Temtris

Curt Lynch and Hunter Johnson

Abstract

Temtris is a Tetris clone we are developing as part of our CSCI 352 class.

1. Introduction

Temtris is a WPF clone of Tetris with our own spin added in. Tetris, is a game where 4 block section shapes fall from top down and you try to fit rows of these blocks together at an ever increasing pace until the screen inevitably fills causing game over. A score is computed based on how efficiently you clear the board as the game progresses, with more points being awarded for streaks of row clears and tetris row clears (4 simultaneous row clears). Our target audience is anyone looking to pass time with a casual game of Tetris. We are making Temtris as a duo as part of our CSCI 352 class. That being the case, we still want Temtris to be a fun and interesting game for anyone looking to play.

1.1. Background

A Tetrimino is one of the 7 classic Tetris shapes; these include the O, I, T, L, J, S, and Z shapes. Each Tetrimino is made up of four squares called minos. The matrix is the space where the Tetris game is played. A 'line clear' happens when a horizontal row is completely filled, causing it to be removed from the matrix. Lock-down is when a Tetrimino is locked into its current position and is no longer controllable.

While attempting to brainstorm ideas for our CSCI 352 class, the idea of a Tetris clone come up (with the awful misspelling Temtris). Eventually we settled on the idea and the name stuck.

1.2. Impacts

Tetris is a classic game that has had a major impact on the gaming community since its debut. Continuing to iterate upon Tetris and bring it back into the community repeatedly will only deepen its legacy. Temtris is a great way to do this while improving our skills and preparing us for working in teams and on larger projects in the future.

1.3. Challenges

This is the first moderately sized project either of us has attempted. This is also our first real foray into WPF and C# so there are some growing pains there as well. As for the project itself, we suspect getting the movement and rotations to feel like a proper Tetris game will take much more fine tuning than we have needed to use in the past. Rotating the pieces will likely also pose some sort of challenge.

2. Scope

Our initial goal is to implement the following typical parts of a Tetris game:

- Implement a main menu.
- Have controllable tetriminos (2x1 shapes initially) that fall into place in the matrix.
- Have rows that disappear when they fill up, i.e., the line clears.
- Keep score or track how well the player does in some way.
- Display the next piece to spawn.

For stretch goals, we will to add the following:

- Implement all 7 tetrimino variants.
- Ability to swap active tetrimino with the next tetrimino during gameplay.
- Implement a multiplayer mode. Having at least two clients with a competitive game mode over LAN or just sharing a keyboard.
- Different power-ups may be earned by removing a certain amount of one color from the board. Perhaps board clear (of that color) or maybe have the next five pieces all be 'L' shapes.
- Play background music and have sound effects for line clears, player actions, etc.
- Have random shapes form and fall into the matrix. (instead of the standard Tetris shapes)

2.1. Requirements

These requirements are things we feel make Tetris what it is. Without them the Temtris would ultimately not be a Tetris clone.

2.1.1. Functional.

- Users can control Tetriminos (rotate, move left or right).
- Tetriminos are created and fall from the top of the matrix.
- Row clears should happen as the rows are filled.
- Main menu allows the user to start the game.

2.1.2. Non-Functional.

- Temtris should display controls needed to play the game.
- User interface should be attractive and inviting.

2.2. Use Cases

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Start game	Player	Easy	1
2	Move tetriminos left or right	Player	Easy	1
3	Lower tetrimonios into position	Player	Med	2
4	Rotate tetriminos	Player	Hard	3

TABLE 1. USE CASE TABLE

Use Case Number: 1
Use Case Name: Start game
Description: A player wishes to pass some time by playing Temptris. They start the game and click on the ”Start Game” button from the main menu.

1) The Player launches the application.
2) The Player left-clicks on ”Start Game” button.
3) The game screen appears and the game begins.

Termination Outcome: A game of Temtris is now running.

Use Case Number: 2
Use Case Name: Move tetriminos left or right
Description: A game of Temtris has been started and the player wishes to move the falling piece to the left or right. The player presses the left or right arrow on the keyboard and the active tetrimino moves accordingly.

1) The player presses the left arrow key on the keyboard.
2) The active tetrimino shifts one position to the left.
3) Player presses the right arrow key on the keyboard.
4) The active tetrimino shifts one block to the right.

Termination Outcome: The player has control over the left and right motion of the active tetrimino.

Use Case Number: 3
Use Case Name: Lower tetriminos into position
Description: The player is comfortable with the current trajectory of the active tetrimino and wishes to lower it into place. The player presses the down arrow key on the keyboard and the active tetrimino moves down into place.

1) The player presses the down arrow key on the keyboard.
2) The active tetrimino begins to fall faster.
3) The active tetrimino stops falling when it reaches inactive tetriminos and is set into place.

Termination Outcome: The player can speed up the falling of the active tetrimino.

Use Case Number: 4
Use Case Name: Rotate tetriminos
Description: The player wishes to rotate a tetrimino and presses the up arrow key on the keyboard. The active tetrimino rotates ninety degrees allowing the player to better control how it falls.

1) The player presses the up arrow key on the keyboard.
2) The active tetrimino rotates 90 degrees clockwise.
3) The player presses the up arrow key on the keyboard.
4) The active tetrimino rotates an additional 90 degrees clockwise.

Termination Outcome: The player has control over the rotation of the active tetrimino.

2.3. Interface Mockups

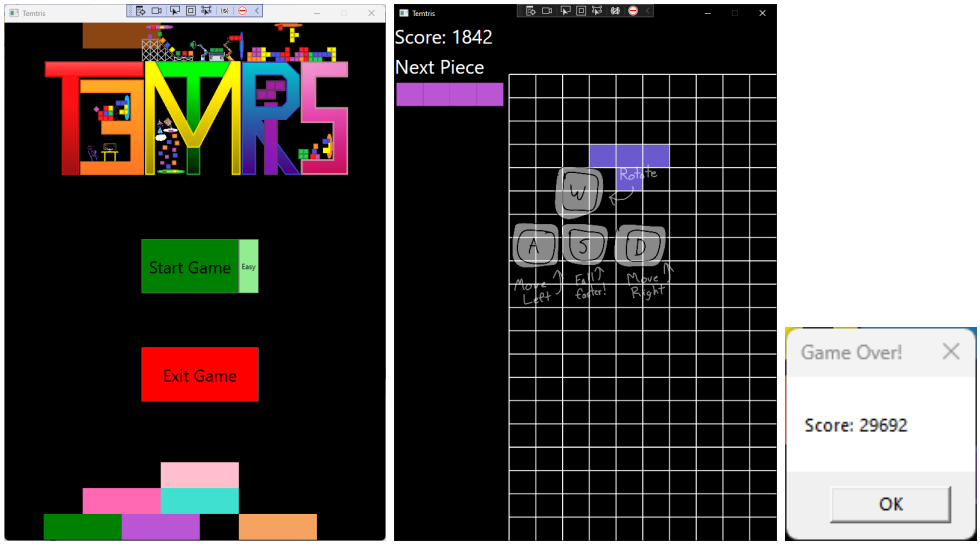

Figure 1. Temtris at the main menu(left), during a game(center), and the game over pop-up(right).

Figure 1 shows various screenshots from Temtris as the user plays the game. The left image, showing the main menu, allows the user to start a game, as in use case 1, or exit the application. The center image shows the game after a game has started and is where use cases 2, 3, and 4 take place. The image on the right is an example of what is displayed when the player finishes a game.

3. Project Timeline

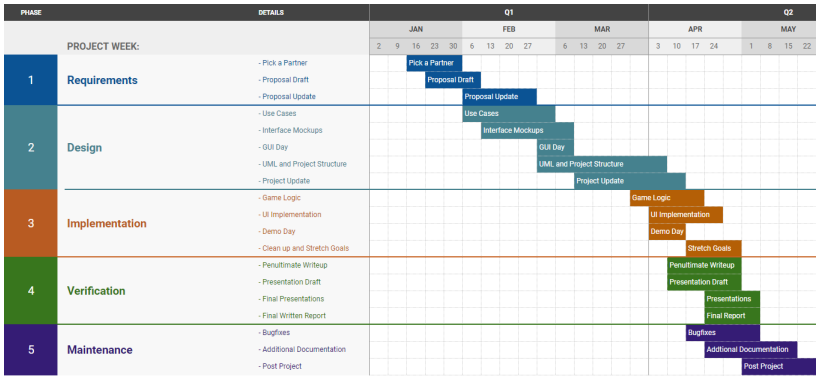

Figure 2. Project Timeline

Figure 2 shows the Temtris project timeline. As with most projects following the waterfall development cycle we wanted to solidify most of the design as early possible. This unfortunately left us with little time to actually implement the project and work out any rough edges in our design. If we had the chance to do this again we would've spent more time early learning WPF before attempting to outline a project using it.

4. Project Structure

The UI exists within the auto generated MainWindow class, which calls upon an instantiation of the TemtrisGame class to handle all of the game logic and keyboard input.

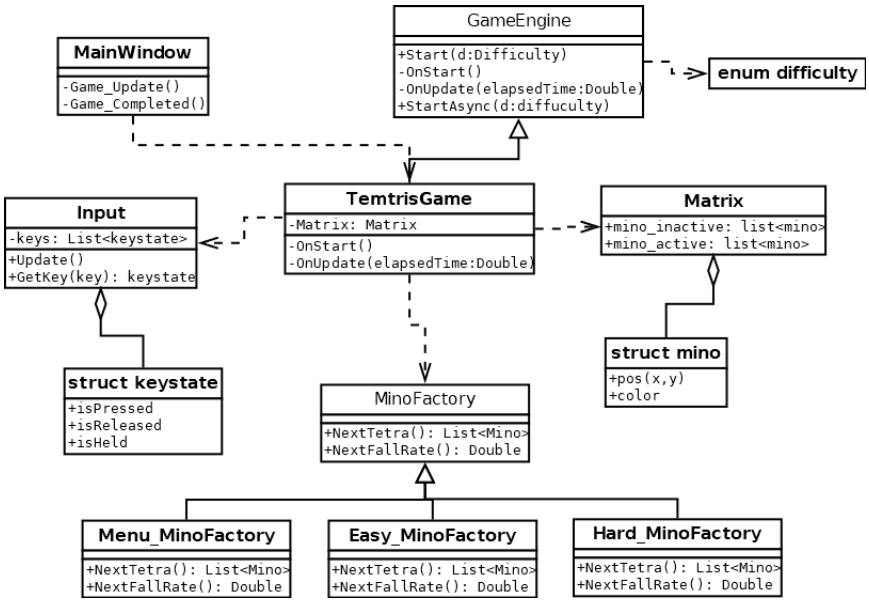

Figure 3. UML Diagram for Temtris

4.1. UML Outline

Figure 3 shows the breakdown of the class hierarchy for Temtris. MainWindow is the auto generated class given when starting a WPF project in Visual Studio. This is the home of our UI and, for our purposes, the entry point of the application. MainWindow utilizes TemtrisGame to get the layout and position of all tetriminos. GameEngine handles some of the more mundane non-Tetris aspects of the game engine, i.e., timekeeping, pushing updates to the UI, etc. TemtrisGame handles all of the game logic, i.e., tetrimino movement, row clearing, checking for user input, etc. TemtrisGame also houses the Matrix, MinoFactory, and KeyboardInput objects. KeyboardInput is a simple class that keeps track of user input for TemtrisGame. It gives us a bit more flexibility by allowing us to choose when it gets updated. Matrix is really just a data container for the score and all of the minos. Minofactory and its children are responsible for telling TemtrisGame how fast the tetriminos should fall and generating the tetriminos used.

4.2. Design Patterns Used

The GameEngine and TemtrisGame classes demonstrate the template design pattern. This lets us separate the game engine logic from the Temtris specific logic and potentially allows for the reuse of the GameEngine class in other projects.

The MinoFactory and subclasses demonstrate the factory method and strategy design patterns. The NextTetra method creates different types of tetraminos with different frequencies depending on the concrete class used. The NextFallRate method allows the concrete classes to determine how fast the tetraminos fall during gameplay.

5. Results

While not as polished as we would've liked, we managed to achieve our initial goals and implement the "all 7 tetrimino variants" stretch goal. All 4 of the use cases described above are also feasible in the current state of Temtris.

5.1. Future Work

Temtris is in a playable state and will likely not be receiving any further updates.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.