Game Design Document

Game Overview

- **Genre:** Falling block puzzle game (Tetris clone)
- **Target Audience:** Casual players, puzzle enthusiasts, retro gaming fans
- **Core Mechanics:** Rotate and drop tetrominoes to form complete lines. Completed lines are cleared, awarding points. Game speed increases as more pieces are placed.

Gameplay Description

- **Rules:** Pieces fall from the top. Player can move them left, right, rotate, or drop faster.
- **Objectives:** Maximize score by clearing as many lines as possible.
- **Win Condition:** None, game is endless.
- **Lose Condition:** Game ends when pieces stack to the top of the board.

Technical Specifications

- **Screen Size:** Runs in terminal window. Uses dynamic sizing. Default board is 10x20.
- **Input Controls:**
    - Arrow Left/Right → Move piece
    - Arrow Up → Rotate
    - Arrow Down → Soft drop
    - Spacebar → Hard drop
    - P → Pause

    - S → Save
    - L → Load
    - Q → Quit
- **Data Structures:**
    - `Vec<Option<Color>>` for the board grid
    - `Piece` struct for tetromino definitions
    - `ActivePiece` struct for current piece position/rotation

Game Objects

- **Pieces (Tetrominoes):** Seven unique shapes, each with color and rotations.
- **Board:** Grid storing placed blocks.
- **Game State:** Tracks active piece, next piece, score, speed, and status (paused, over).

Game Flow

1. Start screen with title and instructions
2. Gameplay loop: falling pieces, player input, score updates
3. End screen with final score and high score tracking

Variables and Data Structures

- `board: Vec<Option<Color>>` – Current placed pieces
- `active_piece: ActivePiece` – Currently falling tetromino
- `next_piece_id: usize` – Randomly generated next piece

- `score: u32` – Player score
- `high_score: u32` – Stored across sessions
- `gravity_delay: Duration` – Controls fall speed

Function Architecture

- **Game::new()** – Initializes board and state
- **Game::update()** – Advances game logic (falling pieces, collision check)
- **Game::render()** – Draws board and UI to terminal
- **Game::try_move() / try_rotate()** – Handles player input
- **Game::lock_piece()** – Fixes piece into board and checks for line clears
- **Game::save_game() / load_game()** – JSON-based save/load
- **show_start_screen() / show_end_screen()** – Handles pre/post-game screens

Development Timeline

- **Phase 1:** Implement board, tetromino definitions, piece movement
- **Phase 2:** Add scoring, line clearing, game loop
- **Phase 3:** Implement start and end screens
- **Phase 4:** Add save/load functionality
- **Phase 5:** Add high score tracking and polish

Code Documentation

Inline Comments

- Logic-heavy parts such as rotation, collision, and save/load include explanations.
- Example: Wall kick offsets in `try_rotate()` are documented for clarity.

Function Documentation

- Each core function describes **purpose**, **parameters**, and **return values**.
    - Example: `fn save_game(&self) -> io::Result<()>` → Saves current state as JSON file.

Module Organization

- **main.rs** (single file project) contains:
    - Argument parsing
    - Piece/board definitions
    - Game struct and methods
    - High score handling
    - Start/end screen functions
    - `main()` entry point

README.md

Installation

```
git clone https://github.com/Khantdotcom/Mad_tris.git

cd Mad_tris

cargo build --release
```

Running the Game

```
cargo run --release
```

Controls

- Arrow keys → Move / Rotate / Drop
- Space → Hard drop
- P → Pause
- S → Save
- L → Load
- Q → Quit

Save/Load

- Game saves to `tetris_save.json`
- High score saved in `highscore.txt`

Code Style

- Follows Rust 2021 edition defaults
- Consistent indentation (4 spaces)
- Snake_case for functions and variables
- UpperCamelCase for structs