TETRIS - final project report

Adam Novocký, Monika Pinteková, Dušan Podmanický

Introduction

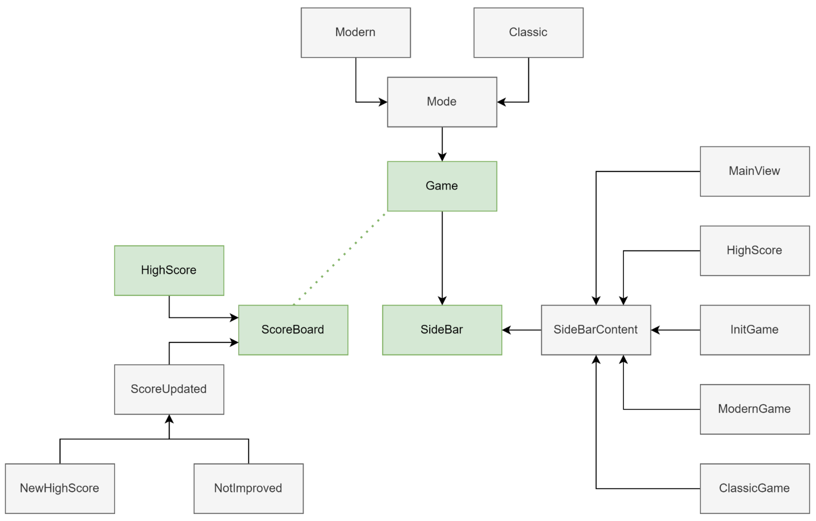
As our final project, we created an application of a classic game of Tetris. We believe that Tetris is a great project for beginners in Rust as it offers a fun and engaging way to learn game development while also exploring the capabilities of the Rust programming language. Building Tetris in Rust will help us learn several key features of the Rust programming language, including ownership and borrowing, error handling, traits and generics.

Requirements

The project fulfills the following requirements:

- single player Tetris game
- multiple levels with varying rising difficulty
- starting level selection
- different game modes: classic and modern
- scoring and high score leaderboard
- background music

Design diagram



Design choices

In the main.rs file we initialise our raylib window and audio device, after which we render the Game and Sidebar components inside the window. The game is initialised based on the Game Mode setting in the Sidebar. The game spawns a random Tetris shape that keeps falling periodically until a collision is detected either with the bottom of the playing grid or other shapes. When a whole row of the grid is occupied with shapes, the row is cleared and the user score is incremented.

Sidebar acts like the menu of the game, which allows for game mode change or a high score leaderboard, which is also saved into a .txt file.

We based our design choices on classic tetris implementations. We have decided to use Raylib as we already have some experience in utilising this library. We were also curious about how Rust operates with non-Rust code and whether some direct interaction with C code would be needed on our part. In the end, we did not need to make use of concurrency or music playing directly, as everything was handled by Raylib itself.
We have also used a basic .txt file for serialising high scores. If we had split highscores depending on gamemodes, or had it been more complex, we could have used a serialisation library such as serde. This could have also been used for key bindings saving and customisation, which we did not have time to get into in the end.

Dependencies
The main dependencies for our project include:

- raylib
  A simple and easy-to-use game development library for C and other languages including Rust. It provided us with a comprehensive set of functions for handling graphics, audio, input, and many more features that were essential for the project.
- std::ffi
  A module in the Rust standard library that provides support for working with C and other foreign languages. Because the Raylib library is written in C, we needed to interact with functions and data structures defined in C, such as handling strings, pointers, and other low-level concepts.
- std::fs::File
  A module in the Rust standard library that provides an interface for working with files - opening and closing files, reading and writing data. We used this for handling the scoreboard data.
- std::io
  Another set of modules in the Rust standard library that provided us with various types for performing input and output operations while handling the scoreboard data.
- rand::Rng
  A rand crate that provides a trait called Rng that defines methods for generating random numbers. We implemented this for spawning random tetromino pieces.

Evaluation

The result of the final project is a playable and in our opinion enjoyable game with all the requirements we wrote out in the project proposal. Overall, implementing a tetris game in Rust was a positive experience. Rust's strict type system and ownership rules helped us catch many potential bugs early in the development process. Additionally, the language's focus on performance and memory safety allowed for efficient and stable code. Using the Raylib game engine also greatly simplified the task of handling graphics and user input.
While Rust's strict type system was helpful in some cases, it also made certain tasks more challenging, particularly when working with raylib that required conversion between Rust's and C's data types. Additionally, Rust's ownership rules, while powerful, can be difficult to fully grasp, especially when dealing with more complex data structures. Overall, we think that Rust's learning

curve can be in some cases steeper than other languages, and this made the initial stages of our project more challenging.
Compared to other languages, implementing a bigger project in Rust was an overall positive experience, though it required more upfront investment in learning the language and its unique features.