Tetra Tangle: A 2.5-Dimensional Adaptation From a Familiar Retro Block Puzzle Game

Michael Scornavacca
COS B.S.E, Class of 2024

COS 426
Princeton University
Princeton, New Jersey

***Abstract*—This paper outlines the development of a block puzzle game completely compliant with the annually released Guideline from the Tetris Company [1]. This includes but is not limited to, frame-perfect controls, the Super Rotation System (SRS), and the extended placement lockdown mechanism. The engine is built in Typescript, solely off of the three.js library, while the user interface overlay is created via React. Other features include fully customizable controls, a public leaderboard, and visual enhancements such as selective bloom. The project is composed of 2,796 lines of code and is thoroughly documented and well-structured, making it suitable as a template for others. An academic overview of the project's design philosophy and technical achievements is presented in this paper.**

Word count—2159

***Index Terms*—tetris, three.js, computer graphics, react, typescript**

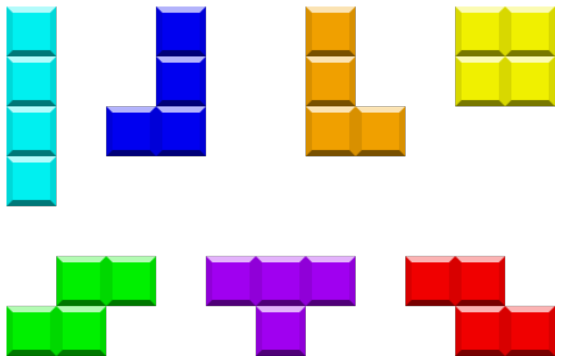


Fig. 1. All seven tetrominoes in a classic game of Tetris [3]

I. Introduction

A. Background

Tetris, one of the most successful and recognizable video games of all time, was created in 1984 by Alexey Pajitnov [2]. Its name comes from the rudimentary combination of the Greek prefix for four, ”tetra”, and Pajitnov's favorite sport, tennis [2]. Its simple yet innovative design quickly traveled across the world and resulted in the captivation of players worldwide.

B. Basic Premise

For those unfamiliar, the objective of Tetris is to place falling ”tetrominoes”, which are shapes formed with only four squares (hence the term ”tetra”), and create lines with no gaps across a ten by twenty playing field. These possible tetrominoes can be seen in figure 1. Once a line with no gaps is formed, it is cleared, and the entire playing field shifts down by a single unit. Clearing multiple lines, or clearing them in special ways, warrants more points and can be used to earn higher and higher scores. The player loses once there is no space at the top of the playing field for another tetromino to spawn in.

C. Previous Work and Aim

There are enumerable official Tetris games released at the time of this project, all with their own quirks and twists.

However, for web-based, browser versions of Tetris, the two most notable are jstris and tetr.io, which are both 2D implementations. There has yet to be a clean and presentable 2.5D implementation of Guideline Tetris implemented for the web that looks similar to the Tetris Effect.

The goal is to create a version of Tetris that, by abiding by the Tetris Guideline, would theoretically be endorsed by the Tetris Company if presented[1]. This includes implementing controls, movement, and scoring as a modern game of Tetris would. Success would simply be making a version of Tetris that feels indistinguishable (if not better) than modern official Tetris games, and looks as such.

II. System/Methodology

The project is broken into three essential sections: (1) the engine, (2) the actual game and its logic, and (3) the overlay user interface.

A. Engine

The engine is essential for rendering scenes, updating entities and input, and providing a simple yet extensible base for all game logic and interactions. The engine present in this project has been adapted from a popular GitHub repository named `simple-threejs-typescript-starter` and modified to suit the needs of this project [4].

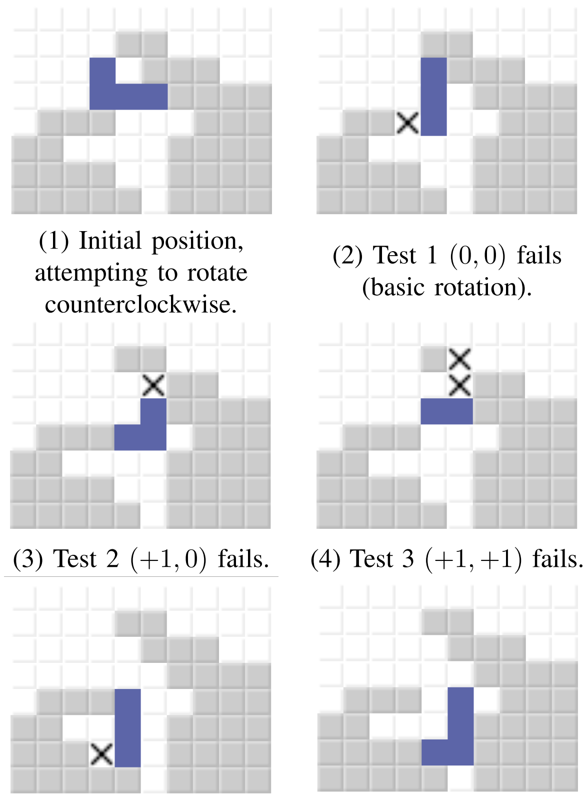
[1]Also, assuming that I had the budget, marketing team, and recognition to actually be endorsed

The heart of this system is the `Engine` class, which instantiates and manages all other components. For example, this includes the render loop, scene instantiation, and frame updates.

An object that implements the `Entity` interface can receive ”updates” every frame (a call to the `update` function with the time since the last frame). This is a very similar structure to how Unity3D handles their `Update()` function with `MonoBehaviours` [5]. However, these objects need to manually receive this call from the engine, and this can get hectic and difficult to manage when more and more entities are created.

Thus, the `Scene` and `GameEntity` classes were created. `Scene` extends from the three.js `Scene` class and implements the `Entity` interface, and manages a list of entities. In `Scene`'s update function, the update function is called on each member of the list of entities. `Scene`'s update function is managed and called by the root engine. `GameEntity` takes a `Scene` and automatically registers it in the constructor, thus guaranteeing that it receives updates if its scene is active. This creates a managed, tree-like structure, where entities within a scene are managed by a `Scene` rather than the root engine.

The other aspects of the engine are rather simplistic and self-explanatory, such as the `RenderLoop`, `RenderEngine`, and `Camera` files.



(1) Initial position, attempting to rotate counterclockwise.

(2) Test 1 $(0, 0)$ fails (basic rotation).

(3) Test 2 $(+1, 0)$ fails.

(4) Test 3 $(+1, +1)$ fails.

(5) Test 4 $(0, -2)$ fails.

(6) Final position, test 5 $(+1, -2)$ succeeds.

Fig. 2. Demonstrates a J tetromino rotating counterclockwise from neutral position utilizing the super-rotation system [6]

B. Game

The game itself is based on the latest Tetris Guideline and enforces the same exact rules and functionality as a modern Tetris game. This includes the super-rotation system (SRS), frame-accurate movement, and ”move-reset” auto-locking.

1) **Super Rotation System (SRS)** - Players can rotate falling tetrominoes to better fit them into the play area. This works perfectly, except when the already placed tetrominoes block the potential cells that a rotated tetromino would occupy. In retro Tetris, the rotation would simply fail, and the tetromino would not move; however, this is not the case in modern Tetris. One of the essential portions of the Tetris Guideline is the rotation system, named the Super Rotation System, or SRS for short.

   If a tetromino cannot rotate via basic rotation, the game tries *5* different positional offsets in order to place the rotated tetromino (6 including basic rotation). These tests are attempted in order until one succeeds, at which point the rotation *and* the positional offset are applied to the falling tetromino. There is a different set of offsets for each type of rotation, one example being shown in 2.

   SRS allows for a *much* more complicated and intricate set of gameplay mechanics, most importantly including T-spins.

2) **Frame-Accurate Movement** - There are three values essential for movement within any modern game of Tetris, all represented in milliseconds: (1) delayed auto-shift (DAS), (2) auto-repeat rate (ARR), and (3) soft drop (SD). DAS is the amount of time a movement key must be held down (left or right) before a falling tetromino begins to automatically ”slide” across the play area (or ”auto-shift”). Once DAS is activated, ARR determines how fast the tetromino ”slides,” representing the amount of time in between each automatic movement. SD determines the speed at which a player can speed up the rate a tetromino falls, only activating if the level speed is not already at this pace.

   These values are all handled within the `update()` function in the `Tetromino` class. However, since different machines will run the game at different framerates, it is pertinent to keep these timings accurate. If a value for DAS, ARR, or SD is less than the time in between frames, it could be the case that the falling tetromino might need to move *twice or more* in a single frame. By keeping track of these values over each frame, these can ensure frame-accurate movement, which is essential for any modern Tetris game.

   Most modern Tetris games endorsed by the Tetris Company do not allow for customization of DAS, ARR, or SD. However, I think it's useful to expose these parameters since they enhance gameplay and increase customizability, so they are configurable in the settings page of the UI.

3) **Extended Placement Lockdown** - In retro Tetris, tetrominoes would lock automatically upon no longer being

able to fall downwards due to either another placed tetromino or the bottom of the play area. This is called *Classic Lockdown*, where tetrominoes would lock down after 0.5 seconds of not falling any further. This form of lockdown is rather unforgiving and can be frustrating. There were two remedies proposed by the Tetris Guideline: (1) Infinite Placement Lockdown and (2) Extended Placement Lockdown. Infinite Placement Lockdown would reset this 0.5-second timer upon any rotation or movement, essentially allowing the piece to never be placed if it was free to move or rotate. This can cause issues, especially in multiplayer titles of Tetris. Hence, Extended Placement Lockdown limits this reset to 15 moves/rotations per tetromino and is what is widely used by almost all modern Tetris games, including this one.

C. *User interface (UI)*

The user interface was built solely with React and overlayed on top of the `<canvas>` element that rendered the game scene. This might've been slightly more overhead than necessary for the purpose of this project; however, the development speed and structure of React made it essential, given the short timeframe. Using React allowed the project to have additional features, such as customizable controls and a leaderboard, that may have taken too much time with plain HTML/CSS.

In addition to React, Recoil, a global state management library, was used in order to update the interface from within the logic of the game. The state of the app is organized into "atoms" which are essentially globally accessible variables that trigger React component rerenders upon being changed. Different atoms were maintained for the app screen, score, level, and lines cleared.

III. Results

The final version of the game contained roughly 2,800 lines of code (excluding the template and boilerplate code used) and consisted of:

1) A fully featured Tetris game that abides by the Guideline released by the company once a year. This includes frame-perfect controls, the Super Rotation System (or SRS), and extended placement lockdown for pieces.
2) A React-based user interface that is overlayed over three.js and interacts with game mechanics via a global state library called Recoil.
3) Scoring system that supports most types across all Tetris games. This includes all types of line clears, t-spins, and t-spin minis. The only scoring types that aren't implemented are "back-to-back" and "combos."
4) A level system that makes the game more difficult as the player clears more and more lines. This is represented by UI and a glowing bar that expands in size as the player clears more lines.
5) A set of fully customizable controls where every key/button is customizable from the settings and the movement specification (ARR, DAS, and SD).
6) Post-processing effects, mainly centered around selective bloom for falling and placed tetrominoes. A screen capture of the glowing tetromino (and some of the fading ones that are already placed) can be seen in figure 3.

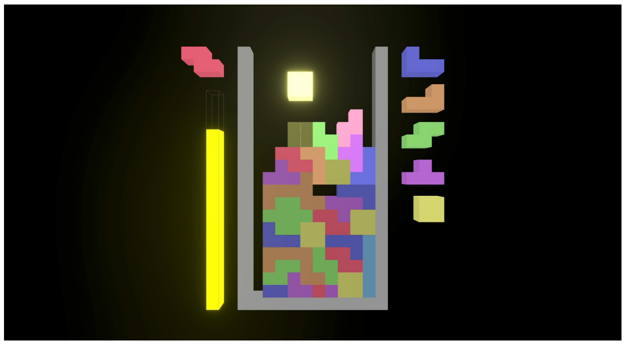
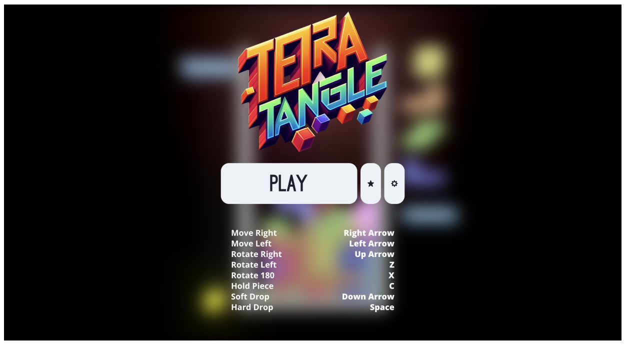


Fig. 3. A screen capture of the middle of a game, without the score and level counters.

7) A leaderboard system that allows users to enter their name and upload their best score after completing a game. This shows the top-ten scores of all time from the game so players can compete against each other.
8) A menu that has a logo (courtesy of DALL-E) and quick access to the settings/leaderboard. It shows a reference to the current controls and how to play. It features a blurred background of some footage of me playing the game. This can be seen in figure 4.



Move Right	Right Arrow
Move Left	Left Arrow
Rotate Right	Up Arrow
Rotate Left	Z
Rotate 180	X
Hold Piece	C
Soft Drop	Down Arrow
Hard Drop	Space

Fig. 4. A screen capture of the menu screen, including potential navigation to the settings and leaderboard

9) A customized game engine that allows for simple creation of scenes that maintain the status of children objects and can easily swap between different scenes without overlapping meshes/entities.
10) An organized project structure that is well-commented, and simple to understand for others to build off of.

IV. Conclusion

Overall, *this is Tetris*. The goal for this project was directly attained: create a Tetris game that indistinguishably feels like an official title endorsed by the Tetris company. The Tetris Guideline was explicitly followed and implemented exactly according to specification, with no shortcuts taken.

There are a couple of areas of improvement, mostly visual, that could be worth investigating given more time:

1) **Line Clear Animations** - Currently, completed lines are cleared instantly, and the board immediately shifts down to accommodate the gap. This is how my personal favorite websites handle line clears, however, popular Tetris titles, such as Tetris Effect, usually have fancy animations that emphasize the importance of clearing lines and advancing in level.
2) **Feel and Juice [7]** - Game "feel" and "juice" is an elusive aspect of any game, that immediately arises when a player starts interacting with the controls. It involves the screen shaking when something momentous occurs, or time slowing down when a player has a close call. All of these seemingly tiny features culminate into significantly more satisfying gameplay.
3) **Reactive Background** - The background, currently, is completely black. This allows for the player to focus on the pieces, and limits distractions, but is mildly boring. Implementing a background that reacts with what is happening within the game, or to music playing, could add some interesting visuals.

If it isn't clear by this point, Tetris is my all-time favorite game. I grew up playing it, and have always wanted to implement a complete version of my own at some point in time. This project, albeit a technical achievement of mine, is a personal tribute to a classic game that has shaped my love for computer science.

References

[1] 2009. URL: https://archive.org/download/2009-tetris-variant-concepts_202201/2009%20Tetris%20Design%20Guideline.pdf.

[2] Dan Ackerman. *The Tetris effect: the game that hypnotized the world*. Public Affairs, 2016.

[3] Wikipedia. *Tetris — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Tetris&oldid=1188517352. [Online; accessed 12-December-2023]. 2023.

[4] Maya Nedeljković Batić. *simple-threejs-typescript-starter*. https://github.com/mayacoda/simple-threejs-typescript-starter. 2023.

[5] Unity Technologies. *MonoBehaviour*. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. [Online; accessed 12-December-2023]. 2023.

[6] URL: https://tetris.wiki/Super_Rotation_System.

[7] Game Maker's Toolkit. *Secrets of Game Feel and Juice*. Feb. 2017. URL: https://www.youtube.com/watch?v=216_5nu4aVQ.