coords.0 as i8 - 1).max(0) as u8;
164     distance += 1;
165 }
166 Move::CursorRedRight => {
167     red_coords.0 = (red_coords.0 + 1).min(size - 1);
168     distance += 1;
169 }
170 Move::CursorBlueUp => {
171     blue_coords.1 = (blue_coords.1 as i8 - 1).max(0) as u8;
172     distance += 1;
173 }
174 Move::CursorBlueDown => {
175     blue_coords.1 = (blue_coords.1 + 1).min(size - 1);
176     distance += 1;
177 }
178 Move::CursorBlueLeft => {
179     blue_coords.0 = (blue_coords.0 as i8 - 1).max(0) as u8;
180     distance += 1;
181 }
182 Move::CursorBlueRight => {
183     blue_coords.0 = (blue_coords.0 + 1).min(size - 1);
184     distance += 1;
185 }
186 Move::Submit => {
187     if distance <= optimal_distance {
188         anomalous_distances += 1;
189     }
190     distance = 0;
191
192     if grid[(red_coords.0 * size + red_coords.1) as usize]
193         && grid[(blue_coords.0 * size + blue_coords.1) as usize]
194         && !(blue_coords == red_coords)
195     {
196         score += 1;
197         let mut count = 0;
198         let r = red_coords.0 * size + red_coords.1;
199         let b = blue_coords.0 * size + blue_coords.1;
200         while count < 2 {
201             let x: u8 = (rng.next() * size as f64).floor() as u8;
202             let y: u8 = (rng.next() * size as f64).floor() as u8;
203             if !grid[(x * size + y) as usize]
204                 && (x * size + y != r || x * size + y != b)
205             {
206                 grid[(x * size + y) as usize] = true;
207                 count += 1;
208             }
209         }
210         grid[r as usize] = false;
211         grid[b as usize] = false;
212         optimal_distance =
213             get_optimal_paths(grid.clone(), red_coords, blue_coords, size)
214                 .await
215                 .iter()
216                 .min()
217                 .unwrap_or(&0)
218                 .to_owned();
219     } else {

```

```

220             score = 0
221         }
222     }
223 }
224 }
225 println!(
226     "Game completed with score {} (anomaly ratio: {:.2})",
227     score,
228     anomalous_distances as f64 / score as f64
229 );
230 Ok(score)
231 }
232
233
234 pub async fn get_optimal_paths(grid: Vec<bool>, r: (u8, u8), b: (u8, u8), size: u8) -> Vec<u32> {
235     let mut paths = Vec::new();
236     for i in 0..grid.len() {
237         for j in 0..grid.len() {
238             if grid[i] && grid[j] && i != j {
239                 let r_cell = ((i / size as usize) as u8, (i % size as usize) as u8);
240                 let b_cell = ((j / size as usize) as u8, (j % size as usize) as u8);
241                 let r_dist = (r.0.abs_diff(r_cell.0) + r.1.abs_diff(r_cell.1)) as u32;
242                 let b_dist = (b.0.abs_diff(b_cell.0) + b.1.abs_diff(b_cell.1)) as u32;
243                 paths.push(r_dist + b_dist);
244             }
245         }
246     }
247     paths
248 }
249

```

#### 0.10.5.7. Backend Error Handling

```

1 use axum::http::{Response, StatusCode};
2 use axum::response::IntoResponse;
3 use bcrypt::BcryptError;
4 use thiserror::Error;
5 #[derive(Error, Debug)]
6 pub enum AppError {
7     #[error("statuscode")]
8     Status(StatusCode),
9     #[error("bcrypt error")]
10    Hash(#[from] BcryptError),
11    #[error("Ulid Encode Error")]
12    UEncode(#[from] ulid::EncodeError),
13    #[error("Ulid Decode Error")]
14    UDecode(#[from] ulid::DecodeError),
15    #[error("failed to deserialize")]
16    Json(#[from] serde_json::Error),
17    #[error("pool failed to execute")]
18    Pool(#[from] sqlx::Error),
19 }
20
21 impl IntoResponse for AppError {
22     fn into_response(self) -> axum::response::Response {
23         let (body, code) = match self {
24             AppError::Status(e) => ("", e),

```

```

25         _ => ("Unknown", StatusCode::INTERNAL_SERVER_ERROR),
26     };
27     Response::builder().status(code).body(body.into()).unwrap()
28 }
29 }
30
31
32

```

#### 0.10.5.8. Database Models

rust

```

1  #[derive(serde::Serialize, serde::Deserialize, Clone)]
2  pub struct User {
3      pub id: uuid::Uuid,
4      pub password: String,
5      pub username: String,
6      pub admin: Option<bool>,
7      pub cheater: Option<bool>,
8  }
9
10 #[derive(serde::Serialize, serde::Deserialize, Clone, Debug)]
11 pub struct UserExt {
12     pub id: uuid::Uuid,
13     pub username: String,
14     pub admin: Option<bool>,
15     pub cheater: Option<bool>,
16 }
17
18 #[derive(serde::Serialize, serde::Deserialize)]
19 pub struct UserStatistics {
20     pub stat_id: uuid::Uuid,
21     pub highest_score: Option<i16>,
22     pub victories: Option<i16>,
23     pub games_played: Option<i16>,
24     pub elo: Option<i16>,
25     pub user_id: uuid::Uuid,
26 }
27
28 #[derive(serde::Serialize, serde::Deserialize)]
29 pub struct Game {
30     pub game_id: uuid::Uuid,
31     pub score: Option<i16>,
32     pub average_time: Option<f32>,
33     pub dimension: Option<i16>,
34     pub time_limit: Option<i16>,
35     pub user_id: uuid::Uuid,
36 }
37
38 #[derive(serde::Serialize, serde::Deserialize)]
39 pub struct Statistics {
40     pub stat_id: uuid::Uuid,
41     pub total_timings: Option<f32>,
42     pub total_score: Option<i64>,
43     pub games_played: Option<i64>,
44 }
45
46 #[derive(serde::Serialize, serde::Deserialize)]

```

```

47 pub struct AnomalousGames {
48     pub agame_id: uuid::Uuid,
49     pub moves: serde_json::Value,
50     pub user_id: uuid::Uuid,
51 }
52
53 #[derive(serde::Serialize, serde::Deserialize)]
54 pub struct Session {
55     pub ssid: uuid::Uuid,
56     pub expiry_date: chrono::NaiveDate,
57     pub user_id: uuid::Uuid,
58 }
59
60
61

```

#### 0.10.5.9. Multiplayer game management

```

1 pub async fn ws_upgrader(
2     ws: WebSocketUpgrade,
3     State(state): State<Arc<AppState>>,
4     Extension(user): Extension<Option<UserExt>>,
5 ) -> Response {
6     // required due to state not implementing copy
7     let cloned_state = state.clone();
8     ws.on_upgrade(move |socket| ws_handler(socket, cloned_state, user))
9 }
10
11 pub async fn ws_handler(ws: WebSocket, state: Arc<AppState>, user: Option<UserExt>) {
12     state.game_manager.clone().assign_game(ws, user).await
13 }
14
15 use axum::extract::ws::{Message, WebSocket};
16 use futures::stream::SplitSink;
17 use futures::{SinkExt, StreamExt, TryFutureExt};
18 use sillyrng::{Gen, Xoshiro256plus};
19 use std::{collections::HashMap, sync::Arc};
20 use tokio::select;
21 use tokio::sync::{mpsc, Mutex};
22 use tokio::time::{interval, Duration};
23 use ulid::Ulid;
24
25 use crate::misc::Queue;
26 use crate::models::UserExt;
27 use crate::Move;
28
29 #[derive(Clone)]
30 pub struct GameManager {
31     pub user_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>>>>>,
32     pub anon_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>>>>>,
33     pub cheater_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>>>>>,
34 }
35
36 pub struct Game {
37     players: HashMap<Ulid, Player>,
38     inactive_players: HashMap<Ulid, Player>,
39     seed: u32,

```



```

40     quota: u32,
41 }
42
43 #[derive(Clone, Debug)]
44 pub struct Player {
45     grid: [bool; 16],
46     b_coords: (u8, u8),
47     r_coords: (u8, u8),
48     current_score: u8,
49     rng: sillyrng::Xoshiro256plus,
50 }
51
52 #[derive(serde::Serialize, serde::Deserialize, Debug)]
53 pub struct MMove {
54     player_id: Ulid,
55     action: Move,
56 }
57
58 #[derive(serde::Serialize, serde::Deserialize)]
59 #[serde(tag = "type", content = "data")]
60 pub enum WsMessage {
61     Move(MMove),
62     Quota { quota: u32, players_left: u32 },
63     ID(Ulid),
64     Start(u32),
65     Out(u32),
66     Win,
67     Ping,
68 }
69
70 impl GameManager {
71     pub async fn assign_game(&self, ws: WebSocket, user: Option<UserExt>) {
72         println!("Attempting to assign player to a game");
73         let games = match user {
74             Some(u) => {
75                 //innocent until proven guilty, very demure, very fashionable
76                 if u.cheater.unwrap_or(false) {
77                     self.cheater_games.clone()
78                 } else {
79                     self.user_games.clone()
80                 }
81             }
82             None => self.anon_games.clone(),
83         };
84         let mut attempts = games.lock().await.size;
85         let mut ws = ws;
86         while attempts > 0 {
87             let mut lock = games.lock().await;
88             if let Some(game) = lock.dequeue() {
89                 match game.1.send(ws).await {
90                     Ok(()) => {
91                         lock.enqueue(game.clone());
92                         return;
93                     }
94                     Err(mpsc::error::SendError(rws)) => {
95                         ws = rws;
96                         attempts -= 1;
97                     }
98                 }
99             }
100         }

```

```

98         }
99     } else {
100         break;
101     }
102 }
103
104 let (tx, rx) = mpsc::channel(40);
105 let game_id = Uuid::new();
106 tokio::spawn(game_handler(game_id.clone(), rx));
107
108 match tx.send(ws).await {
109     Ok(_) => {
110         println!("Created new game with ID: {}", game_id);
111         games.lock().await.enqueue((game_id, tx));
112     }
113     Err(e) => {
114         println!("failed to send to game error: {}", e);
115     }
116 };
117 }
118 }
119
120
121 async fn game_handler(id: Uuid, mut rx: mpsc::Receiver<WebSocket>) {
122     println!("Game {} initialized, waiting for players", id);
123     let mut state = Game {
124         players: HashMap::new(),
125         inactive_players: HashMap::new(),
126         seed: rand::random:::<u32>(),
127         quota: 0,
128     };
129     let mut senders: HashMap<Uuid, SplitSink<WebSocket, Message>> = HashMap::new();
130     let mut receivers = vec![];
131     while state.players.len() <= 5 {
132         match rx.recv().await {
133             Some(mut p) => {
134                 let meow_id = Uuid::new();
135                 println!("Player {} joined game {}", meow_id, id);
136                 p.send(axum::extract::ws::Message::Text(
137                     serde_json::to_string(&WsMessage::ID(meow_id)).unwrap(),
138                 ))
139                 .await
140                 .unwrap();
141                 state.players.insert(
142                     meow_id.clone(),
143                     Player {
144                         grid: [false; 16],
145                         b_coords: (0, 0),
146                         r_coords: (3, 3),
147                         current_score: (0),
148                         rng: Xoshiro256plus::new(Some(3)),
149                     },
150                 );
151                 let (sender, receiver) = p.split();
152                 senders.insert(meow_id, sender);
153                 receivers.push(receiver);
154             }
155             None => {}

```

```

156     }
157 }
158
159 println!("Game {} starting with {} players", id, state.players.len());
160 for i in state.players.iter_mut() {
161     i.l.rng = Xoshiro256plus::new(Some(state.seed.clone() as u64));
162     let mut count = 0;
163     while count < 4 {
164         let x: u8 = (i.l.rng.next() * 4 as f64).floor() as u8;
165         let y: u8 = (i.l.rng.next() * 4 as f64).floor() as u8;
166         if i.l.grid[(x * 4 + y) as usize] == false {
167             i.l.grid[(x * 4 + y) as usize] = true;
168             count += 1;
169         }
170     }
171 }
172 for (_p, i) in senders.iter_mut() {
173     i.send(axum::extract::ws::Message::Text(
174         serde_json::to_string(&WsMessage::Start(state.seed)).unwrap(),
175     ))
176     .await
177     .unwrap();
178 }
179
180 println!("Game {} is now running", id);
181 let mut interval = interval(Duration::from_secs(5));
182
183 loop {
184     let websocket_futures = futures::future::select_all(
185         receivers
186             .iter_mut()
187             .enumerate()
188             .map(|(i, ws)| Box::pin(async move { (i, ws.next().await) })),
189     );
190
191     select! {
192         (result, _, _) = websocket_futures => {
193             let (idx, msg_result) = result;
194             match msg_result {
195                 Some(Ok(axum::extract::ws::Message::Text(text))) => {
196                     match serde_json::from_str::(<WsMessage>(&text)) {
197                         Ok(WsMessage::Move(mrrp)) => {
198                             println!("Received move from socket {}: {:?}", idx, mrrp);
199                             let player = state.players.get_mut(&mrrp.player_id);
200                             match player {
201                                 Some(p) => {
202                                     match mrrp.action {
203                                         Move::CursorRedUp => p.r_coords.1 = (p.r_coords.1 as
i8 - 1).max(0) as u8,
204                                         // 3 should be constant, but multilpayer is only 4x4
205                                         Move::CursorRedDown => p.r_coords.1 = (p.r_coords.1 +
1).min(3),
206                                         Move::CursorRedLeft => p.r_coords.0 = (p.r_coords.0 as
i8 - 1).max(0) as u8,
207                                         Move::CursorRedRight => p.r_coords.0 = (p.r_coords.0 +
1).min(3),
208                                         Move::CursorBlueUp => p.b_coords.1 = (p.b_coords.1 as
i8 - 1).max(0) as u8,

```

```

209         Move::CursorBlueDown => p.b_coords.1 = (p.b_coords.1 +
1).min(3),
210         Move::CursorBlueLeft => p.b_coords.0 = (p.b_coords.0
as i8 - 1).max(0) as u8,
211         Move::CursorBlueRight => p.b_coords.0 = (p.b_coords.0
+ 1).min(3),
212         Move::Submit => {
213             dbg!(p.grid);
214             dbg!(p.r_coords, p.b_coords);
215             if p.grid[(p.r_coords.0 * 4 + p.r_coords.1) as
usize] && p.grid[(p.b_coords.0 * 4 + p.b_coords.1) as usize] && !(p.b_coords == p.r_coords) {
216                 println!("successfull submission");
217                 p.current_score += 1;
218                 let mut count = 0;
219                 let r = p.r_coords.0 * 4 + p.r_coords.1;
220                 let b = p.b_coords.0 * 4 + p.b_coords.1;
221                 while count < 2 {
222                     let x: u8 = (p.rng.next() * 4 as
f64).floor() as u8;
223                     let y: u8 = (p.rng.next() * 4 as
f64).floor() as u8;
224                     if !p.grid[(x * 4 + y) as usize]
&& (x * 4 + y != r || x * 4 + y != b)
225                     {
226                         p.grid[(x * 4 + y) as usize] = true;
227                         count += 1;
228                     }
229                 }
230                 p.grid[r as usize] = false;
231                 p.grid[b as usize] = false;
232             } else {
233                 p.current_score = 0;
234             }
235         },
236     },
237 },
238 }
239 None => {
240     println!("Recieved message from invalid player");
241 }
242 }
243 }
244 }
245 Ok(_) => {
246     println!("Received non-move message from socket {}", idx);
247 }
248 Err(e) => {
249     println!("Error parsing message from socket {}: {}", idx, e);
250 }
251 }
252 }
253 None => {
254     println!("Socket {} closed for game {}", idx, id);
255     let _ = receivers.remove(idx);
256 }
257 _ => continue,
258 }
259 }
260 _ = interval.tick() => {

```

```

261         println!("New quota for game {}", id);
262         let mut culled_players = vec![];
263         let player_count = state.players.len();
264         for (i, p) in state.players.iter_mut() {
265             dbg!(p.current_score);
266             if (p.current_score as u32) < state.quota {
267                 let position = (player_count - culled_players.len()) as u32;
268
269                 if let Err(e) = senders.get_mut(&i.clone()).unwrap()
270                     .send(axum::extract::ws::Message::Text(
271                         serde_json::to_string(&WsMessage::Out(position))
272                         .expect("Failed to serialize Out message")
273                     )).await
274                 {
275                     println!("Failed to send message to player, removing {}: {}", i, e);
276
277                     continue;
278                 }
279                 culled_players.push(i.clone());
280                 state.inactive_players.insert(i.clone(), p.clone());
281                 senders.remove(&i);
282             }
283             p.current_score = 0;
284         }
285         for i in culled_players {
286             state.players.remove(&i);
287         }
288         if state.players.len() <= 1 {
289             println!("Game {} ended - {} player(s) remaining", id, state.players.len());
290             for (i, _) in state.players.iter() {
291                 // add to database when that gets done
292             }
293             senders.get_mut(&i.clone()).unwrap().send(axum::extract::ws::Message::Text(serde_json::to_string(&WsMessage::Out(0))
294                 .expect("Failed to serialize Out message"))).await
295             break;
296         }
297         state.quota += 1;
298         for (_i, sender) in &mut senders {
299             sender.send(axum::extract::ws::Message::Text(
300                 serde_json::to_string(&WsMessage::Quota {
301                     quota: state.quota,
302                     players_left: state.players.len() as u32,
303                 })
304                 .unwrap(),
305             )).await
306             .unwrap();
307         }
308     }
309 }
310 }
311 }
312 println!("Game {} has ended", id);
313 }
314
315
316
317

```

```

1 pub trait Gen {
2     type NumberType;
3     fn new(seed: Option<u64>) -> Self;
4     fn next(&mut self) -> Self::NumberType;
5     fn sigmoid(x: f64) -> f64 {
6         1.0 / (1.0 + (-x).exp())
7     }
8 }
9
10 pub struct SplitMix {
11     seed: u64,
12 }
13
14 impl Gen for SplitMix {
15     type NumberType = u64;
16     fn new(seed: Option<u64>) -> Self {
17         SplitMix {
18             seed: seed.unwrap(),
19         }
20     }
21     /// based on https://xoshiro.di.unimi.it/splitmix64.c and rand_xoshiro
22     fn next(&mut self) -> u64 {
23         self.seed = self.seed.wrapping_add(0x9e3779b97f4a7c15);
24         let mut z: u64 = self.seed;
25         z = (z ^ (z >> 30)).wrapping_mul(0xbf58476d1ce4e5b9);
26         z = (z ^ (z >> 27)).wrapping_mul(0x94d049bb133111eb);
27         z ^ (z >> 31)
28     }
29 }
30
31 #[derive(Debug, Clone)]
32 pub struct Xoshiro256plus {
33     seed: [u64; 4],
34 }
35
36 impl Gen for Xoshiro256plus {
37     type NumberType = f64;
38
39     fn new(seed: Option<u64>) -> Self {
40         let mut rng = SplitMix::new(seed);
41         Xoshiro256plus {
42             seed: [rng.next(), rng.next(), rng.next(), rng.next()],
43         }
44     }
45
46     fn next(&mut self) -> Self::NumberType {
47         let result = self.seed[0].wrapping_add(self.seed[3]);
48         let t = self.seed[1] << 17;
49
50         self.seed[2] ^= self.seed[0];
51         self.seed[3] ^= self.seed[1];
52         self.seed[1] ^= self.seed[2];
53         self.seed[0] ^= self.seed[3];
54
55         self.seed[2] ^= t;
56         self.seed[3] = Xoshiro256plus::rol64(self.seed[3], 45);

```

```

57         (result >> 11) as f64 * (1.0 / (1u64 << 53) as f64)
58     }
59 }
60
61 impl Xoshiro256plus {
62     pub fn rol64(x: u64, k: i32) -> u64 {
63         (x << k) | (x >> (64 - k))
64     }
65     pub fn get_seed(&self) -> String {
66         format!("{:?}", self.seed)
67     }
68 }
69
70
71

```

```

1 import * as wasm from "./xoshiro_wasm_bg.wasm";
2 export * from "./xoshiro_wasm_bg.js";
3 import { __wbg_set_wasm } from "./xoshiro_wasm_bg.js";
4 __wbg_set_wasm(wasm);
5 wasm.__wbindgen_start();
6
7 /* tslint:disable */
8 /* eslint-disable */
9 export const memory: WebAssembly.Memory;
10 export const sigmoid: (a: number) => number;
11 export const __wbg_splitmix_free: (a: number, b: number) => void;
12 export const splitmix_new: (a: number, b: bigint) => number;
13 export const splitmix_next: (a: number) => bigint;
14 export const __wbg_xoshiro256plus_free: (a: number, b: number) => void;
15 export const xoshiro256plus_new: (a: number, b: bigint) => number;
16 export const xoshiro256plus_next: (a: number) => number;
17 export const xoshiro256plus_get_seed: (a: number) => [number, number];
18 export const __wbindgen_export_0: WebAssembly.Table;
19 export const __wbindgen_free: (a: number, b: number, c: number) => void;
20 export const __wbindgen_start: () => void;
21
22 /* tslint:disable */
23 /* eslint-disable */
24 export function sigmoid(x: number): number;
25 export class SplitMix {
26     free(): void;
27     constructor(seed?: bigint);
28     next(): bigint;
29 }
30 export class Xoshiro256plus {
31     free(): void;
32     constructor(seed?: bigint);
33     next(): number;
34     get_seed(): string;
35 }
36

```

#### 0.10.5.11. Settings Component

```
1 <script lang="ts">
```

```

2  import { getContext } from 'svelte';
3  let meow = 0;
4  export let showModal: boolean;
5  export let closeModal: any;
6  let dialog: any;
7  let idx: any;
8  let state: any = getContext('state');
9  let keycodes: any;
10
11  $: keycodes = $state.keycodes;
12  const reset = () => {
13    $state = JSON.parse(
14      JSON.stringify({
15        gameMode: 'timer',
16        timeLimit: 30,
17        keycodes: {
18          wU: 'w',
19          wD: 's',
20          wL: 'a',
21          wR: 'd',
22          aU: 'ArrowUp',
23          aD: 'ArrowDown',
24          aL: 'ArrowLeft',
25          aR: 'ArrowRight',
26          submit: ' ',
27          reset: 'r'
28        },
29        size: 4,
30        das: 133,
31        dasDelay: 150
32      })
33    );
34    meow += 1;
35  };
36  const getChar = (i: any) => {
37    let char: any;
38    switch (i) {
39      case '0':
40        char = keycodes.wU;
41        break;
42      case '1':
43        char = keycodes.aU;
44        break;
45      case '00':
46        char = keycodes.wL;
47        break;
48      case '01':
49        char = keycodes.wD;
50        break;
51      case '02':
52        char = keycodes.wR;
53        break;
54      case '10':
55        char = keycodes.aL;
56        break;
57      case '11':
58        char = keycodes.aD;
59        break;

```



```

60     case '12':
61         char = keycodes.aR;
62         break;
63     case '20':
64         char = keycodes.submit;
65         break;
66     case '21':
67         char = keycodes.reset;
68         break;
69 }
70 switch (char) {
71     case 'ArrowUp':
72         char = '↑';
73         break;
74     case 'ArrowDown':
75         char = '↓';
76         break;
77     case 'ArrowLeft':
78         char = '←';
79         break;
80     case 'ArrowRight':
81         char = '→';
82         break;
83 }
84 return char;
85 };
86
87 const keyClick = (i: any) => {
88     idx = i;
89     setTimeout(() => {
90         window.addEventListener('keydown', setChar, { once: true });
91     }, 0);
92 };
93 const setChar = (e: any) => {
94     switch (idx) {
95         case '0':
96             $state.keycodes.wU = e.key;
97             break;
98         case '1':
99             $state.keycodes.aU = e.key;
100             break;
101         case '00':
102             $state.keycodes.wL = e.key;
103             break;
104         case '01':
105             $state.keycodes.wD = e.key;
106             break;
107         case '02':
108             $state.keycodes.wR = e.key;
109             break;
110         case '10':
111             $state.keycodes.aL = e.key;
112             break;
113         case '11':
114             $state.keycodes.aD = e.key;
115             break;
116         case '12':
117             $state.keycodes.aR = e.key;

```

```

118     break;
119     case '20':
120         $state.keycodes.submit = e.key;
121         break;
122     case '21':
123         $state.keycodes.reset = e.key;
124         break;
125 }
126 let doc: any = document.getElementById(idx);
127 let char = e.key;
128 switch (char) {
129     case 'ArrowUp':
130         char = '↑';
131         break;
132     case 'ArrowDown':
133         char = '↓';
134         break;
135     case 'ArrowLeft':
136         char = '←';
137         break;
138     case 'ArrowRight':
139         char = '→';
140         break;
141 }
142 doc.textContent = char;
143 idx = 69420;
144 };
145
146 $: if (dialog && showModal) dialog.showModal();
147 </script>
148
149 <dialog
150   bind:this={dialog}
151   on:close={closeModal}
152   class="h-screen w-screen bg-crust/0 flex items-center justify-center {showModal ? '' :
153     'hidden'}"
154   >
155     {#key meow}
156     <div class="flex flex-col bg-surface0 w-fit h-fit rounded-md">
157       <div class="text-text text-3xl m-4 mb-0">settings</div>
158       <div class="text-xl text-text mb-0 m-4">movement:</div>
159       <div class="flex flex-row m-4">
160         {#each Array(2) as _, x}
161         <div class="flex flex-col items-center mx-4">
162           <!-- svelte-ignore ally-click-events-have-key-events -->
163           <!-- svelte-ignore ally-no-static-element-interactions -->
164           <div
165             id={x.toString()}
166             class=" rounded-md w-16 h-16 hover:scale-105 transition flex items-center justify-
167             center text-crust text-xl bold focus:bg-surface0 m-1 select-none cursor-pointer {idx ==
168             x.toString()
169             ? 'bg-green'
170             : 'bg-text'}"
171             on:click={() => keyClick(x.toString())}
172           >
173             {getChar(x.toString())}
174           </div>
175         <div class="flex flex-row">

```

```

174         {#each Array(3) as _, y}
175             <!-- svelte-ignore ally-click-events-have-key-events -->
176             <!-- svelte-ignore ally-no-static-element-interactions -->
177             <div
178                 id={x.toString() + y.toString()}
179                 class=" rounded-md w-16 h-16 hover:scale-105 transition flex items-center
justify-center text-crust text-xl bold focus:bg-surface0 m-1 select-none cursor-pointer {idx ==
180                     x.toString() + y.toString()
181                     ? 'bg-green'
182                     : 'bg-text'}"
183                 on:click={() => keyClick(x.toString() + y.toString())}
184                 >
185                     {getChar(x.toString() + y.toString())}
186             </div>
187         {/each}
188     </div>
189 </div>
190 {/each}
191 </div>
192 <div class="text-xl text-text mb-0 m-4">place:</div>
193 <!-- svelte-ignore ally-click-events-have-key-events -->
194 <!-- svelte-ignore ally-no-static-element-interactions -->
195 <div
196     id={'20'}
197     class=" rounded-md max-w-full h-16 hover:scale-105 transition flex items-center justify-
center text-crust text-xl bold focus:bg-surface0 mx-8 my-4 select-none cursor-pointer {idx ==
198     '20'
199     ? 'bg-green'
200     : 'bg-text'}"
201     on:click={() => keyClick('20')}
202     >
203         {getChar('20')}
204     </div>
205 <div class="text-xl text-text mb-0 m-4">reset:</div>
206 <!-- svelte-ignore ally-click-events-have-key-events -->
207 <!-- svelte-ignore ally-no-static-element-interactions -->
208 <div
209     id={'21'}
210     class="rounded-md max-w-full h-16 hover:scale-105 transition flex items-center justify-
center text-crust text-xl bold focus:bg-surface0 mx-8 my-4 select-none cursor-pointer {idx ==
211     '21'
212     ? 'bg-green'
213     : 'bg-text'}"
214     on:click={() => keyClick('21')}
215     >
216         {getChar('21')}
217     </div>
218 <div class="text-xl text-text mb-0 m-4">auto repeat rate:</div>
219 <div class="flex flex-row text-text text-xl mx-8">
220     <div class="w-8">
221         {$state.das}
222     </div>
223     <input class="mx-4 w-64" type="range" min="0" max="1000" step="1"
bind:value={$state.das} />
224 </div>
225 <div class="text-xl text-text mb-0 m-4">delayed auto shift:</div>
226 <div class=" flex flex-row text-text text-xl mx-8">
227     <div class="w-8">

```

```

228     {$state.dasDelay}
229   </div>
230   <input
231     class="mx-4 w-64"
232     type="range"
233     min="0"
234     max="1000"
235     step="1"
236     bind:value={$state.dasDelay}
237   />
238 </div>
239 <div class="flex flex-row self-center m-4">
240   <button class="text-crust bg-red rounded-md w-16 h-8 mx-2 hover:scale-105"
on:click={reset}
241     >reset</button>
242   >
243   <button
244     class="text-crust bg-blue rounded-md w-16 h-8 mx-2 hover:scale-105"
245     on:click={() => dialog.close()}>exit</button>
246   >
247 </div>
248 </div>
249 {/key}
250 </dialog>
251

```

#### 0.10.5.12. Layout and Styling

svelte

```

1 <script lang="ts">
2   import './app.css';
3   import studio from '$lib/assets/studio.png';
4   import Modal from '$lib/settings.svelte';
5   import Trophy from 'svelte-material-icons/Trophy.svelte';
6   import AccountCircle from 'svelte-material-icons/AccountCircle.svelte';
7   import Settings from 'svelte-material-icons/Cog.svelte';
8   import { onMount, setContext } from 'svelte';
9   import { browser } from '$app/environment';
10  import { writable } from 'svelte/store';
11  import Information from 'svelte-material-icons/Information.svelte';
12  import { redirect } from '@sveltejs/kit';
13  import { goto } from '$app/navigation';
14  const FLAVOUR = 'mocha';
15  let showModal = false;
16  let showWelcome = false;
17  let selectedElement: { focus: () => void; };
18  //TODO custom bg
19  type gameState = {
20    gameMode: string;
21    timeLimit: number;
22    keycodes: object;
23    size: number;
24  };
25  const defaults = JSON.stringify({
26    gameMode: 'timer',
27    timeLimit: 30,
28    keycodes: {
29      wU: 'w',

```

```

30     wD: 's',
31     wL: 'a',
32     wR: 'd',
33     aU: 'ArrowUp',
34     aD: 'ArrowDown',
35     aL: 'ArrowLeft',
36     aR: 'ArrowRight',
37     submit: ' ',
38     reset: 'r'
39   },
40   size: 4,
41   das: 133,
42   dasDelay: 150
43 });
44 const getState = (): gameState => {
45   if (browser) {
46     return JSON.parse(localStorage.getItem('state') || defaults);
47   } else {
48     return JSON.parse(defaults);
49   }
50 };
51 const state = writable<gameState>(getState());
52
53 if (browser) {
54   state.subscribe(($state) => {
55     localStorage.setItem('state', JSON.stringify($state));
56   });
57 }
58
59 setContext('state', state);
60
61 onMount(() => {
62   if (browser) {
63     const hasSeenWelcome = document.cookie.includes('seenWelcome=true');
64     if (!hasSeenWelcome) {
65       showWelcome = true;
66     }
67   }
68 });
69
70 const closeWelcome = (permanent: boolean) =>{
71   showWelcome = false;
72   if (permanent) {
73     document.cookie = 'seenWelcome=true; max-age=31536000; path=/';
74   }
75 }
76
77 const openModal = (e: any) => {
78   selectedElement = e.currentTarget;
79   showModal = true;
80 }
81
82 const closeModal = () => {
83   showModal = false;
84   if (selectedElement) {
85     selectedElement.focus();
86   }
87 }

```

```

88
89
90 </script>
91
92
93 <main class={FLAVOUR}>
94   <div class="flex flex-col justify-between h-full max-h-screen min-w-screen font-mono">
95     <div class="flex flex-row bg-base justify-between h-fit w-full items-center">
96       <a class="flex flex-row text-4xl text-rosewater p-2" href="/">
97         <x class="text-blue">Double</x> <x class="text-mauve font-bold">TAPP</x>
98       </a>
99       <div class="flex flex-row">
100         <button on:click={() => showWelcome = true}>
101           <Information color="#cdd6f4" class="h-12 w-12 p-2" />
102         </button>
103         <button on:click={openModal}>
104           <Settings color="#cdd6f4" class="h-12 w-12 p-2" />
105         </button>
106         <button on:click={() => goto('/leaderboards')}>
107           <Trophy color="#cdd6f4" class="h-12 w-12 p-2" />
108         </button>
109         <button on:click={() => goto('/signup')}>
110           <AccountCircle color="#cdd6f4" class="h-12 w-12 p-2" />
111         </button>
112       </div>
113     </div>
114     <div class="bg-base h-screen">
115       <slot></slot>
116     </div>
117     <div class="flex flex-row bg-base justify-between h-24 w-full items-center">
118       <div class="flex flex-row items-center opacity-50">
119         <a href="https://studiosquared.co.uk">
120           <img class="m-4 h-10" src={studio} alt="[S]^2" />
121         </a>
122       </div>
123     </div>
124   </div>
125
126   {#if showWelcome}
127     <div class="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center">
128       <div class="bg-base p-6 rounded-lg max-w-md">
129         <h2 class="text-2xl text-rosewater mb-4">Welcome to DoubleTAPP</h2>
130         <p class="text-text mb-4">
131           In DoubleTAPP, your aim is to move both your cursors onto different active tiles to
132           score points. (WASD and arrow keys as default controls)
133         </p>
134         <p class="text-text mb-4">
135           You get a point for each correct move, and lose all your points if you place your
136           cursors incorrectly, good luck!
137         </p>
138         <p class="text-text mb-4">
139           you can customize your controls and other settings in the settings menu.
140         </p>
141         <button
142           class="bg-blue text-base px-4 py-2 rounded"
143           on:click={() => closeWelcome(true)}
144         >
145           Got it!

```

```

144     </button>
145   </div>
146 </div>
147   {/if}
148
149   <Modal bind:showModal />
150 </main>
151
152

```

## 0.11. Testing

Test Description	Status	Proof
Test user registration with valid credentials	Pass	
Test user registration with existing username	Pass	
Test user login with valid credentials	Pass	
Test user login with invalid credentials	Pass	
Test session persistence across page reloads	Pass	
Test session expiry after timeout	Pass	
Test grid initialization with correct size (4x4)	Pass	
Test grid initialization with correct size (5x5)		
Test grid initialization with correct size (6x6)		
Test initial cursor positions (blue at 0,0 and red at size-1,size-1)		
Test initial grid has exactly 'size' active tiles		
Test blue cursor movement in all directions with keyboard		
Test red cursor movement in all directions with keyboard		
Test cursor movement boundary limits (cannot move outside grid)		
Test DAS (Delayed Auto Shift) functionality for cursor movement		
Test valid submission when both cursors are on active tiles		
Test invalid submission when cursors are on the same tile		
Test invalid submission when one cursor is not on an active tile		

Test score increment on valid submission		
Test score reset on invalid submission		
Test visual feedback (green) for correct submissions		
Test visual feedback (red) for incorrect submissions		
Test new active tiles appear after valid submission		
Test deactivation of submitted tiles after valid submission		
Test timer countdown functionality		
Test game end when timer reaches zero		
Test game statistics display after game end		
Test leaderboard display with correct pagination		
Test leaderboard filtering by grid size		
Test leaderboard filtering by time limit		
Test leaderboard filtering for personal bests		
Test multiplayer game joining functionality		
Test multiplayer game quota system		
Test multiplayer game player elimination		
Test multiplayer game final rankings		
Test WebSocket connection establishment		
Test WebSocket message handling for different action types		
Test WebSocket reconnection on connection loss		
Test server-side move verification with valid moves		
Test server-side move verification with invalid moves		
Test server-side timing verification for normal play		
Test server-side timing verification for suspicious patterns		



Test server-side path optimization detection		
Test PRNG (Xoshiro256+) deterministic output with same seed		
Test game state persistence in database		
Test user statistics update after game completion		
Test keybind customization persistence		
Test settings reset to defaults		
Test game performance with rapid inputs		
Test game performance with simultaneous inputs		

## 0.12. Evaluation

### 0.12.1. Overall Effectiveness

### 0.12.2. Evaluation Against Objectives

### 0.12.3. User Feedback

### 0.12.4. Response to Feedback

### 0.12.5. Future Improvements

## 1. Bibliography

- [1] Hard Drop - Tetris Wiki, "Delayed Auto Shift - Understanding Modern Tetris Controls." Accessed: Dec. 17, 2023. [Online]. Available: <https://harddrop.com/wiki/DAS>
- [2] MDN Web Docs, "HTTP - Overview of the Hypertext Transfer Protocol." Accessed: Dec. 20, 2023. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
- [3] IETF - Network Working Group, "User Datagram Protocol (UDP) - Internet Protocol Suite." Accessed: Dec. 20, 2023. [Online]. Available: <https://www.ietf.org/rfc/rfc768.txt>
- [4] Mozilla Developer Network, "An Introduction to WebSockets." Accessed: Dec. 10, 2023. [Online]. Available: [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)
- [5] David Blackman and Sebastiano Vigna, "xoshiro / xoroshiro generators and the PRNG shootout." Accessed: Dec. 15, 2023. [Online]. Available: <https://prng.di.unimi.it/>
- [6] Svelte Team, "SvelteKit Documentation." Accessed: Dec. 19, 2023. [Online]. Available: <https://kit.svelte.dev/docs/introduction>
- [7] Rust Language Team, "Asynchronous Programming in Rust." Accessed: Dec. 02, 2023. [Online]. Available: <https://rust-lang.github.io/async-book/>
- [8] Tailwind Labs, "Tailwind CSS Documentation." Accessed: Dec. 05, 2023. [Online]. Available: <https://tailwindcss.com/docs>
- [9] Catppuccin Team, "Catppuccin Color Scheme." Accessed: Dec. 08, 2023. [Online]. Available: <https://github.com/catppuccin/catppuccin>
- [10] Towards Data Science, "Understanding the Sigmoid Function in Mathematics." Accessed: Dec. 14, 2023. [Online]. Available: <https://towardsdatascience.com/understanding-the-sigmoid-function-385f5c01ae3e>
- [11] Joshua Glazer and Sanjay Madhav, *Multiplayer Game Programming*. Accessed: Dec. 12, 2023. [Online]. Available: <https://www.informit.com/store/multiplayer-game-programming-architecting-networked-9780134034300>