Project Description:

- Developed a playable LCD screen Tetris game using STM32CubeIDE with STM429xx HAL.
- Peripherals used: GPIO (button), NVIC (interrupts), I2C3 (touch interface), SPI5 (LCD screen), RNG (random number generator), and TIM6 (game tick).

Project Timeline:

1st step: The Initial Development of Blocks and LCD functions

- ➢ Block Representation:
  - Used 16-bit binary numbers to represent 4x4 bit grids (0 = empty, 1 = filled).
  - Utilized bit masking and bit shifting to simplify bit parsing in order to achieve relatively simple handling for drawing blocks.
- ➢ LCD Functions:
  - Enhanced functionality and ease of code by creating custom functions: draw vertical line, draw square, and display string.

2nd step: Block Movement Implementation

- ➢ Delete Block Function:
  - Iterated through 4x4 grids to erase block by redrawing cells in black to “visually erase”
  - Once the delete block and draw block were both working, I had a strong foundation for movement, rotation, and locking blocks.
- ➢ Block Movement: Integrated left, right, down, and rotation functions.
  - Relied on a TetrisBlockPropertiesTypeDef struct to manage block properties (rotation, color, coordinates, shape).

3rd step: Interrupt Handling

- ➢ Mapped the button press interrupts to block rotations and the touch interface interrupts to left/right movements.
- ➢ Developed the TIM6 to interrupt after 3 seconds, which allows the block to constantly “drop” down the screen.

4th step: More Advanced Game Logic

- ➢ Developed more complex functions; collision detection and clearing rows.
- ➢ Iteratively tested and debugged gameplay logic.

5th step: Final Features

- ➢ Fully functional Tetris game with:
  - Timer to track elapsed time.
  - Row-clear counter & game score/stats displayed on the end screen.
  - Aesthetically pleasing start and end screens.
  - Relatively smooth and bug-free gameplay.

Project Outcome

- Successfully implemented a fluid, playable Tetris game with relatively accurate game logic.

Work Breakdown:

- I first started this project by attempting to create the blocks. I had several options, a matrix, arrays, etc, but ended up choosing to build my blocks using 16 bit binary numbers. In this form, each block represents a 4x4 binary grid, where a 0 represents empty space, and a 1 represents a cell that is taken up by the block. This proved effective because I could easily parse each row (4 bits), and handle each row individually.

- This allowed me to then create my function to draw the block, which was relatively simple considering we were provided the LCD functions. With that being said, I did create a couple of LCD functions to streamline my code. Using the existing draw horizontal line, draw circle, and display character functions, I developed draw vertical line, draw square, and display string functions.

- Once I was able to draw and spawn a block, it became time to try and handle game movement. An essential part of block movement is my delete block function. This is because prior to any movement, I delete the block, then perform the move, then redraw the block to the screen. Delete block was complicated and time consuming to implement, but it laid out the foundation for most of my other functions moving forward. Essentially, I iterate through the 4x4 cell grid, find which cells are a part of the block shape, and then redraw them, filling them in with black to visually erase them. I was able to nearly replicate this same logic throughout other functions in my code. For example, locking a block in place also iterates through the block, finds what cells are taken up by the block shape, and then writes the block color to the game grid at the coordinates of the cells. From here, movement of the blocks became easy. Rotating a block is as simple as deleting it, incremententing the rotation integer by 1, then redrawing it. Left/Right/Down movement works the same way, all in adherence to the TetrisBlockPropertiesTypeDef struct which declares the rotation, color, coordinates, and shape of the block.

- Once able to move and rotate a block in the debugger (stepping through the code), I then shifted my focus to handling interrupts, specifically attaching a button press to rotate and screen press to movements. This proved to be easily the most frustrating and time consuming part of the project. I ran into several issues with my touchscreen, often finding my code hanging in a HAL error handler. After a few days of debugging, I was able to fix my issues by commenting out a few lines within the touch interface IRQ handler. At this point, I was able to move blocks and rotate blocks upon button press and touch interface press.

- Once game movement was fluid and playable, I focused on the game logic. The two most complex and difficult functions to code were the two in charge of detecting collisions and clearing out full tetris rows. I'd say that this was a little bit out of my project scope and I surprised myself when I got these functions working, after some help from TA’s in class and some peers' advice. Testing the game logic was a long process. I sat for hours playing the game, moving in unpredictable ways, rotating strangely, etc. all to

find potential bugs in the code. The issues were plentiful, some much harder than others to fix and troubleshoot. However, after a few days of consistent testing, while making small fixes and improvements to the code, I eventually arrived at a fluid, functional tetris game. The game tracks elapsed time, counts the number of row clears, and displays a nice aesthetically pleasing start and end screen. Overall, I am extremely happy with my final project and very pleased that I got it to be fully functional.

Testing Strategy:

- Throughout the project, I used a variety of methods to test the functionality of the game. Near the beginning of the project, I relied fully on the debugger to step through the code. I would step through, watch a block spawn, step through as it moves, and follow a block through its life span using only the step into or step over button. This worked well because it showed me where my issues were, allowing me to fix errors throughout the process of developing my code.

- Once I implemented the button/press interrupts and playable functionality, debugging the project became a lot harder. I typically had to just play around with the game, trying to purposefully break it in order to find issues. When I did come across a bug, I would try and replicate it. After repeatedly replicating the bug, I was typically able to find what the issue was. In some instances, it wasn't that easy. I occasionally had to play through the game, then pause the debugger right before I expected to encounter my error. Then I could step through the code while holding the button down to see its behavior.

- After countless run throughs of the game both by stepping through the debugger and by physically playing the game, I eventually fixed most, if not all bugs and ended with a functional project.

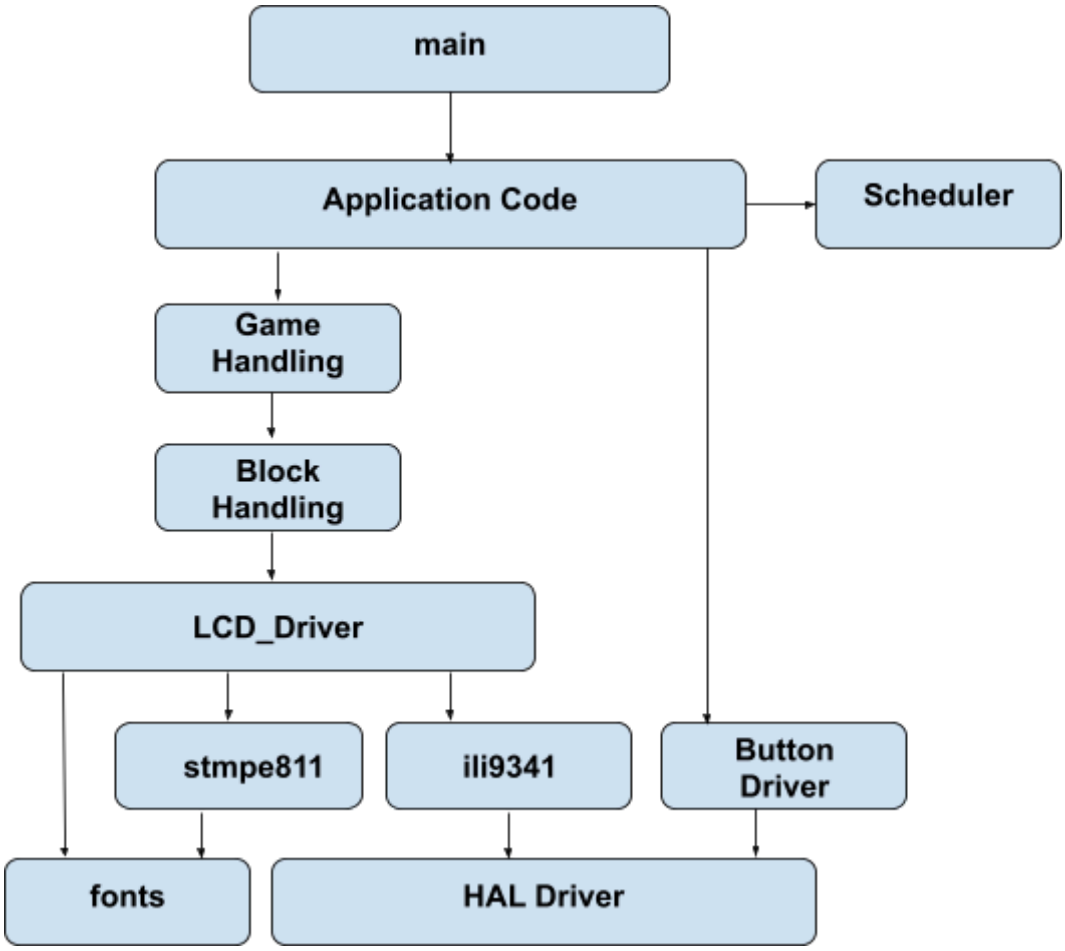
Use Case:

- The project is a fully functional tetris game, where a user button press rotates a block clockwise, and a left or right screen press results in a left or right move. Once a horizontal row (tetris line) fills up, it is automatically deleted from the screen, and added to the score. Upon completion of the game, an end screen is displayed with results; tetris clears and elapsed time.

Project Documentation:

Github Repository Link: https://github.com/coso5375/CSorrellLabFinal/tree/main/Core

Coding Hierarchy:



Code 《》 Block Handling:

- Blocks are 16 bit binary numbers, where each represents a 4x4 binary grid. A 0 represents empty space, and a 1 represents a cell that is taken up by the block.
- Within the BlockShape data type, a block has a color and a shape_rotation array.
- Within the BlockProperties data type, a block has x,y coordinates, a color, a rotation, cell size, and a shape type of type BlockShape.
- drawBlock(): Grabs the 16 bit shape, parses it to handle 4 bits (a row) at once, detects what cells the block shape should take up, and then draws to the LCD.
- deleteBlock(): Functions almost identically to drawBlock, but visually erases from the LCD by drawing with the color black.
- lockBlock(): Functions almost identical to function above, but finds which cells the block takes up and then permanently locks them in place by setting their color to the result of the uint16_t 2D array, gameGrid, at their respective cells.
- spawnBlock(): Initializes basic block parameters such as cell size, the starting (x,y) coordinates, and no initial rotation. Utilizes a random number generator to randomly spawn a block 1-7.
- detectCollision(): Pretends a movement goes through, whether rotation or a movement, and then finds if the cells taken up by the block would collide with existing blocks or grid boundaries.

Code《》Game Handling:

- Display_Start_Screen(): Displays a button that starts the game when pressed.
- Display_End_Screen(): Displays elapsed time and tetris stats/score.
- Game_Init(): Sets the screen to black, draws the tetris gridlines, initializes game stats to zero, and then spawns the first block in the top middle of the screen
- Gameplay(): Handles the game tick scheduled event. Moves block down every 3 seconds when there's no collision and checks for any full rows. If a collision is found, it locks the block in place and spawns a new one.
- Move functions: All work the same way; check for a collision, if collision is found, do nothing. Otherwise, delete the block off the screen, change the blocks coordinates or rotation in accordance to the movement, and then redraw the block.
- Check_Game_Over(): Constantly checks if any block lies in the row that is second to the top. If a block is locked in place in this row, or the row above, the game ends and displays the end screen.
- clearTetrisRows(): Iterates from the bottom row upward, checks for any full rows. If a full row is found, it iterates through the row, clears it, and overwrites its data with the row above it. Each row is cleared and then shifted down, until no full rows exist.

Struggles & Obstacles:

- My first big obstacle was my function for detecting block collisions. It was relatively simple to prevent blocks from moving outside the grid, but much more intuitive than I thought. Because my detect collision function worked in tandem with my lock block function, whenever a collision occurred below the current block, it would lock it in place. However, I ran into an issue with the left side of my game grid. Whenever a block touched the left edge, it would somehow detect a collision and lock the block, floating in space. After hours of trying to figure out what was causing this, I found that it was an extremely easy fix. My x and y coordinates were unsigned integers, so division and negative numbers destroyed the conditional logic. Upon switching them to just be 'ints', I no longer had the issue. This was one of those bugs that you are extremely relieved to fix, but at the same time angry at yourself for making a foolish mistake.

- A key struggle was getting the touch screen to work without the code hanging inside the HAL I2C3 error handler. This issue took everything I had, spending hours to try and find the root of the problem proved unsuccessful. Eventually with some help from Xavion, I was able to troubleshoot the issue by commenting out a few lines of code inside the provided IRQ15 Handler.

- Another obstacle was clearing full tetris rows. I approached this function with a plan; to iterate through each row, find if a row is full, delete it, and then move every row above it downward. However, it did not play out that easily. I ran into an unexpected issue, where if it did find a full row, it would delete it, but not preserve the rest of the game. The issue was that this function was getting in the way of my function that locks a block on the screen. It wasn't just as easy as moving every row down one row. Instead, I ended up having to initialize my game grid to be a 16 bit unsigned integer. This way, I could store each block color at their respective position within the grid. From here, I was able to copy rows and maintain blocks and their colors when moving every row downward. This function took a lot of planning, troubleshooting and thinking, but once successfully it was very satisfying to watch rows delete themselves.

- Git was a huge obstacle for me in this project. I've previously used git, so I didn't think much of it. However, it proved to be a huge pain. The git I have been accustomed to was on a windows computer, and with macOS my previous experience did not translate over. The git desktop app did not provide any help to me, as it could for some reason not connect to my remote repository. After a lot of research, I ended up just using the mac terminal to use git commands. I'm disappointed it took so long to figure out how to use it, but in retrospect I am extremely happy I did the research, because I now feel like a git professional. With no git software on my computer, I can successfully run git commands through my terminal and push and clone to and from my remote repository.

Project Takeaways:

I feel like I've learned a lot of valuable skills and lessons in programming throughout the duration of this project.

- Git: Git is an extremely powerful and useful tool for version control, and I feel much more prepared to code in the future now having a great sense of how to use it. Git was helpful for me many times during this project where I ended up breaking my code, but had a remote version to fall back onto. I made sure to commit changes to git whenever I had something working, and it proved to be an extreme life saver in times where my code had mysteriously stopped working or ran into issues.

- C programming: After this project, I feel like a much better programmer in C. I utilized a lot of structs, pointers, and other coding techniques in which I haven't been extremely familiar with, and in some cases haven't even touched since CSCI 2270. I feel that in this one project by itself, I have not only learned a lot of C, but also re familiarized myself with a lot of valuable coding skills. I had to make use of a lot of constant and external variables, another thing I was not too familiar with. I also put to use many of the concepts we've learned in this class, for example accessing my shape data by shifting bits and using a bitmask. This was definitely a challenge but allowed me to streamline my block & shape logic and at the same time reinforced some coding techniques we have not touched since October. Overall, because of how challenging this project was, I feel that I improved a lot as a programmer and feel more prepared and confident for computer science classes moving forward, even compared to how I felt two weeks ago.

- Overall, I learned many valuable lessons in project management when it comes to developing code. I found out that it's extremely important to prioritize important features, especially when working in a complex, multi-faceted project. I also realized the importance of testing code iteratively, and not developing hundreds of lines of code and testing when it's "done". Frequent testing, in combination with version control, allowed me to always have a somewhat stable version to fall back on if something went horribly wrong. This project also challenged my time management skills, which forced me to set specific goals day by day and stay on schedule. I feel that I completed the project relatively early compared to my peers, and I credit that to regularly tracking my progress and spending a little time every day developing a function or troubleshooting issues. It's clear to me that this project not only brushed up on my technical skills, but also gave me a better sense of how to approach a large project in a structured and thoughtful manner.

If I were to restart/Improve my project:

- If I were to do anything differently, I'd definitely change how I initially tackled the project. I think that at the beginning, I tried to do too many things at once, which often left me too confused to continue, or stressed and overwhelmed. I think that it would be smart to be more intentional with splitting the project into smaller, more manageable parts. For example, instead of working on my collision and lock blocking functions at the same time, I should have tackled them one at a time, and moved on to the other after the first was successful. This would have allowed me to remove some of the redundant code that can be seen in several of the block handling functions. This would also have made debugging easier, since I would've also been able to test out specific parts of the code without worrying about the project as a whole.

- Another change I'd make is trying to be more intentional about efficiency and performance. I most definitely have code that's not extremely optimized and probably some unnecessary lines. Moving forward, it's smart to address and think about memory management techniques to make the code run smoother, streamline some handling, and overall increase the efficiency of the game. It wasn't super essential for this project, but in the future, memory management and organization will be of extreme importance.

- Lastly, I think that if I restarted, I'd build test cases. At some points in this project it seemed impossible to find issues solely with the debugger. Creating test cases for some of the key functions would have helped catch some issues early and prevent major fixing later in the process. Test cases are also helpful in understanding the execution of the code, which would have helped me track the program flow in times where the debugging got super complex and hard to follow.

Overall, with a more structured approach, better debugging practices, and more attention to efficiency, the game would probably have been easier to develop and would definitely appear more polished and run smoother as well.