**Goals**

This project describes my process in recreating one of the most iconic video games ever made, Tetris! I specifically wanted to use class inheritance to reinforce what I learned in my java coding classes in school. To do this, I planned on making a general class for the blocks and a specific class for each individual block. I wanted the controls to be done through keyboard inputs by the player, and a start, pause, and game over screen to make the game more accessible. To be more faithful towards Tetris, a score, level, and lines cleared tab will be needed on the sides of the main grid. The game will also display a next block and held block feature. These will be used to give the player some help with the game. Finally, I wish to make the resolution a moderate to small size so that the game could be more accessible. This project will be done in Processing, which is a Java tool that helps to make drawing shapes and objects much easier.

**Process**

To keep track of everything in this project, I made a google slides list to document the different stages of my project. To see more of this directly, you can scroll down to my YouTube video below. What I talk about here will be an overview of that but more condensed. The first thing I did was draw out on my computer how I wished for the game to look to the user. To achieve this, I made my first class, the Menu class. This class would handle showing most of the game’s elements, except for the cells. It would display and keep track of the strings and integers for the lines cleared, score, and level values. I ended up settling on a dark gray theme for the general User Interface, since this would help make the Tetris blocks stand out more.

To make a usable grid for the actual Tetris blocks, I ended up making a Cell and Grid class. The cell class would keep track of the variables of each block on the grid. The Grid class would then take an x and y parameters in its constructor to make a 2d array of Cells. This helped model my layout of cells, since I could access this 2d array object inside of the grid class and use certain x and y indices to manipulate the cells. The cells were not constructed anywhere but inside of the grid class. After the grid class was constructed inside of the main program in setup(), it would loop to be shown to the UI inside of draw(). To specify, the setup() function is always ran once, while the draw() function loops infinitely unless it is directly stopped. This is helpful, since you can initiate objects once inside of setup() then show them and modify them in draw() for however long you need.

Now that the grid and cells were made, I could start designing the Block class and each specific Tetris block. Each separate Tetris piece inherited the block class and passed their color into their constructors. The specific blocks are named after the letter they represent by their shape, like ‘T’ or ‘I’. After succeeding in getting each block to be constructed and placed at the top of the grid, I worked on figuring out how to make the blocks fall and rotate. To do this, I created methods to detect if there were 'solid' cells around the 'active' cells. The solid cells were the borders and placed Tetris pieces, while the active cells were where the current Tetris block was occupying. This ended up making my code more complicated by the end of the project, but the final product still works well. A spawn zone was also added at the top of the grid, and is hidden from the player, so the blocks spawn right above the active grid. User input was also added at this stage with Processing's keyPressed() and keyReleased() functions. These helped to make it so you could get multiple inputs at once, instead of using their 'key' keyword.

Now that the blocks could be manipulated, I needed to add conditional checks to make the lines and score update when blocks were placed. I used two for loops to go through the grid array to check if all horizontal blocks in a row were solid. If so, another grid method would be called to clear that specific row, while also moving every row above it down one y position. This made it so the line would go away while also updating the other remaining solid cells. This is when I also implemented updating the score, lines, and level values inside of menu, since they could now also be properly updated. For every ten lines, the level goes up one number. Every level, the speed of the game gets faster, so the number of frames before the block drops gets lower. The score will increase based on your level and how many lines you clear at once. The next block and held block features were also implemented now. They would construct a new 4x2 grid each to display the blocks for their respective purposes. Hidden integer values were kept to keep track of these blocks outside of the grid. A helper function was defined inside of main to help randomly calculate the next block and swap the current and held block.

The final features that I added to this game were the different game states and menus, which include the start, pause, and game over screens. I made a switch case statement in the draw() loop in my main program and a integer variable called gameState to keep track of the current state. Constants were defined for the different states and user input or conditions would be used to swap them. The different states would be placed into their own methods to make readability higher for the switch case. This is why the main draw() loop is so short. Most of the games functionality comes from the statePlaying() helper function in Tetris.pde.

**Reflection**

Since I also made a game in processing during my first year in a group project, I really enjoyed tackling another one on my own. I felt more confident in designing the menus and manipulating objects than I did in my first game. Using inheritance for the Tetris blocks also helped me reinforce what I was learning in my java classes at the time around finals for the semester. Although, I did come across a few issues with this project. My cell class ended up being more complicated than I wished for, and it made manipulating every other method much harder than it had to be. If I was going to start again, I would have made the grid out of integer values in a 2d array instead of cell objects. This is because integers are primitive and it is easier to change them since they do not point to objects like the Cell class did. This made moving and swapping the cells much more complicated than it had to be. Yet, this did teach me to really think through how early design choices can impact the rest of a project. Once I got so far in the project, I couldn't really change anything in these classes since they were used so much. I would like to remake this game at some point after planning out how to set up the whole project up with pseudocode in the future. Overall, this was a great project that helped reinforce my java and coding skills. I am happy to say that this is my first project posted on my website, so I hope you enjoyed it! If you want to play my game, you can download the files on GitHub and open them with the processing java application. You can also play the game right under this text, since it is embedded on the website if you are on a desktop computer! Thank you for reading!