COMPUTER SCIENCE NEA

Multiplayer Grid Based Dexterity Training Game

DoubleTapp

Georgiy Tnimov

Contents

1. Analysis 4
   1.1. Abstract 4
   1.2. Problem Definition 4
   1.3. Client 4
      1.3.1. Client Synopsys (conclusion) 4
      1.3.2. Interview Notes 4
   1.4. Success Criteria 5
   1.5. Research 5
      1.5.1. Similar Solutions 5
         1.5.1.1. Tetris 5
         1.5.1.2. Tapp 6
         1.5.1.3. Other Dexterity Training Applications 7
      1.5.2. Multiplayer 7
      1.5.3. PRNG's (Pseudorandom Number Generators) 8
      1.5.4. Statistics(anti-cheat) 11
         1.5.4.1. Player timings [1] 11
         1.5.4.2. Path optimality 11
         1.5.4.3. Rust Memory Management [2] 11
         1.5.4.4. Postgres Triggers [3] 12
   1.6. Prototyping 12
   1.7. Critical Path 14
   1.8. Objectives 14
      1.8.1. User Interface 14
2. Documented Design 15
      2.0.1. Libraries Used 15
         2.0.1.1. Frontend Libraries 15
         2.0.1.2. Backend Libraries 16
      2.0.2. Iterative Design 18
      2.0.3. Algorithms 19
         2.0.3.1. Xoshiro256+ [4] 19
         2.0.3.2. Sigmoid Function 20
         2.0.3.3. Manhattan Distance 20
         2.0.3.4. MergeSort [5] 21
         2.0.3.5. Standard deviation [1] 21
         2.0.3.6. Delayed Auto Shift [6] 22
         2.0.3.7. Game Verification 22
      2.0.4. API routes 24
      2.0.5. Database Design and Queries 26
         2.0.5.1. User Authentication Queries 27
         2.0.5.2. User Registration Query 27
         2.0.5.3. Session Management 27
         2.0.5.4. Leaderboard Queries 28
         2.0.5.5. Game Submission 28
         2.0.5.6. Statistics Trigger 29
      2.0.6. Data Structures 29
         2.0.6.1. Circular Queue [7] 29
         2.0.6.2. HashMap 29
         2.0.6.3. Option/Result Types 29

2.0.6.4. Arc [8] 30
2.0.6.5. Mutex 30
2.0.7. Diagrams 31
2.0.8. Frontend 31
2.0.9. Backend 35
3. Technical Solution 39
3.0.1. Code Contents 39
3.0.2. Skill table 41
3.0.3. Completeness of Solution 42
3.0.4. Code Quality 42
3.0.5. Source Code 43
3.0.5.1. Grid Component 43
3.0.5.2. Authentication 56
3.0.5.2.1. Frontend 56
3.0.5.2.2. Backend 58
3.0.5.3. Queue 60
3.0.5.4. Leaderboard 61
3.0.5.5. Server Routing 64
3.0.5.6. Singleplayer Game Management 65
3.0.5.7. Backend Error Handling 69
3.0.5.8. Database Models 70
3.0.5.9. Multiplayer game management 71
3.0.5.10. WASM 77
3.0.5.11. Settings Component 79
3.0.5.12. Layout and Styling 83
3.1. Testing 86
4. Evaluation 91
4.0.1. Evaluation Against Objectives 91
4.0.2. User Feedback 93
4.0.2.1. Client Feedback 93
4.0.2.2. Test User Feedback 93
4.0.3. Future Improvements 93
4.0.4. Conclusion 93
5. Bibliography 94

1. Analysis

1.1. 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.

1.2. 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.

1.3. Client

1.3.1. Client Synopsys (conclusion)

The Client is Alexander Tahiri, a student and part-time Software Developer, 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 actives 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 whithin 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.

1.3.2. 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.

1.4. 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

1.5. Research

1.5.1. Similar Solutions

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

1.5.1.1. 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 DoubleTapp in several key ways:

Figure 1: Tetris UI (jstris.jezevec10.com)

- 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)[6], 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

1.5.1.2. 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.

1.5.1.3. 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

1.5.2. Multiplayer

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

Method	Pros	Cons
HTTP [9]	• Simple implementation • Reasonably performant • Easily Debuggable • widely supported	• Slow with many simultaneous users • Requires entire connection sequence for each request • relatively high latency

		• not designed for bidirectional communication
UDP [10]	• Very performant • allows for low level optimisations • minimal overhead	• susceptible to packet loss, and is not guaranteed to have data parity (important for doubletapp) • complex to implement, and difficult to interconnect with existing libraries without significant performance declines • often blocked by firewalls • no ordering guarantees
Websockets [11]	• allows for fast and safe data transmission • relatively complex to implement, as need to handle assignment of websockets to individual games • compatible with existing web server libraries • fully duplex, no need to reestablish connection sequence each request	• websockets don't recover when connections are terminated • some networks block the websocket protocol, limiting accessibility • high memory usage per connection compared to UDP/HTTP

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.

1.5.3. 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 [4] 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)	• Simple implementation • Fast for small applications	• Cryptographically broken

	• Variable key size	• Biased output in early stream • Vulnerable to related-key attacks
Seedrandom.js	• Browser-friendly • Multiple algorithm options • Good for web applications	• JavaScript performance limitations • Depends on implementation quality • Not cryptographically secure by default
ChaCha20 [12], [13]	• Cryptographically secure • Excellent statistical properties • Fast in software (no large tables) • Parallelizable	• Complex implementation • Overkill for non-security applications • Higher computational cost
Xorshift [14]	• Extremely fast • Simple implementation • Good statistical quality	• Not cryptographically secure • Simpler variants have known weaknesses • Some states can lead to poor quality
Linear Congruential Generator (LCG) [15], [16]	• Simplest implementation • Very fast • Small state	• Poor statistical quality • Short period for 32-bit implementations • Predictable patterns
Mersenne Twister [17]	• Very long period • Good statistical properties • Industry standard in many fields	• Large state (2.5KB) • Not cryptographically secure • Slow initialization
Xoshiro256+/++ [4]	• Excellent speed • Great statistical properties • Small state (256 bits) • Fast initialization	• Not cryptographically secure • Newer algorithm (less scrutiny) • Some variants have issues with specific bits

PCG (Permuted Congruential Generator) [18]	• Excellent statistical properties • Small state • Good performance • Multiple variants available	• More complex than basic PRNGs • Not cryptographically secure • Relatively new

PRNG Algorithm	Estimated Time	Cycle Length	State Size	Performance
ARC4 [19]	Medium	$10^{\{100\}}$	256 bits	Moderate
seedrandom.js [20]	Medium	(multiple selectable algorithms)	Varies by algorithm	Moderate (JS limited)
ChaCha20 [12], [13]	High	2^{256}	384 bits	High for crypto
Xorshift [14]	Very Low	$2^{128} - 1$	128-256 bits	Very High
Linear Congruential Generator (LCG) [15], [16]	Extremely Low	Up to 2^{32}	32-64 bits	Extremely High
Mersenne Twister [17]	Medium	$2^{19937} - 1$	2.5 KB (19937 bits)	Moderate
Xoshiro256+/++ [4]	Very Low	$2^{256} - 1$	256 bits	Very High
PCG (Permuted Congruential Generator) [18]	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

1.5.4. 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.

1.5.4.1. Player timings [1]

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

1.5.4.2. 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)
Djikstra'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.

1.5.4.3. Rust Memory Management [2]

in rust, variables are managed through a system called "ownership" which is enforced at compile time. Unlike languages with garbage collection (like JavaScript) or manual memory management (like C++), Rust uses three core rules:

1 Each value has a single owner variable
2 There can only be one owner at a time
3 When the owner goes out of scope, the value is automatically dropped

Rust distinguishes between stack and heap memory:

- **Stack**: Fixed-size, fast memory for data with known size at compile time (primitives like integers, booleans)
- **Heap**: Variable-size memory for data that may change size or needs to live beyond a function's scope

When a value is assigned from one variable to another, Rust's behavior depends on the data type:

- For simple types implementing the `Copy` trait (integers, booleans), values are duplicated
- For complex types like `String` or custom structs, ownership is transferred (moved) unless explicitly cloned

1.5.4.4. Postgres Triggers [3]

in postgres, triggers are a way to automatically execute a function when a specific event occurs, i.e a row is inserted, updated, or deleted, or a table is created, dropped, or truncated.

```
CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
EXECUTE FUNCTION function_name();
```

these are very useful, as they can minimize round trips to the server, and can be used to validate data, or perform other actions, i.e when a row is inserted, the trigger can be used to automatically update a counter, or perform other actions.

1.6. 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 PrototypeUI

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

1.7. Critical Path

Figure 6: Intended Critical Path

1.8. Objectives

1.8.1. 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
  3.6 settings are saved on the browsers
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 successfull 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
4.9 user can move using keybinds, or by holding down keys (Das,Arr)
4.10 user can react to changes in a successive manner
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 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
6.2 user can choose to see their personal leaderboard, or global leaderboard
6.3 user can select different dimensions
6.4 user can select different time limits
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 (authenticated by database)
8.2 user is shown error codes depending on if account already exists or their login details are incorrect
8.3 if user is signed up, their games can be saved
9 user can see rules / introduction
9.1 if user has never played before, popup should be shown to user
9.2 user has option to view again
10 user can use keybinds to reset game

2. Documented Design

2.0.1. Libraries Used

2.0.1.1. 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	svelte.dev
SvelteKit	2.0.0+	Full-stack framework built on Svelte, allowing for simplification of operations between the frontend and the backend	kit.svelte.dev
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	tailwindcss.com
Tailwind Catppuccin	0.1.6	Client-requested color scheme	GitHub
Svelte Material Icons	3.0.5	Icon library for Svelte, MIT licensed	npm
UUID	11.0.4	frontend library for generating UUID's, used for game management	npm
Xoshiro WASM	Local	Custom WASM implementation of Xoshiro256+	in code
TypeScript	5.0.0+	Typed JavaScript for better development	typescriptlang.org
Vite	5.0.3	Modern frontend build tool, used in frontend to allow for fast development and optimized production builds	vitejs.dev
Vite Plugin WASM	3.4.1	Vite plugin for WebAssembly integration	npm

2.0.1.2. Backend Libraries

Name	Version	Reason	Link
Axum	0.7.5	Modern Rust web framework with WebSocket support, one of the fastest web frameworks currently	GitHub

		available, asynchronous and type-safe	
Axum-Extra	0.9.4	Extension crate for Axum with additional features like cookie handling and typed headers	GitHub
Tokio	1.39.2	Asynchronous runtime for Rust, required by axum and used for thread handling in websockets	tokio.rs
SQLx	0.8.0	Async SQL toolkit with compile-time checked queries, used for database operations, inherently supports pooling and multithreading.	GitHub
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	serde.rs
Serde_json	1.0.128	JSON implementation for Serde, used for parsing and generating JSON data in WebSocket communication	GitHub
Bcrypt	0.17.0	Password hashing library, used before storing passwords in the database, salted and performant, although slightly outdated	crates.io
Tower-HTTP	0.5.2	HTTP middleware stack, baseline from axum, used for low level websocket handling	GitHub
UUID	1.7.0	Library for generating UUIDs, used for game management	crates.io
ULID	1.1.3	Sortable identifier generation, used for game management	crates.io
Validator	0.20.0	Data validation library, used for validating user input	crates.io
Chrono	0.4.37	Date and time library with timezone support, used for	crates.io

		handling timestamps and durations to verify games	
SCC	2.1.11	Concurrent collections for server applications, performant asynchronous hashmaps	crates.io
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	crates.io
Dotenvy	0.15.7	Loads environment variables from .env files, used for configuration management	crates.io
Futures	0.3.31	Async programming primitives, used for handling asynchronous websocket operations	crates.io
Rand	0.8.5	Random number generation utilities, used for game seeding	crates.io
Thiserror	2.0.11	Error handling library that simplifies custom error types, used for robust error management	crates.io
Tracing-subscriber	0.3.18	Utilities for implementing and composing `tracing` subscribers, used for logging and diagnostics	crates.io

2.0.2. Iterative Design

i have been using git to manage the project, including reverting to commits when i made mistakes, and using branches to experiment with different features, and then merging them into the main branch.

additionally git has statistics, which allows me to see the changes i have made to the code, and the impact of the changes, which allows me to make more informed decisions about the code.

2.0.3. Algorithms

2.0.3.1. Xoshiro256+ [4]

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)
```

2.0.3.2. 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))
```

2.0.3.3. 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()
}
```

2.0.3.4. MergeSort [5]

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(nlog(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
```

2.0.3.5. Standard deviation [1]

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 ith 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.

```rust
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()
}
```

2.0.3.6. Delayed Auto Shift [6]

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 customizeable, this isn't much of a problem.

```rust
// 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);
    }
```

2.0.3.7. Game Verification

```rust
pub async fn verify_moves(moves: Vec<Move>, size: u8, seed: u32) -> Result<u32, String> {
    //this is assuming we start at 0,0 and size,size (should be a client side force, now enforced)
    let mut rng = sillyrng::Xoshiro256plus::new(Some(seed as u64));
    let mut grid: Vec<bool> = vec![false; (size * size) as usize];
```

```
    let mut blue_coords: (u8, u8) = (0, 0);
    let mut red_coords: (u8, u8) = (size - 1, size - 1);
    let mut score = 0;
    let mut distance = 0;
    let mut anomalous_distances = 0;
    let mut optimal_distance = 0;
    let mut count = 0;
    // grid initialisation
    while count < size {
        let x: u8 = (rng.next() * size as f64).floor() as u8;
        let y: u8 = (rng.next() * size as f64).floor() as u8;
        if grid[(x * size + y) as usize] == false {
            grid[(x * size + y) as usize] = true;
            count += 1;
        }
    }
    for i in moves.iter() {
        match i {
            Move::CursorRedUp => {
                red_coords.1 = (red_coords.1 as i8 - 1).max(0) as u8;
                distance += 1;
            }
            // all other moves are handled in the same way, albeit with different coordinates
            Move::Submit => {
                // simple anomaly check
                if distance <= optimal_distance {
                    anomalous_distances += 1;
                }
                distance = 0;
                // verifies that the cursors are not on the same cell, and that both cursors are on active tiles
                if grid[(red_coords.0 * size + red_coords.1) as usize]
                    && grid[(blue_coords.0 * size + blue_coords.1) as usize]
                    && !(blue_coords == red_coords)
                {
                    score += 1;
                    let mut count = 0;
                    let r = red_coords.0 * size + red_coords.1;
                    let b = blue_coords.0 * size + blue_coords.1;
                    // regenerates two new tiles, using the same rng as the client side
                    while count < 2 {
                        let x: u8 = (rng.next() * size as f64).floor() as u8;
                        let y: u8 = (rng.next() * size as f64).floor() as u8;
                        if !grid[(x * size + y) as usize]
                            && (x * size + y != r || x * size + y != b)
                        {
                            grid[(x * size + y) as usize] = true;
                            count += 1;
                        }
                    }
                    // resets selected tiles to inactive
                    grid[r as usize] = false;
                    grid[b as usize] = false;
                    // calculates the optimal distance for the new grid
                    optimal_distance =
                        get_optimal_paths(grid.clone(), red_coords, blue_coords, size)
                            .await
                            .iter()
```

```
                            .min()
                            .unwrap_or(&0)
                            .to_owned();
                } else {
                    score = 0
                }
            }
        }
    }


```

2.0.4. API routes

although API routes are simple, the actual functions linked are quite complex, and require a lot of error handling, which is why I have included them here.

Method	Route	Handler Function	Description
POST	“/get-seed”	“create_seed”	Creates a new game seed and returns it along with a game ID.
POST	“/submit-game”	“submit_game”	Submits a completed game for verification and scoring.
ANY	“/game”	“ws_upgrader”	Handles WebSocket connections for game actions.
POST	“/get_scores”	“misc::get_scores”	Retrieves game scores, with options for pagination and user-specific filtering.
POST	“/user/signup”	“misc::signup”	Registers a new user.
POST	“/user/login”	“misc::login”	Logs in an existing user.

Middleware	“(All authenticated routes)”	“misc::authorization”	Middleware applied to all routes, checking for a valid session cookie to authenticate the user.

2.0.5. Database Design and Queries

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

ERM

2.0.5.1. 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.

2.0.5.2. 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.

2.0.5.3. 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.

2.0.5.4. 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.

2.0.5.5. Game Submission

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

inserts a game into the game table, with the game id, score, average time, dimension, time limit, and user id.

2.0.5.6. 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.

2.0.6. Data Structures

2.0.6.1. Circular Queue [7]

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

2.0.6.2. 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.

2.0.6.3. Option/Result Types

an Optional type, is a simple data structure (technically a monad) 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

2.0.6.4. Arc [8]

Crucial feature in the rust programming language, the Arc(atomic reference counted pointer), allows for thread safe sharing of data, by using a reference counted pointer, which allows for the data to be shared between threads, and the data not needing to be cloned, allowing for efficient memory management, and the data to be shared between threads safely, without having to worry about race conditions.
refer to Rusts Memory Management section for more information about how rust handles variable scopes.
I use Arcs to wrap all websocket connections, additionally wrapping the game state, so it can be concurrently accessed by multiple threads, allowing for many users to simultaneously modify the game state without loss of atomicity.

2.0.6.5. Mutex

the Mutex (mutual exclusion) is a data structure, that when combined with the Arc, allows for thread safe data sharing, you “lock” the data when you need to write to it, which means all other references to the mutex are blocked from writing to the mutex, althought this has performance downgrades, it pairs well with Arcs, as you don’t have to worry about race conditions, and can still have multiple readers.

2.0.7. Diagrams

2.0.8. Frontend

Figure 8: Client Component and Flow diagram

Figure 9: Grid Component

Figure 10: Game Handler Flowchart

Figure 11: Singleplayer Game Flowchart

2.0.9. Backend

Figure 12: Multiplayer Game Flowchart

Figure 13: WebSocket message diagram

Figure 14: Backend Multiplayer Flowchart

Figure 15: Backend Flowchart

Figure 16: Class Diagram

3. Technical Solution

3.0.1. Code Contents

Component	Description	Path/Location

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

3.0.2. Skill table

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

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

3.0.3. Completeness of Solution

everything apart from cheater game verification is complete

3.0.4. 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.

Composite Framework Scores

Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. ***172 total frameworks ranked, 171 visible, 1 hidden by filters. See filter panel above.***

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score	
1	may-minihttp	3,102,062	1,448,273	88,108	1,327,378	58,532	27,906,423	**21,495**	100.0%
2	xitca-web	3,023,263	1,379,908	89,666	1,240,072	63,028	27,970,357	**21,293**	99.1%
3	ntex	2,963,471	1,380,317	88,589	1,210,348	58,757	24,925,515	**20,301**	94.4%
4	just	2,868,204	1,289,246	89,470	982,024	58,885	25,739,560	**19,277**	89.7%
5	libh2o	2,923,581	1,348,681	86,788	1,226,814	58,146	9,496,470	**17,678**	82.2%
6	vert.x	2,889,592	1,315,264	88,708	1,040,599	58,300	14,340,940	**17,668**	82.2%
7	lithium	3,041,049	1,040,675	52,239	1,073,846	37,229	27,953,333	**17,464**	81.2%
8	axum	2,709,794	1,270,151	87,231	1,114,265	55,708	12,392,330	**17,184**	79.9%
9	quarkus	2,756,818	1,258,371	88,902	1,028,408	57,616	13,080,084	**17,139**	79.7%
10	vertx-web	2,836,665	1,313,163	89,289	856,756	58,138	13,010,230	**16,647**	77.4%
11	wizzardo-http	2,972,955	1,372,100	86,504	620,058	45,641	22,212,355	**16,506**	76.8%
12	uwebsockets.js	2,731,252	711,561	81,365	483,035	57,875	27,991,649	**16,224**	75.5%
13	officefloor	2,550,592	1,076,847	85,997	911,341	56,791	13,994,309	**16,170**	75.2%
14	jooby	2,705,638	1,105,323	86,183	917,719	52,019	13,795,851	**16,046**	74.6%
15	inverno	2,579,842	1,134,919	87,919	798,618	59,146	10,274,217	**15,437**	71.8%
16	viz	884,917	1,259,172	88,468	1,060,105	59,374	12,354,890	**15,390**	71.6%

Figure 17: TechEmpower Framework Benchmark

3.0.5. Source Code

3.0.5.1. Grid Component

rust

```
<script lang="ts">
  // general imports
  import Clock from 'svelte-material-icons/Timer.svelte';
  import Trophy from 'svelte-material-icons/Trophy.svelte';
  import Dice from 'svelte-material-icons/Dice5.svelte';
  import Meow from 'svelte-material-icons/ViewGrid.svelte';
  import Party from 'svelte-material-icons/PartyPopper.svelte';
  import { browser } from '$app/environment';
  import { getContext, onMount } from 'svelte';
  import { json } from '@sveltejs/kit';
  import { v4 as uuidv4 } from 'uuid';
  import { Xoshiro256plus } from 'xoshiro';

  async function initWasm() {
    rng = new Xoshiro256plus(BigInt(69));
  }

  let rng: Xoshiro256plus;
  if (browser) {
    initWasm().catch(console.error);
  }
  // variable initializations
  export let showModal;
  let state: any = getContext('state');
  let scoreboard: any = 0;
  let end = true;
  let interval: any;
  let dasIntervals = Array(8).fill(0);
  let gameStarted = false;
  let gameId = 0;
  let time = $state.timeLimit;
  let score = 0;
  let quota = 0;
  let playersLeft = 0;
  let moves: any = [];
  let grid = Array(Math.pow($state.size, 2)).fill(false);
  let cGrid = Array(Math.pow($state.size, 2)).fill('neutral');
  let wcursorX = 0;
```

```
let wcursorY = 0;
let acursorX = $state.size - 1;
let acursorY = $state.size - 1;
let lastActionTime = 0;
let temp_id: String = '';
let ws: WebSocket;
const initGrid = () => {
  gameStarted = false;
  wcursorX = 0;
  wcursorY = 0;
  acursorX = $state.size - 1;
  acursorY = $state.size - 1;

  grid = Array(Math.pow($state.size, 2)).fill(false);
};
const endGame = () => {
  score = 0;
  time = $state.timeLimit;
  wcursorX = 0;
  wcursorY = 0;
  acursorX = $state.size - 1;
  acursorY = $state.size - 1;
  moves = [];
  clearInterval(interval);

  // Close WebSocket if in multiplayer mode
  if ($state.gameMode === 'multiplayer' && ws) {
    ws.close();
    temp_id = '';
  }

  initGrid();
};
const startGame = () => {
  switch ($state.gameMode) {
    case 'timer':
      gameStarted = true;
      startTimer();
      break;
    case 'multiplayer':
      startMultiplayerGame();
    // case 'pulse':
    // case 'endless':
  }
};
const startMultiplayerGame = () => {
  if (ws) {
    ws.close();
  }
  ws = new WebSocket('/ws/game');
  ws.onopen = (e) => {
    console.log('WebSocket opened');
  };
  ws.onmessage = (e) => {
    const data = e.data;

    try {
      const message = JSON.parse(data);
```

```
        switch (message.type) {
          case 'Start':
            console.log('Game starting with seed:', message.data);
            gameStarted = true;
            rng = new Xoshiro256plus(BigInt(message.data));
            time = 5;
            wcursorX = 0;
            wcursorY = 0;
            acursorX = $state.size - 1;
            acursorY = $state.size - 1;
            interval = setInterval(() => {
              time -= 1;
              if (time <= 0) {
                clearInterval(interval);
              }
            }, 1000);
            fillGrid($state.size);
            break;
          case 'Quota':
            console.log(
              'Quota update:',
              message.data.quota,
              'players left:',
              message.data.players_left
            );
            quota = message.data.quota;
            playersLeft = message.data.players_left;
            time = 5;
            score = 0;
            clearInterval(interval);
            interval = setInterval(() => {
              time -= 1;
              if (time <= 0) {
                clearInterval(interval);
              }
            }, 1000);
            break;
          case 'Move':
            console.log('Received move:', message.data);
            break;
          case 'Out':
            console.log('player out placed', message.data);
            end = false;
            scoreboard = message.data;
            ws.close();
            break;
          case 'Win':
            console.log("you won!!!!", message.data);
            end = false;
            scoreboard = 1;
            ws.close();
            break;
          case 'ID':
            console.log('Received ID:', message.data);
            temp_id = message.data;
            break;
          case 'Ping':
            console.log('Received ping');
```

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

```
    }, 1000);
  };
  const submit = (time: any) => {
    if (!gameStarted && $state.gameMode === 'timer') {
      lastActionTime = Date.now();
      startGame();
      return;
    }
    if ($state.gameMode === 'timer') {
      moves.push(['Submit', time]);
    } else if ($state.gameMode === 'multiplayer') {
      ws.send(
        JSON.stringify({ type: 'Move', data: { player_id: `${temp_id}`, action: 'Submit' } })
      );
    }
    if (end) {
      let wIndex = wcursorX * $state.size + wcursorY;
      let aIndex = acursorX * $state.size + acursorY;
      let wStatus = grid[wIndex];
      let aStatus = grid[aIndex];
      if (wStatus && aStatus && (wcursorX !== acursorX || wcursorY !== acursorY)) {
        cGrid[wIndex] = 'correct';
        cGrid[aIndex] = 'correct';
        let count = 0;
        while (count < 2) {
          let x = Math.floor(rng.next() * $state.size);
          let y = Math.floor(rng.next() * $state.size);
          if (
            !grid[x * $state.size + y] &&
            (wIndex !== x * $state.size + y || aIndex !== x * $state.size + y)
          ) {
            grid[x * $state.size + y] = true;
            count += 1;
          }
        }
        grid[wIndex] = false;
        grid[aIndex] = false;
        score += 1;
      } else {
        if (wStatus && aStatus) {
          cGrid[wIndex] = 'incorrect';
        } else if (wStatus) {
          cGrid[aIndex] = 'incorrect';
          cGrid[wIndex] = 'correct';
        } else if (aStatus) {
          cGrid[wIndex] = 'incorrect';
          cGrid[aIndex] = 'correct';
        } else {
          cGrid[wIndex] = 'incorrect';
          cGrid[aIndex] = 'incorrect';
        }
        score = 0;
      }
      setTimeout(() => {
        cGrid[wIndex] = 'neutral';
        cGrid[aIndex] = 'neutral';
      }, 150);
    }
```

```
};
const onKeyUp = (e: any) => {
  let i = 0;
  switch (e.key) {
    case $state.keycodes.wU:
      i = 0;
      break;
    case $state.keycodes.wD:
      i = 1;
      break;
    case $state.keycodes.wL:
      i = 2;
      break;
    case $state.keycodes.wR:
      i = 3;
      break;
    case $state.keycodes.aU:
      i = 4;
      break;
    case $state.keycodes.aD:
      i = 5;
      break;
    case $state.keycodes.aL:
      i = 6;
      break;
    case $state.keycodes.aR:
      i = 7;
      break;
  }
  clearInterval(dasIntervals[i]);
  dasIntervals[i] = false;
};
const onKeyDown = (e: any) => {
  if (!gameStarted && $state.gameMode === 'multiplayer') {
    return;
  }
  const timeDiff = Date.now() - lastActionTime;
  switch (e.key) {
    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);
        }, $state.dasDelay);
      }
      wcursorY = Math.max(wcursorY - 1, 0);
      if ($state.gameMode === 'multiplayer') {
```

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

```
        wcursorX = Math.max(wcursorX - 1, 0);
        if ($state.gameMode === 'multiplayer') {
          ws.send(
            JSON.stringify({
              type: 'Move',
              data: { player_id: `${temp_id}`, action: 'CursorBlueLeft' }
            })
          );
        }
        moves.push(['CursorBlueLeft', timeDiff]);
        lastActionTime = Date.now();
        break;
      case $state.keycodes.wR:
        if (dasIntervals[3] == false) {
          dasIntervals[3] = setTimeout(() => {
            dasIntervals[3] = setInterval(() => {
              wcursorX = Math.min(wcursorX + 1, $state.size - 1);
              if ($state.gameMode === 'multiplayer') {
                ws.send(
                  JSON.stringify({
                    type: 'Move',
                    data: { player_id: `${temp_id}`, action: 'CursorBlueRight' }
                  })
                );
              }
              moves.push(['CursorBlueRight', Date.now() - lastActionTime]);
              lastActionTime = Date.now();
            }, $state.das);
          }, $state.dasDelay);
        }
        wcursorX = Math.min(wcursorX + 1, $state.size - 1);
        if ($state.gameMode === 'multiplayer') {
          ws.send(
            JSON.stringify({
              type: 'Move',
              data: { player_id: `${temp_id}`, action: 'CursorBlueRight' }
            })
          );
        }
        moves.push(['CursorBlueRight', timeDiff]);
        lastActionTime = Date.now();
        break;
      case $state.keycodes.aU:
        if (dasIntervals[4] == false) {
          dasIntervals[4] = setTimeout(() => {
            dasIntervals[4] = setInterval(() => {
              acursorY = Math.max(acursorY - 1, 0);
              if ($state.gameMode === 'multiplayer') {
                ws.send(
                  JSON.stringify({
                    type: 'Move',
                    data: { player_id: `${temp_id}`, action: 'CursorRedUp' }
                  })
                );
              }
              moves.push(['CursorRedUp', Date.now() - lastActionTime]);
              lastActionTime = Date.now();
            }, $state.das);
```

```
          }, $state.dasDelay);
        }
        acursorY = Math.max(acursorY - 1, 0);
        if ($state.gameMode === 'multiplayer') {
          ws.send(
            JSON.stringify({
              type: 'Move',
              data: { player_id: `${temp_id}`, action: 'CursorRedUp' }
            })
          );
        }
        moves.push(['CursorRedUp', timeDiff]);
        lastActionTime = Date.now();
        break;
      case $state.keycodes.aD:
        if (dasIntervals[5] == false) {
          dasIntervals[5] = setTimeout(() => {
            dasIntervals[5] = setInterval(() => {
              acursorY = Math.min(acursorY + 1, $state.size - 1);
              if ($state.gameMode === 'multiplayer') {
                ws.send(
                  JSON.stringify({
                    type: 'Move',
                    data: { player_id: `${temp_id}`, action: 'CursorRedDown' }
                  })
                );
              }
              moves.push(['CursorRedDown', Date.now() - lastActionTime]);
              lastActionTime = Date.now();
            }, $state.das);
          }, $state.dasDelay);
        }
        acursorY = Math.min(acursorY + 1, $state.size - 1);
        if ($state.gameMode === 'multiplayer') {
          ws.send(
            JSON.stringify({
              type: 'Move',
              data: { player_id: `${temp_id}`, action: 'CursorRedDown' }
            })
          );
        }
        moves.push(['CursorRedDown', timeDiff]);
        lastActionTime = Date.now();
        break;
      case $state.keycodes.aL:
        if (dasIntervals[6] == false) {
          dasIntervals[6] = setTimeout(() => {
            dasIntervals[6] = setInterval(() => {
              acursorX = Math.max(acursorX - 1, 0);
              if ($state.gameMode === 'multiplayer') {
                ws.send(
                  JSON.stringify({
                    type: 'Move',
                    data: { player_id: `${temp_id}`, action: 'CursorRedLeft' }
                  })
                );
              }
              moves.push(['CursorRedLeft', Date.now() - lastActionTime]);
```

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

```
    while (placed < count) {
      let x = Math.floor(rng.next() * $state.size);
      let y = Math.floor(rng.next() * $state.size);
      if (grid[x * $state.size + y] == false) {
        grid[x * $state.size + y] = true;
        placed += 1;
      }
    }
  };
  initGrid();
</script>

<div class="">
  {#if end}
    <div class="flex flex-row text-3xl text-text justify-between py-2">
      <div class="flex flex-row items-center">
        <Clock />
        <div class="px-2 {time < 3 ? (time < 2 ? 'text-red' : 'text-peach') : 'text-green'}">
          {time}
        </div>
      </div>
      <div class="flex flex-row items-center">
        <Trophy />
        <div class="px-2">
          {$state.gameMode === 'multiplayer' ? `${score}/${quota} (${playersLeft})` : score}
        </div>
      </div>
    </div>
    <div class="flex flex-col items-center">
      <div class="relative w-fit h-fit">
        {#if $state.gameMode === 'multiplayer' && (!temp_id || !gameStarted)}
          <div
            class="absolute top-0 left-0 right-0 bottom-[4.5rem] flex items-center justify-center z-10 bg-base/80"
          >
            {#if !temp_id}
              <button
                class="px-4 py-2 rounded-lg transition-colors duration-300 bg-lavender text-mantle hover:bg-rosewater"
                on:click={startMultiplayerGame}
              >
                Join Game
              </button>
            {:else}
              <div class="text-text text-3xl flex flex-col items-center gap-4">
                <div class="flex items-center gap-2">Waiting for players...</div>
              </div>
            {/if}
          </div>
        {/if}
        <!-- svelte-ignore a11y-autofocus -->
          <div class="w-fit h-fit flex flex-col" autofocus>
            {#each Array($state.size) as _, col}
              <div class="w-fit h-fit flex flex-row">
                {#each Array($state.size) as _, row}
                  <div
                    id={grid[row * $state.size + col]}
                    class="{cGrid[row * $state.size + col] === 'correct'
```

```
                      ? 'bg-green'
                      : cGrid[row * $state.size + col] === 'incorrect'
                        ? 'bg-red'
                        : grid[row * $state.size + col]
                          ? 'bg-crust'
                          : 'bg-text'}

                w-32 h-32 border-crust border flex items-center justify-center transition-colors duration-100"
                  >
                    <div
                      class="h-8 w-8 {row == wcursorX && col == wcursorY
                        ? 'border-t-blue border-l-blue border-t-8 border-l-8'
                        : ''}  {row == acursorX && col == acursorY
                        ? 'border-b-red border-r-red border-b-8 border-r-8'
                        : ''}"
                    />
                  </div>
                {/each}
              </div>
            {/each}
            <div class="text-text flex flex-row text-2xl py-4 justify-between">
              <div class="flex flex-row">
                <select
                  id="gamemodes"
                  name="modes"
                  class="bg-surface0 px-2"
                  bind:value={$state.gameMode}
                >
                  <label for="gamemodes" class="pr-4"> GAMEMODE: </label>
                  <option value="timer"> TIME </option>
                  <option value="multiplayer"> MULTIPLAYER </option>
                  <option value="endless"> ZEN </option>
                </select>
              </div>
              <select
                id="size"
                name="sizes"
                class="bg-surface0 px-2"
                bind:value={$state.size}
                on:change={() => {
                  endGame();
                }}
              >
                <option value={4}> 4x4 </option>
                <option value={5}> 5x5 </option>
                <option value={6}> 6x6 </option>
              </select>
              <select
                id="time"
                name="times"
                class="bg-surface0 px-2 {$state.gameMode == 'timer'
                  ? 'bg-surface0'
                  : 'bg-surface0/0 text-crust/0'}"
                bind:value={$state.timeLimit}
                on:change={() => {
                  time = $state.timeLimit;
                  endGame();
```

```
                }}
              >
                <option value={30}> 30s </option>
                <option value={45}> 45s </option>
                <option value={60}> 60s </option>
              </select>
              <button class="bg-surface0 px-2" on:click={endGame}> RESET </button>
            </div>
          </div>
      </div>
    </div>
  {:else}
    <div class="text-text flex align-right flex-col w-96">
      <div class="text-5xl py-2 font-bold flex items-center border-b-4 border-b-subtext0">
        <Party class="mr-4" />game ended
      </div>
      <div class="text-4xl py-2 flex items-center justify-between">
        score: {score}
        <div class="text-overlay1">
          {#if $state.gameMode === 'multiplayer' && scoreboard > 0}
            Position: #{scoreboard}
          {:else}
            #{scoreboard}
          {/if}
        </div>
      </div>
      <div class="flex-col items-center text-3xl justify-between pb-2">
        <div class="flex items-center my-1">
          <Dice /> gamemode:
          <div class="ml-1 text-overlay1">{$state.gameMode}</div>
        </div>
        <div class="flex items-center my-1">
          <Meow /> size:
          <div class="ml-1 text-overlay1">{$state.size}x{$state.size}</div>
        </div>
        <div class="flex items-center my-1">
          {#if $state.gameMode == 'timer'}
            <Clock /> time:
            <div class="ml-1 text-overlay1">{$state.timeLimit}s</div>
          {/if}
        </div>
      </div>
      <button
        class="text-2xl h-12 my-2 bg-blue/80 hover:bg-blue border-rosewater transition-colors duration-150 font-bold"
        on:click={() => {
          end = true;
          endGame();
        }}
      >
        submit score?
      </button>
      <button
        class="text-2xl h-12 my-2 bg-mauve/80 hover:bg-mauve border-rosewater transition-colors duration-150 font-bold"
        on:click={() => {
          end = true;
          endGame();
```

```
        }}
      >
        play again?
      </button>
    </div>
  {/if}
</div>

<svelte:window on:keydown={onKeyDown} on:keyup={onKeyUp} />
```

3.0.5.2. Authentication

3.0.5.2.1. Frontend

svlt

```svelte
<script lang="ts">
    import { goto, invalidateAll } from "$app/navigation";
    let isSignup = true;
    let error = "";

    async function handleSubmit(e: SubmitEvent) {
      e.preventDefault();
      let data = new URLSearchParams(new FormData(e.target as HTMLFormElement));
      // ternary operators
      let path = isSignup ? "/api/user/signup" : "/api/user/login";
      const res = await fetch(path, {
        method: "POST",
        headers: {
          "Content-Type": "application/x-www-form-urlencoded",
        },
        body: data,
      });
      if (res?.ok) {
        await invalidateAll();
        goto("/");
      } else {
        switch (res?.status) {
          case 409:
            error = "Username or email already exists";
            break;
          case 401:
            error = "Invalid credentials";
            break;
          case 404:
            error = "not found";
            break;
          default:
            error = "An unknown error occurred";
            break;
        }
      }
    }
  </script>
```

```
39
40   <div class="min-h-screen w-screen flex items-center justify-center bg-base">
41     <div class="w-full max-w-md mx-4 bg-mantle rounded-xl shadow-xl p-8">
42       {#if isSignup}
43         <div class="flex flex-col gap-8">
44           <div class="text-center">
45             <h1 class="text-3xl font-medium text-lavender mb-2">Create Account</h1>
46           </div>
47           <form on:submit={handleSubmit} class="flex flex-col gap-6">
48             <div class="flex flex-col gap-4">
49               <input
50                 type="text"
51                 name="username"
52                 placeholder="Username"
53                 class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
   focus:border-lavender transition-colors"
54               />
55               <input
56                 type="password"
57                 name="password"
58                 placeholder="Password"
59                 class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
   focus:border-lavender transition-colors"
60               />
61             </div>
62             {#if error}
63               <div class="bg-red/10 border border-red/20 text-red px-4 py-3 rounded-lg text-sm">
64                 {error}
65               </div>
66             {/if}
67             <button
68               type="submit"
69               class="w-full px-4 py-3 rounded-lg font-medium bg-lavender text-mantle hover:bg-
   rosewater transition-colors"
70             >
71               Sign Up
72             </button>
73           </form>
74           <button
75             on:click={() => (isSignup = false)}
76             class="text-subtext0 hover:text-text transition-colors pt-2"
77           >
78             Already have an account? Login here
79           </button>
80         </div>
81       {:else}
82         <div class="flex flex-col gap-8">
83           <div class="text-center">
84             <h1 class="text-3xl font-medium text-lavender mb-2">Welcome Back!</h1>
85           </div>
86           <form on:submit={handleSubmit} class="flex flex-col gap-6">
87             <div class="flex flex-col gap-4">
88               <input
89                 type="username"
90                 name="username"
91                 placeholder="Username"
92                 class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0
   focus:border-lavender transition-colors"
```

```
            />
            <input
              type="password"
              name="password"
              placeholder="Password"
              class="w-full px-4 py-3 rounded-lg bg-base text-text border border-surface0 focus:border-lavender transition-colors"
            />
          </div>
          {#if error}
            <div class="bg-red/10 border border-red/20 text-red px-4 py-3 rounded-lg text-sm">
              {error}
            </div>
          {/if}
          <button
            type="submit"
            class="w-full px-4 py-3 rounded-lg font-medium bg-lavender text-mantle hover:bg-rosewater transition-colors"
          >
            Login
          </button>
        </form>
        <button
          on:click={() => (isSignup = true)}
          class="text-subtext0 hover:text-text transition-colors pt-2"
        >
          Don't have an account? Sign up here
        </button>
      </div>
    {/if}
  </div>
</div>
```

3.0.5.2.2. Backend

```rust
pub struct SignForm {
    pub(crate) username: String,
    #[validate(length(min = 8))]
    pub(crate) password: String,
}

#[axum::debug_handler]
pub async fn signup(
    State(state): State<Arc<AppState>>,
    headers: HeaderMap,
    Form(details): Form<SignForm>,
) -> Result<CookieJar, AppError> {
    let mut conn = state.db.acquire().await?;
    let jar = CookieJar::from_headers(&headers);
    let exists: Option<(i64,)> = sqlx::query_as("SELECT 1 FROM \"user\" WHERE username = $1")
        .bind(&details.username)
        .fetch_optional(&mut *conn)
        .await?;

    if exists.is_some() {
        return Err(AppError::Status(StatusCode::CONFLICT));
    }
```

```rust

    let hashed = bcrypt::hash(details.password, bcrypt::DEFAULT_COST)?;
    let user_id = uuid::Uuid::new_v4();

    sqlx::query!(
        "INSERT INTO \"user\" (id, username, password) VALUES ($1, $2, $3)",
        user_id,
        details.username,
        hashed
    )
    .execute(&mut *conn)
    .await?;

    let session_id = uuid::Uuid::new_v4();
    sqlx::query!(
        "INSERT INTO session (ssid, user_id, expiry_date) VALUES ($1, $2, NOW() + INTERVAL '7 DAYS')",
        session_id,
        user_id
    )
    .execute(&mut *conn)
    .await?;

    Ok(jar.add(
        Cookie::build(("session", session_id.to_string()))
            .path("/")
            .build(),
    ))
}

pub async fn login(
    State(state): State<Arc<AppState>>,
    jar: CookieJar,
    Form(details): Form<SignForm>,
) -> Result<CookieJar, AppError> {
    let mut conn = state.db.acquire().await?;

    let user: Option<(uuid::Uuid, String)> =
        sqlx::query_as("SELECT id, password FROM \"user\" WHERE username = $1")
            .bind(&details.username)
            .fetch_optional(&mut *conn)
            .await?;

    let (user_id, hashed) = user.ok_or(AppError::Status(StatusCode::UNAUTHORIZED))?;

    if !bcrypt::verify(details.password, &hashed)? {
        return Err(AppError::Status(StatusCode::UNAUTHORIZED));
    }

    let session_id = uuid::Uuid::new_v4();
    sqlx::query!(
        "INSERT INTO session (ssid, user_id, expiry_date) VALUES ($1, $2, NOW() + INTERVAL '7 DAYS')",
        session_id,
        user_id
    )
    .execute(&mut *conn)
    .await?;
```

```rust

    Ok(jar.add(
        Cookie::build(("session", session_id.to_string()))
            .path("/")
            .build(),
    ))
}

#[axum::debug_middleware]
pub async fn authorization(
    State(state): State<Arc<AppState>>,
    headers: HeaderMap,
    mut request: Request,
    next: Next,
) -> Result<Response, AppError> {
    let jar = CookieJar::from_headers(&headers);
    let user = if let Some(cookie) = jar.get("session") {
        if let Ok(session_id) = uuid::Uuid::parse_str(cookie.value()) {
            let mut conn = state.db.acquire().await?;
            sqlx::query_as!(
                crate::models::UserExt,
                r#"
                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()
                "#,
                session_id
            )
            .fetch_optional(&mut *conn)
            .await?
        } else {
            None
        }
    } else {
        None
    };

    request.extensions_mut().insert(user);
    let response = next.run(request).await;
    Ok(response)
}
```

3.0.5.3. Queue

rust

```rust
#[derive(Debug)]
pub struct Queue<T> {
    items: [Option<T>; 64],
    pub size: usize,
    front: usize,
}

impl<T> Queue<T> {
    pub fn new() -> Self {
        Self {
            items: std::array::from_fn(|_| None),
            size: 0,
```

```
            front: 0,
        }
    }

    pub fn enqueue(&mut self, item: T) -> bool {
        if self.size == self.items.len() {
            return false;
        }
        let rear = (self.front + self.size) % self.items.len();
        self.items[rear] = Some(item);
        self.size += 1;
        true
    }

    pub fn dequeue(&mut self) -> Option<T> {
        if self.size == 0 {
            return None;
        }
        let item = self.items[self.front].take();
        self.front = (self.front + 1) % self.items.len();
        self.size -= 1;
        item
    }
}

impl<T> Default for Queue<T> {
    fn default() -> Self {
        Self::new()
    }
}
```

3.0.5.4. Leaderboard

svlt

```svlt
<script lang="ts">
  import { onMount } from 'svelte';

  let dimension = 4;
  let timeLimit = 30;
  let leaderboard: Array<[string, number]> = [];
  let currentPage = 1;
  let userOwned = false;
  onMount(() => {
    fetchScores();
  });

  async function fetchScores() {
    const res = await fetch(`/api/get_scores`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ page: currentPage, dimension, time_limit: timeLimit, user_scores: userOwned }),
    });
    const data = await res.json();
```

```
    leaderboard = mergesort(data);
  }

  function mergesort(arr: Array<[string, number]>): Array<[string, number]> {
    if (arr.length < 2) return arr;
    const mid = Math.floor(arr.length / 2);
    const left = mergesort(arr.slice(0, mid));
    const right = mergesort(arr.slice(mid));
    return merge(left, right);
  }

  function merge(left: Array<[string, number]>, right: Array<[string, number]>): Array<[string, number]> {
    let result = [];
    while (left.length && right.length) {
      if (left[0][1] > right[0][1]) {
        result.push(left.shift());
      } else {
        result.push(right.shift());
      }
    }
    return [...result, ...left, ...right];
  }
  //TODO make the dimension and timelimit a select
</script>

<div class="min-h-screen bg-mantle text-text p-8">
  <div class="text-3xl font-bold mb-6">Leaderboards</div>
  <div class="flex gap-4 items-center mb-6">
    <div class="font-semibold">Dimension:</div>
    <select id="size" name="dimension" class="bg-surface0 px-2" bind:value={dimension}>
      <option value={4}> 4x4 </option>
      <option value={5}> 5x5 </option>
      <option value={6}> 6x6 </option>
    </select>
    <div class="font-semibold">Time Limit:</div>
    <select id="size" name="dimension" class="bg-surface0 px-2" bind:value={timeLimit}>
      <option value={30}> 30s </option>
      <option value={45}> 45s </option>
      <option value={60}> 60s </option>
    </select>
    <div class="font-semibold">Personal Bests:</div>
    <input type="checkbox" bind:checked={userOwned} class="bg-surface0 px-2">
    <button on:click={fetchScores} class="px-4 py-1 rounded bg-green text-text font-semibold hover:bg-sky">Refresh</button>
  </div>
  <table class="min-w-full border-collapse">
    <thead class="bg-[#1e2030]">
      <tr>
        <th class="px-4 py-2 border border-text">Username</th>
        <th class="px-4 py-2 border border-text">Score</th>
      </tr>
    </thead>
    <tbody>
      {#each leaderboard as [user, score]}
        <tr class="hover:bg-text">
          <td class="px-4 py-2 border border-text">{user}</td>
          <td class="px-4 py-2 border border-text">{score}</td>
```

```
        </tr>
      {/each}
    </tbody>
  </table>
  <div>
    <button on:click={() => {currentPage -= 1; fetchScores()}} disabled={currentPage === 1} > back
</button>
    {currentPage}
    <button on:click={() => {currentPage += 1; fetchScores()}}  > next </button>
  </div>
</div>
```

rust

```rust

#[derive(Serialize, Deserialize, sqlx::FromRow)]
pub struct Score {
    username: String,
    score: Option<i16>,
}

#[derive(Serialize, Deserialize)]
pub struct GetScore {
    page: u32,
    dimension: u8,
    time_limit: u8,
    user_scores: bool,
}

#[axum::debug_handler]
pub async fn get_scores(
    State(state): State<Arc<AppState>>,
    Extension(user): Extension<Option<UserExt>>,
    Json(data): Json<GetScore>,
) -> Result<Json<Vec<(String, usize)>>, AppError> {
    let query_string = if data.user_scores && user.is_some() {
        r#"
        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
        "#
    } else {
        r#"
        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
        "#
    };
```

```rust
    let user_id = match user.is_some() {
        true => user.unwrap().id,
        false => uuid::Uuid::new_v4(),
    };
    let res: Vec<(String, usize)> = sqlx::query_as::<_, Score>(query_string)
        .bind(data.dimension as i32)
        .bind(data.time_limit as i32)
        .bind(data.page as i32)
        .bind(user_id)
        .fetch_all(&mut *state.db.acquire().await?)
        .await?
        .iter()
        .map(|x| (x.username.clone(), x.score.unwrap() as usize))
        .collect();
    Ok(Json(res))
}
```

3.0.5.5. Server Routing

```rust
[tokio::main]
async fn main() {
    // basic initialization
    dotenvy::dotenv().ok();

    let database_url = std::env::var("DATABASE_URL").expect("DB_URL must be set");

    let pool = PgPool::connect(&database_url).await.unwrap();

    tracing_subscriber::fmt::init();

    let state = Arc::new(AppState {
        games: Mutex::new(HashMap::new()),
        game_manager: GameManager {
            user_games: Arc::new(Mutex::new(Queue::<(
                ulid::Ulid,
                tokio::sync::mpsc::Sender<WebSocket>,
            )>::new())),
            cheater_games: Arc::new(Mutex::new(Queue::<(
                ulid::Ulid,
                tokio::sync::mpsc::Sender<WebSocket>,
            )>::new())),
            anon_games: Arc::new(Mutex::new(Queue::<(
                ulid::Ulid,
                tokio::sync::mpsc::Sender<WebSocket>,
            )>::new())),
        },
        db: pool,
    });
    let app = Router::new()
        .route("/get-seed", post(create_seed))
        .route("/submit-game", post(submit_game))
        .route("/game", any(ws_upgrader))
        .route("/get_scores", post(misc::get_scores))
        .route("/user/signup", post(misc::signup))
        .route("/user/login", post(misc::login))
        .layer(middleware::from_fn_with_state(
            state.clone(),
```

```rust
            misc::authorization,
        ))
        .with_state(state)
        .layer(CorsLayer::permissive())
        .layer(TraceLayer::new_for_http());

    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

3.0.5.6. Singleplayer Game Management

```rust
/// enum representing all possible moves done by the client
#[repr(u8)]
#[derive(Serialize, Deserialize, Debug, PartialEq, Eq, Clone, Copy)]
pub enum Move {
    CursorRedUp,
    CursorRedDown,
    CursorRedLeft,
    CursorRedRight,
    CursorBlueUp,
    CursorBlueDown,
    CursorBlueLeft,
    CursorBlueRight,
    Submit,
}
#[derive(Serialize, Deserialize)]
pub struct GameForm {
    dimension: u8,
    time_limit: u8,
}

#[derive(Debug, Copy, Clone)]
pub struct GameState {
    seed: u32,
    dimension: u8,
    time_limit: Duration,
    start_time: Instant,
}

#[derive(Serialize)]
pub struct Seed {
    id: String,
    seed: u32,
}
#[derive(Serialize, Deserialize, Debug)]
pub struct GameEnd {
    id: String,
    score: u32,
    //u32 is time difference in ms
    moves: Vec<(Move, u32)>,

}

/// creates a new seed using the implemented splitmix and xoshiro256+ algorithms from sillyrng
#[axum::debug_handler]
pub async fn create_seed(
```

```rust
    State(state): State<Arc<AppState>>,
    Json(form): Json<GameForm>,
) -> (StatusCode, Json<Seed>) {
    let game_id = ulid::Ulid::new();
    let seed = rand::random::<u32>();
    let game_state = GameState {
        seed,
        dimension: form.dimension,
        time_limit: Duration::from_secs(form.time_limit.into()),
        start_time: Instant::now(),
    };

    println!(
        "Creating game {} with dimension {} and time limit {}s",
        game_id, form.dimension, form.time_limit
    );

    state.games.lock().await.insert(game_id, game_state);

    let res = Json(Seed {
        id: game_id.to_string(),
        seed,
    });

    (StatusCode::OK, res)
}

#[axum::debug_handler]
pub async fn submit_game(
    State(state): State<Arc<AppState>>,
    Extension(user): Extension<Option<UserExt>>,
    Json(game): Json<GameEnd>,
) -> Result<(StatusCode, Json<u32>), AppError> {
    println!(
        "Received submission for game {} with {} moves",
        game.id,
        game.moves.len()
    );

    let id = ulid::Ulid::from_string(&game.id).unwrap();
    let lock = state.games.lock().await;
    let mut conn = state.db.acquire().await?;
    let details = lock.get(&id).unwrap();


    let elapsed = Instant::now().duration_since(details.start_time);
    if elapsed > details.time_limit + Duration::from_secs(3) {
        println!("Game {} exceeded time limit ({}s + 3s)", game.id, details.time_limit.as_secs());
        return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
    }

    let time = verify_timings(game.moves.iter().map(|(_, m)| *m).collect(), state.clone()).await;

    if !time.0 {
        println!("Rejected game {} due to suspicious timings", game.id);
        return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
    }
    let score = match verify_moves(
```

```
        game.moves.iter().map(|(m, _)| *m).collect(),
        details.dimension,
        details.seed,
    )
    .await
    {
        Ok(s) => s,
        Err(e) => {
            println!("{:?}", e);
            // TODO anomalous game pushing
            return Ok((StatusCode::NOT_ACCEPTABLE, Json(0)));
        }
    };
    if score == game.score {
        if let Some(u) = user {
            println!(
                "Game {} submitted with score {}, user exists : {}",
                game.id,
                score,
                u.clone().username
            );
            sqlx::query!("INSERT INTO \"game\" (game_id,score,average_time,dimension,time_limit,user_id) VALUES ($1,$2,$3,$4,$5,$6)",uuid::Uuid::new_v4(),score as i32,time.1, details.dimension as i32,30,u.id).execute(&mut *conn).await?;
        }
        Ok((StatusCode::OK, Json(score)))
    } else {
        Ok((StatusCode::NOT_ACCEPTABLE, Json(0)))
    }
}

pub async fn verify_moves(moves: Vec<Move>, size: u8, seed: u32) -> Result<u32, String> {
    //this is assuming we start at 0,0 and size,size (should be a client side force, now enforced)
    let mut rng = sillyrng::Xoshiro256plus::new(Some(seed as u64));
    let mut grid: Vec<bool> = vec![false; (size * size) as usize];
    let mut blue_coords: (u8, u8) = (0, 0);
    let mut red_coords: (u8, u8) = (size - 1, size - 1);
    let mut score = 0;
    let mut distance = 0;
    let mut anomalous_distances = 0;
    let mut optimal_distance = 0;
    let mut count = 0;
    while count < size {
        let x: u8 = (rng.next() * size as f64).floor() as u8;
        let y: u8 = (rng.next() * size as f64).floor() as u8;
        if grid[(x * size + y) as usize] == false {
            grid[(x * size + y) as usize] = true;
            count += 1;
        }
    }
    for i in moves.iter() {
        match i {
            Move::CursorRedUp => {
                red_coords.1 = (red_coords.1 as i8 - 1).max(0) as u8;
                distance += 1;
            }
            Move::CursorRedDown => {
```

```
                red_coords.1 = (red_coords.1 + 1).min(size - 1);
                distance += 1;
            }
            Move::CursorRedLeft => {
                red_coords.0 = (red_coords.0 as i8 - 1).max(0) as u8;
                distance += 1;
            }
            Move::CursorRedRight => {
                red_coords.0 = (red_coords.0 + 1).min(size - 1);
                distance += 1;
            }
            Move::CursorBlueUp => {
                blue_coords.1 = (blue_coords.1 as i8 - 1).max(0) as u8;
                distance += 1;
            }
            Move::CursorBlueDown => {
                blue_coords.1 = (blue_coords.1 + 1).min(size - 1);
                distance += 1;
            }
            Move::CursorBlueLeft => {
                blue_coords.0 = (blue_coords.0 as i8 - 1).max(0) as u8;
                distance += 1;
            }
            Move::CursorBlueRight => {
                blue_coords.0 = (blue_coords.0 + 1).min(size - 1);
                distance += 1;
            }
            Move::Submit => {
                if distance <= optimal_distance {
                    anomalous_distances += 1;
                }
                distance = 0;

                if grid[(red_coords.0 * size + red_coords.1) as usize]
                    && grid[(blue_coords.0 * size + blue_coords.1) as usize]
                    && !(blue_coords == red_coords)
                {
                    score += 1;
                    let mut count = 0;
                    let r = red_coords.0 * size + red_coords.1;
                    let b = blue_coords.0 * size + blue_coords.1;
                    while count < 2 {
                        let x: u8 = (rng.next() * size as f64).floor() as u8;
                        let y: u8 = (rng.next() * size as f64).floor() as u8;
                        if !grid[(x * size + y) as usize]
                            && (x * size + y != r || x * size + y != b)
                        {
                            grid[(x * size + y) as usize] = true;
                            count += 1;
                        }
                    }
                    grid[r as usize] = false;
                    grid[b as usize] = false;
                    optimal_distance =
                        get_optimal_paths(grid.clone(), red_coords, blue_coords, size)
                            .await
                            .iter()
                            .min()
```

```rust
                            .unwrap_or(&0)
                            .to_owned();
                } else {
                    score = 0
                }
            }
        }
    }
    println!(
        "Game completed with score {} (anomaly ratio: {:.2})",
        score,
        anomalous_distances as f64 / score as f64
    );
    Ok(score)
}


pub async fn get_optimal_paths(grid: Vec<bool>, r: (u8, u8), b: (u8, u8), size: u8) -> Vec<u32> {
    let mut paths = Vec::new();
    for i in 0..grid.len() {
        for j in 0..grid.len() {
            if grid[i] && grid[j] && i != j {
                let r_cell = ((i / size as usize) as u8, (i % size as usize) as u8);
                let b_cell = ((j / size as usize) as u8, (j % size as usize) as u8);
                let r_dist = (r.0.abs_diff(r_cell.0) + r.1.abs_diff(r_cell.1)) as u32;
                let b_dist = (b.0.abs_diff(b_cell.0) + b.1.abs_diff(b_cell.1)) as u32;
                paths.push(r_dist + b_dist);
            }
        }
    }
    paths
}

```

3.0.5.7. Backend Error Handling

rust

```rust
use axum::http::{Response, StatusCode};
use axum::response::IntoResponse;
use bcrypt::BcryptError;
use thiserror::Error;
#[derive(Error, Debug)]
pub enum AppError {
    #[error("statuscode")]
    Status(StatusCode),
    #[error("bcrypt error")]
    Hash(#[from] BcryptError),
    #[error("Ulid Encode Error")]
    UEncode(#[from] ulid::EncodeError),
    #[error("Ulid Decode Error")]
    UDecode(#[from] ulid::DecodeError),
    #[error("failed to deserialize")]
    Json(#[from] serde_json::Error),
    #[error("pool failed to execute")]
    Pool(#[from] sqlx::Error),
}

impl IntoResponse for AppError {
```

```rust
    fn into_response(self) -> axum::response::Response {
        let (body, code) = match self {
            AppError::Status(e) => ("", e),
            _ => ("Unknown", StatusCode::INTERNAL_SERVER_ERROR),
        };
        Response::builder().status(code).body(body.into()).unwrap()
    }
}



```

3.0.5.8. Database Models

```rust
#[derive(serde::Serialize, serde::Deserialize, Clone)]
pub struct User {
    pub id: uuid::Uuid,
    pub password: String,
    pub username: String,
    pub admin: Option<bool>,
    pub cheater: Option<bool>,
}

#[derive(serde::Serialize, serde::Deserialize, Clone, Debug)]
pub struct UserExt {
    pub id: uuid::Uuid,
    pub username: String,
    pub admin: Option<bool>,
    pub cheater: Option<bool>,
}

#[derive(serde::Serialize, serde::Deserialize)]
pub struct UserStatistics {
    pub stat_id: uuid::Uuid,
    pub highest_score: Option<i16>,
    pub victories: Option<i16>,
    pub games_played: Option<i16>,
    pub elo: Option<i16>,
    pub user_id: uuid::Uuid,
}

#[derive(serde::Serialize, serde::Deserialize)]
pub struct Game {
    pub game_id: uuid::Uuid,
    pub score: Option<i16>,
    pub average_time: Option<f32>,
    pub dimension: Option<i16>,
    pub time_limit: Option<i16>,
    pub user_id: uuid::Uuid,
}

#[derive(serde::Serialize, serde::Deserialize)]
pub struct Statistics {
    pub stat_id: uuid::Uuid,
    pub total_timings: Option<f32>,
    pub total_score: Option<i64>,
    pub games_played: Option<i64>,
```

```rust
}

#[derive(serde::Serialize, serde::Deserialize)]
pub struct AnomalousGames {
    pub agame_id: uuid::Uuid,
    pub moves: serde_json::Value,
    pub user_id: uuid::Uuid,
}

#[derive(serde::Serialize, serde::Deserialize)]
pub struct Session {
    pub ssid: uuid::Uuid,
    pub expiry_date: chrono::NaiveDate,
    pub user_id: uuid::Uuid,
}



```

3.0.5.9. Multiplayer game management

```rust
pub async fn ws_upgrader(
    ws: WebSocketUpgrade,
    State(state): State<Arc<AppState>>,
    Extension(user): Extension<Option<UserExt>>,
) -> Response {
    // required due to state not implementing copy
    let cloned_state = state.clone();
    ws.on_upgrade(move |socket| ws_handler(socket, cloned_state, user))
}

pub async fn ws_handler(ws: WebSocket, state: Arc<AppState>, user: Option<UserExt>) {
    state.game_manager.clone().assign_game(ws, user).await
}

use axum::extract::ws::{Message, WebSocket};
use futures::stream::SplitSink;
use futures::{SinkExt, StreamExt, TryFutureExt};
use sillyrng::{Gen, Xoshiro256plus};
use std::{collections::HashMap, sync::Arc};
use tokio::select;
use tokio::sync::{mpsc, Mutex};
use tokio::time::{interval, Duration};
use ulid::Ulid;

use crate::misc::Queue;
use crate::models::UserExt;
use crate::Move;

#[derive(Clone)]
pub struct GameManager {
    pub user_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>)>>>,
    pub anon_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>)>>>,
    pub cheater_games: Arc<Mutex<Queue<(ulid::Ulid, mpsc::Sender<WebSocket>)>>>,
}

pub struct Game {
```

```rust
    players: HashMap<Ulid, Player>,
    inactive_players: HashMap<Ulid, Player>,
    seed: u32,
    quota: u32,
}

#[derive(Clone, Debug)]
pub struct Player {
    grid: [bool; 16],
    b_coords: (u8, u8),
    r_coords: (u8, u8),
    current_score: u8,
    rng: sillyrng::Xoshiro256plus,
}

#[derive(serde::Serialize, serde::Deserialize, Debug)]
pub struct MMove {
    player_id: Ulid,
    action: Move,
}

#[derive(serde::Serialize, serde::Deserialize)]
#[serde(tag = "type", content = "data")]
pub enum WsMessage {
    Move(MMove),
    Quota { quota: u32, players_left: u32 },
    ID(Ulid),
    Start(u32),
    Out(u32),
    Win,
    Ping,
}

impl GameManager {
    pub async fn assign_game(&self, ws: WebSocket, user: Option<UserExt>) {
        println!("Attempting to assign player to a game");
        let games = match user {
            Some(u) => {
                //innocent until proven guilty, very demure, very fashionable
                if u.cheater.unwrap_or(false) {
                    self.cheater_games.clone()
                } else {
                    self.user_games.clone()
                }
            }
            None => self.anon_games.clone(),
        };
        let mut attempts = games.lock().await.size;
        let mut ws = ws;
        while attempts > 0 {
            let mut lock = games.lock().await;
            if let Some(game) = lock.dequeue() {
                match game.1.send(ws).await {
                    Ok(()) => {
                        lock.enqueue(game.clone());
                        return;
                    }
                    Err(mpsc::error::SendError(rws)) => {
```

```
                        ws = rws;
                        attempts -= 1;
                    }
                }
            } else {
                break;
            }
        }

        let (tx, rx) = mpsc::channel(40);
        let game_id = Ulid::new();
        tokio::spawn(game_handler(game_id.clone(), rx));

        match tx.send(ws).await {
            Ok(_) => {
                println!("Created new game with ID: {}", game_id);
                games.lock().await.enqueue((game_id, tx));
            }
            Err(e) => {
                println!("failed to send to game error: {}", e);
            }
        };
    }
}


async fn game_handler(id: Ulid, mut rx: mpsc::Receiver<WebSocket>) {
    println!("Game {} initialized, waiting for players", id);
    let mut state = Game {
        players: HashMap::new(),
        inactive_players: HashMap::new(),
        seed: rand::random::<u32>(),
        quota: 0,
    };
    let mut senders: HashMap<Ulid, SplitSink<WebSocket, Message>> = HashMap::new();
    let mut receivers = vec![];
    while state.players.len() <= 5 {
        match rx.recv().await {
            Some(mut p) => {
                let meow_id = Ulid::new();
                println!("Player {} joined game {}", meow_id, id);
                p.send(axum::extract::ws::Message::Text(
                    serde_json::to_string(&WsMessage::ID(meow_id)).unwrap(),
                ))
                .await
                .unwrap();
                state.players.insert(
                    meow_id.clone(),
                    Player {
                        grid: [false; 16],
                        b_coords: (0, 0),
                        r_coords: (3, 3),
                        current_score: (0),
                        rng: Xoshiro256plus::new(Some(3)),
                    },
                );
                let (sender, receiver) = p.split();
                senders.insert(meow_id, sender);
```

```
                receivers.push(receiver);
            }
            None => {}
        }
    }

    println!("Game {} starting with {} players", id, state.players.len());
    for i in state.players.iter_mut() {
        i.1.rng = Xoshiro256plus::new(Some(state.seed.clone() as u64));
        let mut count = 0;
        while count < 4 {
            let x: u8 = (i.1.rng.next() * 4 as f64).floor() as u8;
            let y: u8 = (i.1.rng.next() * 4 as f64).floor() as u8;
            if i.1.grid[(x * 4 + y) as usize] == false {
                i.1.grid[(x * 4 + y) as usize] = true;
                count += 1;
            }
        }
    }
    for (_p, i) in senders.iter_mut() {
        i.send(axum::extract::ws::Message::Text(
            serde_json::to_string(&WsMessage::Start(state.seed)).unwrap(),
        ))
        .await
        .unwrap();
    }

    println!("Game {} is now running", id);
    let mut interval = interval(Duration::from_secs(5));

    loop {
      // combines all websocket futures into one, allows for all to be awaited at once
        let websocket_futures = futures::future::select_all(
            receivers
                .iter_mut()
                .enumerate()
                .map(|(i, ws)| Box::pin(async move { (i, ws.next().await) })),
        );
        //tokio select macro, allows for multiple futures to be awaited at once, while not impacting performance, particularly efficient
        select! {
            (result, _, _) = websocket_futures => {
                let (idx, msg_result) = result;
                match msg_result {
                    Some(Ok(axum::extract::ws::Message::Text(text))) => {
                        match serde_json::from_str::<WsMessage>(&text) {
                            Ok(WsMessage::Move(mrrp)) => {
                                println!("Received move from socket {}: {:?}", idx, mrrp);
                                let player = state.players.get_mut(&mrrp.player_id);
                                match player {
                                    Some(p) => {
                                        match mrrp.action {
                                            Move::CursorRedUp => p.r_coords.1 = (p.r_coords.1 as i8 - 1).max(0) as u8,
                                            // 3 should be constant, but multilpayer is only 4x4
                                            Move::CursorRedDown => p.r_coords.1 = (p.r_coords.1 + 1).min(3),
```

```
                                        Move::CursorRedLeft => p.r_coords.0 = (p.r_coords.0 as i8 - 1).max(0) as u8,
                                        Move::CursorRedRight => p.r_coords.0 = (p.r_coords.0 + 1).min(3),
                                        Move::CursorBlueUp => p.b_coords.1 = (p.b_coords.1 as i8 - 1).max(0) as u8,
                                        Move::CursorBlueDown => p.b_coords.1 = (p.b_coords.1 + 1).min(3),
                                        Move::CursorBlueLeft => p.b_coords.0 = (p.b_coords.0 as i8 - 1).max(0) as u8,
                                        Move::CursorBlueRight => p.b_coords.0 = (p.b_coords.0 + 1).min(3),
                                        Move::Submit => {
                                            dbg!(p.grid);
                                            dbg!(p.r_coords,p.b_coords);
                                            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) {
                                                println!("successfull submission");
                                                p.current_score += 1;
                                                let mut count = 0;
                                                let r = p.r_coords.0 * 4 + p.r_coords.1;
                                                let b = p.b_coords.0 * 4 + p.b_coords.1;
                                                while count < 2 {
                                                    let x: u8 = (p.rng.next() * 4 as f64).floor() as u8;
                                                    let y: u8 = (p.rng.next() * 4 as f64).floor() as u8;
                                                    if !p.grid[(x * 4 + y) as usize]
                                                        && (x * 4 + y != r || x * 4 + y != b)
                                                    {
                                                        p.grid[(x * 4 + y) as usize] = true;
                                                        count += 1;
                                                    }
                                                }
                                                p.grid[r as usize] = false;
                                                p.grid[b as usize] = false;
                                            } else {
                                            p.current_score = 0;
                                        }
                                    },
                                }
                                }
                                None => {
                                    println!("Recieved message from invalid player");
                                }
                            }

                        }
                        Ok(_) => {
                            println!("Received non-move message from socket {}", idx);
                        }
                        Err(e) => {
                            println!("Error parsing message from socket {}: {}", idx, e);
                        }
                    }
                }
                None => {
                    println!("Socket {} closed for game {}", idx, id);
```

```
                        let _ = receivers.remove(idx);
                    }
                    _ => continue,
                }
            }
            // if 5s interval is up, update quota
            _ = interval.tick() => {
                println!("New quota for game {}", id);
                let mut culled_players = vec![];
                let  player_count = state.players.len();
                for (i, p) in state.players.iter_mut() {
                    dbg!(p.current_score);
                    if (p.current_score as u32) < state.quota {
                        let position = (player_count - culled_players.len()) as u32;

                        if let Err(e) = senders.get_mut(&i.clone()).unwrap()
                            .send(axum::extract::ws::Message::Text(
                                serde_json::to_string(&WsMessage::Out(position))
                                    .expect("Failed to serialize Out message")
                            )).await
                        {
                            println!("Failed to send message to player, removing {}: {}", i, e);

                            continue;
                        }
                        culled_players.push(i.clone());
                        state.inactive_players.insert(i.clone(), p.clone());
                        senders.remove(&i);
                    }
                    p.current_score = 0;
                }
                for i in culled_players {
                    state.players.remove(&i);
                }
                if state.players.len() <= 1 {
                    println!("Game {} ended - {} player(s) remaining", id, state.players.len());
                    for (i,_) in state.players.iter() {
                        // add to database when that gets done

senders.get_mut(&i.clone()).unwrap().send(axum::extract::ws::Message::Text(serde_json::to_string(&W
                    }
                    break;
                }

                state.quota += 1;
                for (_i,sender) in &mut senders {
                    sender.send(axum::extract::ws::Message::Text(
                        serde_json::to_string(&WsMessage::Quota {
                            quota: state.quota,
                            players_left: state.players.len() as u32,
                        })
                        .unwrap(),
                    ))
                    .await
                    .unwrap();
                }
            }
        }
```

```
    }
    println!("Game {} has ended", id);
}



```

3.0.5.10. WASM

```rust
pub trait Gen {
    type NumberType;
    fn new(seed: Option<u64>) -> Self;
    fn next(&mut self) -> Self::NumberType;
    fn sigmoid(x: f64) -> f64 {
        1.0 / (1.0 + (-x).exp())
    }
}

pub struct SplitMix {
    seed: u64,
}

impl Gen for SplitMix {
    type NumberType = u64;
    fn new(seed: Option<u64>) -> Self {
        SplitMix {
            seed: seed.unwrap(),
        }
    }
    /// based on https://xoshiro.di.unimi.it/splitmix64.c and rand_xoshiro
    fn next(&mut self) -> u64 {
        self.seed = self.seed.wrapping_add(0x9e3779b97f4a7c15);
        let mut z: u64 = self.seed;
        z = (z ^ (z >> 30)).wrapping_mul(0xbf58476d1ce4e5b9);
        z = (z ^ (z >> 27)).wrapping_mul(0x94d049bb133111eb);
        z ^ (z >> 31)
    }
}

#[derive(Debug, Clone)]
pub struct Xoshiro256plus {
    seed: [u64; 4],
}

impl Gen for Xoshiro256plus {
    type NumberType = f64;

    fn new(seed: Option<u64>) -> Self {
        let mut rng = SplitMix::new(seed);
        Xoshiro256plus {
            seed: [rng.next(), rng.next(), rng.next(), rng.next()],
        }
    }
    fn next(&mut self) -> Self::NumberType {
        let result = self.seed[0].wrapping_add(self.seed[3]);
        let t = self.seed[1] << 17;
```

```rust

        self.seed[2] ^= self.seed[0];
        self.seed[3] ^= self.seed[1];
        self.seed[1] ^= self.seed[2];
        self.seed[0] ^= self.seed[3];

        self.seed[2] ^= t;
        self.seed[3] = Xoshiro256plus::rol64(self.seed[3], 45);

        (result >> 11) as f64 * (1.0 / (1u64 << 53) as f64)
    }
}

impl Xoshiro256plus {
    pub fn rol64(x: u64, k: i32) -> u64 {
        (x << k) | (x >> (64 - k))
    }
    pub fn get_seed(&self) -> String {
        format!("{:?}", self.seed)
    }
}



```

js

```js
import * as wasm from "./xoshiro_wasm_bg.wasm";
export * from "./xoshiro_wasm_bg.js";
import { __wbg_set_wasm } from "./xoshiro_wasm_bg.js";
__wbg_set_wasm(wasm);
wasm.__wbindgen_start();

/* tslint:disable */
/* eslint-disable */
export const memory: WebAssembly.Memory;
export const sigmoid: (a: number) => number;
export const __wbg_splitmix_free: (a: number, b: number) => void;
export const splitmix_new: (a: number, b: bigint) => number;
export const splitmix_next: (a: number) => bigint;
export const __wbg_xoshiro256plus_free: (a: number, b: number) => void;
export const xoshiro256plus_new: (a: number, b: bigint) => number;
export const xoshiro256plus_next: (a: number) => number;
export const xoshiro256plus_get_seed: (a: number) => [number, number];
export const __wbindgen_export_0: WebAssembly.Table;
export const __wbindgen_free: (a: number, b: number, c: number) => void;
export const __wbindgen_start: () => void;

/* tslint:disable */
/* eslint-disable */
export function sigmoid(x: number): number;
export class SplitMix {
  free(): void;
  constructor(seed?: bigint);
  next(): bigint;
}
export class Xoshiro256plus {
  free(): void;
  constructor(seed?: bigint);
```

```
  next(): number;
  get_seed(): string;
}

```

3.0.5.11. Settings Component

svlt

```svlt
<script lang="ts">
  import { getContext } from 'svelte';
  let meow = 0;
  export let showModal: boolean;
  export let closeModal: any;
  let dialog: any;
  let idx: any;
  let state: any = getContext('state');
  let keycodes: any;

  $: keycodes = $state.keycodes;
  const reset = () => {
    $state = JSON.parse(
      JSON.stringify({
        gameMode: 'timer',
        timeLimit: 30,
        keycodes: {
          wU: 'w',
          wD: 's',
          wL: 'a',
          wR: 'd',
          aU: 'ArrowUp',
          aD: 'ArrowDown',
          aL: 'ArrowLeft',
          aR: 'ArrowRight',
          submit: ' ',
          reset: 'r'
        },
        size: 4,
        das: 133,
        dasDelay: 150
      })
    );
    meow += 1;
  };
  const getChar = (i: any) => {
    let char: any;
    switch (i) {
      case '0':
        char = keycodes.wU;
        break;
      case '1':
        char = keycodes.aU;
        break;
      case '00':
        char = keycodes.wL;
        break;
      case '01':
        char = keycodes.wD;
        break;
```

```
      case '02':
        char = keycodes.wR;
        break;
      case '10':
        char = keycodes.aL;
        break;
      case '11':
        char = keycodes.aD;
        break;
      case '12':
        char = keycodes.aR;
        break;
      case '20':
        char = keycodes.submit;
        break;
      case '21':
        char = keycodes.reset;
        break;
    }
    switch (char) {
      case 'ArrowUp':
        char = '↑';
        break;
      case 'ArrowDown':
        char = '↓';
        break;
      case 'ArrowLeft':
        char = '←';
        break;
      case 'ArrowRight':
        char = '→';
        break;
    }
    return char;
  };

  const keyClick = (i: any) => {
    idx = i;
    setTimeout(() => {
      window.addEventListener('keydown', setChar, { once: true });
    }, 0);
  };
  const setChar = (e: any) => {
    switch (idx) {
      case '0':
        $state.keycodes.wU = e.key;
        break;
      case '1':
        $state.keycodes.aU = e.key;
        break;
      case '00':
        $state.keycodes.wL = e.key;
        break;
      case '01':
        $state.keycodes.wD = e.key;
        break;
      case '02':
        $state.keycodes.wR = e.key;
```

```
        break;
      case '10':
        $state.keycodes.aL = e.key;
        break;
      case '11':
        $state.keycodes.aD = e.key;
        break;
      case '12':
        $state.keycodes.aR = e.key;
        break;
      case '20':
        $state.keycodes.submit = e.key;
        break;
      case '21':
        $state.keycodes.reset = e.key;
        break;
    }
    let doc: any = document.getElementById(idx);
    let char = e.key;
    switch (char) {
      case 'ArrowUp':
        char = '↑';
        break;
      case 'ArrowDown':
        char = '↓';
        break;
      case 'ArrowLeft':
        char = '←';
        break;
      case 'ArrowRight':
        char = '→';
        break;
    }
    doc.textContent = char;
    idx = 69420;
  };

  $: if (dialog && showModal) dialog.showModal();
</script>

<dialog
  bind:this={dialog}
  on:close={closeModal}
  class="h-screen w-screen bg-crust/0 flex items-center justify-center {showModal ? '' : 'hidden'}"
>
  {#key meow}
    <div class="flex flex-col bg-surface0 w-fit h-fit rounded-md">
      <div class="text-text text-3xl m-4 mb-0">settings</div>
      <div class="text-xl text-text mb-0 m-4">movement:</div>
      <div class="flex flex-row m-4">
        {#each Array(2) as _, x}
          <div class="flex flex-col items-center mx-4">
            <!-- svelte-ignore a11y-click-events-have-key-events -->
            <!-- svelte-ignore a11y-no-static-element-interactions -->
            <div
              id={x.toString()}
```

```
              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 ==
              x.toString()
                ? 'bg-green'
                : 'bg-text'}"
              on:click={() => keyClick(x.toString())}
            >
              {getChar(x.toString())}
            </div>
            <div class="flex flex-row">
              {#each Array(3) as _, y}
                <!-- svelte-ignore a11y-click-events-have-key-events -->
                <!-- svelte-ignore a11y-no-static-element-interactions -->
                <div
                  id={x.toString() + y.toString()}
                  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 ==
                  x.toString() + y.toString()
                    ? 'bg-green'
                    : 'bg-text'}"
                  on:click={() => keyClick(x.toString() + y.toString())}
                >
                  {getChar(x.toString() + y.toString())}
                </div>
              {/each}
            </div>
          </div>
        {/each}
      </div>
      <div class="text-xl text-text mb-0 m-4">place:</div>
      <!-- svelte-ignore a11y-click-events-have-key-events -->
      <!-- svelte-ignore a11y-no-static-element-interactions -->
      <div
        id={'20'}
        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 ==
        '20'
          ? 'bg-green'
          : 'bg-text'}"
        on:click={() => keyClick('20')}
      >
        {getChar('20')}
      </div>
      <div class="text-xl text-text mb-0 m-4">reset:</div>
      <!-- svelte-ignore a11y-click-events-have-key-events -->
      <!-- svelte-ignore a11y-no-static-element-interactions -->
      <div
        id={'21'}
        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 ==
        '21'
          ? 'bg-green'
          : 'bg-text'}"
        on:click={() => keyClick('21')}
      >
        {getChar('21')}
      </div>
      <div class="text-xl text-text mb-0 m-4">auto repeat rate:</div>
```

```
      <div class="flex flex-row text-text text-xl mx-8">
        <div class="w-8">
          {$state.das}
        </div>
        <input class="mx-4 w-64" type="range" min="0" max="1000" step="1" bind:value={$state.das} />
      </div>
      <div class="text-xl text-text mb-0 m-4">delayed auto shift:</div>
      <div class=" flex flex-row text-text text-xl mx-8">
        <div class="w-8">
          {$state.dasDelay}
        </div>
        <input
          class="mx-4 w-64"
          type="range"
          min="0"
          max="1000"
          step="1"
          bind:value={$state.dasDelay}
        />
      </div>
      <div class="flex flex-row self-center m-4">
        <button class="text-crust bg-red rounded-md w-16 h-8 mx-2 hover:scale-105" on:click={reset}
          >reset</button
        >
        <button
          class="text-crust bg-blue rounded-md w-16 h-8 mx-2 hover:scale-105"
          on:click={() => dialog.close()}>exit</button
        >
      </div>
    </div>
  {/key}
</dialog>

```

3.0.5.12. Layout and Styling

svlt

```
<script lang="ts">
  import '../app.css';
  import studio from '$lib/assets/studio.png';
  import Modal from '$lib/settings.svelte';
  import Trophy from 'svelte-material-icons/Trophy.svelte';
  import AccountCircle from 'svelte-material-icons/AccountCircle.svelte';
  import Settings from 'svelte-material-icons/Cog.svelte';
  import { onMount, setContext } from 'svelte';
  import { browser } from '$app/environment';
  import { writable } from 'svelte/store';
  import Information from 'svelte-material-icons/Information.svelte';
  import { redirect } from '@sveltejs/kit';
  import { goto } from '$app/navigation';
  const FLAVOUR = 'mocha';
  let showModal = false;
  let showWelcome = false;
  let selectedElement: { focus: () => void; };
  //TODO custom bg
  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) => {
```

```
    selectedElement = e.currentTarget;
    showModal = true;
  }

  const closeModal = () => {
    showModal = false;
    if (selectedElement) {
      selectedElement.focus();
    }
  }


</script>


<main class={FLAVOUR}>
  <div class="flex flex-col justify-between h-full max-h-screen min-w-screen font-mono">
    <div class="flex flex-row bg-base justify-between h-fit w-full items-center">
      <a class="flex flex-row text-4xl text-rosewater p-2" href="/">
        <x class="text-blue">Double</x> <x class="text-mauve font-bold">TAPP</x>
      </a>
      <div class="flex flex-row">
        <button on:click={() => showWelcome = true}>
          <Information color="#cdd6f4" class="h-12 w-12 p-2" />
        </button>
        <button on:click={openModal}>
          <Settings color="#cdd6f4" class="h-12 w-12 p-2" />
        </button>
        <button on:click={() => goto('/leaderboards')}>
          <Trophy color="#cdd6f4" class="h-12 w-12 p-2" />
        </button>
        <button on:click={() => goto('/signup')}>
          <AccountCircle color="#cdd6f4" class="h-12 w-12 p-2" />
        </button>
      </div>
    </div>
    <div class="bg-base h-screen">
      <slot></slot>
    </div>
    <div class="flex flex-row bg-base justify-between h-24 w-full items-center">
      <div class="flex flex-row items-center opacity-50">
        <a href="https://studiosquared.co.uk">
          <img class=" m-4 h-10" src={studio} alt="[S]^2" />
        </a>
      </div>
    </div>
  </div>

  {#if showWelcome}
    <div class="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center">
      <div class="bg-base p-6 rounded-lg max-w-md">
        <h2 class="text-2xl text-rosewater mb-4">Welcome to DoubleTAPP</h2>
        <p class="text-text mb-4">
          In DoubleTAPP, your aim is to move both your cursors onto different active tiles to score points. (WASD and arrow keys as default controls)
        </p>
        <p class="text-text mb-4">
```

```
          You get a point for each correct move, and lose all your points if you place your cursors incorrectly, good luck!
        </p>
        <p class="text-text mb-4">
          you can customize your controls and other settings in the settings menu.
        </p>
        <button
          class="bg-blue text-base px-4 py-2 rounded"
          on:click={() => closeWelcome(true)}
        >
          Got it!
        </button>
      </div>
    </div>
  {/if}

  <Modal bind:showModal />
</main>


```

3.1. 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-5.mp4
Test session expiry after timeout	Pass	7 day expiry, functional

Test grid initialization with correct size (4x4)	Pass	shown in all other tests
Test grid initialization with correct size (5x5)	Pass	
Test grid initialization with correct size (6x6)	Pass	
Test initial cursor positions (blue at 0,0 and red at size-1,size-1)	Pass	, test-10.mp4
Test initial grid has exactly 'size' active tiles	Pass	

Test blue cursor movement in all directions with keyboard	Pass	test-12.mp4
Test red cursor movement in all directions with keyboard	Pass	test-13.mp4
Test cursor movement boundary limits (cannot move outside grid)	Pass	test-14.mp4
Test DAS (Delayed Auto Shift) functionality for cursor movement	Pass	test-15.mp4
Test valid submission when both cursors are on active tiles	Pass	test-16.mp4
Test invalid submission when cursors are on the same tile	Pass	test-17.mp4
Test invalid submission when one cursor is not on an active tile	Pass	shown in other tests
Test score increment on valid submission	Pass	test-19.mp4
Test score reset on invalid submission	Pass	test-20.mp4
Test visual feedback (green) for correct submissions	Pass	test-21.mp4
Test visual feedback (red) for incorrect submissions	Pass	test-22.mp4
Test new active tiles appear after valid submission	Pass	test-23.mp4
Test deactivation of submitted tiles after valid submission	Pass	test-25.mp4
Test timer countdown functionality	Pass	shown in other tests

Test game end when timer reaches zero	Pass	
Test game statistics display after game end	Pass	test-28.mp4
Test leaderboard display with correct pagination	Pass	test-29.mp4
Test leaderboard filtering by grid size	Pass	test-30.mp4
Test leaderboard filtering by time limit	Pass	test-31.mp4
Test leaderboard filtering for personal bests	Pass	test-32.mp4
Test multiplayer game joining functionality	Pass	test-33.mp4
Test multiplayer game quota system	Pass	shown in other tests
Test multiplayer game player elimination	Pass	shown in test 33-35 .mp4
Test multiplayer game final rankings	Pass	test-35.mp4
Test WebSocket connection establishment	Pass	shown in other tests
Test WebSocket message handling for different action types	Pass	test-37.mp4
Test server-side move verification with valid moves	Pass	shown in other tests

Test server-side move verification with invalid moves	Pass	shown in other tests
Test server-side timing verification for normal play	Pass	test-39.mp4
Test server-side timing verification for suspicious patterns	Pass	test-40.mp4
Test server-side path optimization detection	Pass	shown in test-40
Test PRNG (Xoshiro256+) deterministic output with same seed	Pass	test-42.mp4
Test game state persistence in database	Pass	tested manually
Test user statistics update after game completion	Pass	tested manually
Test keybind customization persistence	Pass	test-48.mp4
Test settings reset to defaults	Pass	test-49.mp4
Test game performance with rapid inputs	Pass	test-50.mp4
Test game performance with simultaneous inputs	Pass	test-51.mp4

4. Evaluation

4.0.1. Evaluation Against Objectives

Objective	Evaluation	Evidence
User can interact with the grid	Implemented ✓	Grid component manages cursor movements with keyboard controls and displays active tiles with visual feedback.
User can move both cursors using keyboard	Implemented ✓	`onKeyDown` event handler implements movement for both cursors with configurable controls.

User can “submit” moves using keybind	Implemented ✓	`submit` function processes move validation when the submit keybind is pressed.
User can reset game via keybind	Implemented ✓	Reset functionality implemented via the `reset` keybind in `onKeyDown` handler.
User can change gamemode and settings	Implemented ✓	UI includes dropdown selectors for gamemode, grid size, and time limit with reactivity.
User can modify keybinds, DAS and ARR	Implemented ✓	Settings component allows full customization of keyboard controls and timing parameters.
Cursors are visually distinct	Implemented ✓	Blue cursor (top-left borders) and red cursor (bottom-right borders) are visually distinct.
User can view game statistics	Implemented ✓	End-game screen displays score, position, and game parameters.
Leaderboard functionality	Implemented ✓	Leaderboard component with pagination and filtering is fully implemented.
Login/authentication system	Implemented ✓	Complete authentication flow with signup, login, and session management.

Objective	Evaluation	Evidence
User authentication & management	Implemented ✓	Secure authentication with password hashing (bcrypt) and session management.
Database schema	Implemented ✓	Comprehensive relational database schema with user, statistics, and game tables.
Game verification	Implemented ✓	Server-side move validation with comprehensive checking for legitimate gameplay.
Anti-cheat measures	Partially Implemented ⚠	Move timing analysis is implemented, but advanced anti-cheat verification is incomplete.
Multiplayer implementation	Implemented ✓	WebSocket-based real-time multiplayer with quota system and elimination mechanics.
Performance optimizations	Implemented ✓	Efficient data structures (HashMap, Queue) and high-

		performance libraries (Axum, Tokio).

4.0.2. User Feedback

4.0.2.1. Client Feedback

- “the gameplay is surprisingly engaging

the leaderboard system is nice and offers a unique gameplay incentive the user experience was very well executed and intuitive“

UI : 5/5 **Functionality** : 4/5 **UX**: 4/5

4.0.2.2. Test User Feedback

overall users have enjoyed the game, and found it to be quite challenging at first, but with practice they were able to improve. users have particularly enjoyed the multiplayer system, and the challenge of trying to beat their friends.

“I like the lay out and the colour scheme. initiative to use and edit the game format”
“The app was smooth and everything worked well. I found the usage of two hands to be a unique challenge, but the game was explained well and the UI was intuitive and easy to understand.”
“The game’s awesome!! It’s so simple but so fun to play. It didn’t take long to get a hang of it and it works great, would definitely give it a 10/10 and recommend it to others.”

4.0.3. Future Improvements

In the future, i plan to improve the game by adding mobile support, and having more in depth analytics on the users profile page. additionally i would like to implement more stringent verification, such as a obfuscated library that detects html injections and other methods for securing.

4.0.4. Conclusion

the project has been mostly a success, and has been a great learning experience, i have learned a lot about web development, and have a better understanding of the complexities required to create truly random numbers, and securing a web application.

5. Bibliography

[1] Wikipedia contributors, “Standard Deviation.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Standard_deviation

[2] Steve Klabnik and Carol Nichols, *The Rust Programming Language*. Accessed: Dec. 01, 2024. [Online]. Available: https://doc.rust-lang.org/book/

[3] Postgres Team, “Postgres Documentation.” Accessed: Dec. 01, 2024. [Online]. Available: https://www.postgresql.org/docs/

[4] David Blackman and Sebastiano Vigna, “xoshiro / xoroshiro generators and the PRNG shootout.” Accessed: Dec. 15, 2024. [Online]. Available: https://prng.di.unimi.it/

[5] Wikipedia contributors, “Merge Sort.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Merge_sort

[6] Hard Drop - Tetris Wiki, “Delayed Auto Shift - Understanding Modern Tetris Controls.” Accessed: Dec. 17, 2024. [Online]. Available: https://harddrop.com/wiki/DAS

[7] GeeksforGeeks, “Circular Queue.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-circular-queue/

[8] Rust Language Team, “Asynchronous Programming in Rust.” Accessed: Dec. 02, 2024. [Online]. Available: https://rust-lang.github.io/async-book/

[9] MDN Web Docs, “HTTP - Overview of the Hypertext Transfer Protocol.” Accessed: Dec. 20, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[10] IETF - Network Working Group, “User Datagram Protocol (UDP) - Internet Protocol Suite.” Accessed: Dec. 20, 2024. [Online]. Available: https://www.ietf.org/rfc/rfc768.txt

[11] Mozilla Developer Network, “An Introduction to WebSockets.” Accessed: Dec. 10, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[12] IETF - Network Working Group, “ChaCha20.” Accessed: Feb. 14, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7539

[13] Wikipedia contributors, “ChaCha20.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/ChaCha20

[14] Wikipedia contributors, “Xorshift.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Xorshift#xorshift*

[15] Rosetta Code contributors, “Linear Congruential Generators.” Accessed: Feb. 14, 2025. [Online]. Available: https://rosettacode.org/wiki/Linear_congruential_generator

[16] Wikipedia contributors, “Linear Congruential Generators.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Linear_congruential_generator

[17] Wikipedia contributors, “Mersenne Twister.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Mersenne_Twister

[18] Wikipedia contributors, “Permuted Congruential Generator.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Permuted_congruential_generator

[19] Wikipedia contributors, “ARC4.” Accessed: Feb. 14, 2025. [Online]. Available: https://en.wikipedia.org/wiki/RC4

[20] david bau, “Seedrandom.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.npmjs.com/package/seedrandom

[21] Svelte Team, “SvelteKit Documentation.” Accessed: Dec. 19, 2024. [Online]. Available: https://kit.svelte.dev/docs/introduction

[22] Tailwind Labs, “Tailwind CSS Documentation.” Accessed: Dec. 05, 2024. [Online]. Available: https://tailwindcss.com/docs

[23] Catppuccin Team, “Catppuccin Color Scheme.” Accessed: Dec. 08, 2024. [Online]. Available: https://github.com/catppuccin/catppuccin

[24] Wikipedia contributors, “Sigmoid function.” Accessed: Dec. 14, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Sigmoid_function

[25] Joshua Glazer and Sanjay Madhav, *Multiplayer Game Programming*. Accessed: Dec. 12, 2024. [Online]. Available: https://www.informit.com/store/multiplayer-game-programming-architecting-networked-9780134034300