Tetris for the STM32F429I-DISC1 Development Board

GitHub Link: https://github.com/ArjunD112/ECEN2370-FinalProject

Project Description

This project is an implementation of the game *Tetris* on the STM32F429I Discovery Board. The program is written in C, and the game is played using the LCD/touch screen and the user button.

Project Scope and Timeline

This project uses the LCD touchscreen and the user button to play *Tetris*. We have been provided with file sets which easily facilitate communication with the LCD touchscreen. These file sets use the peripherals: GPIO, NVIC, I2C3, SPI5, and the LTDC. The GPIO, I2C3, and SPI5 are used to communicate with the LCD touchscreen, and the NVIC is used to allow the touchscreen to be interrupt driven. I will also integrate the user button, which will be communicated with using the GPIO peripheral and be interrupt driven using the NVIC. A timer (TIM7) will also be used to control the blocks' fall rate and keep track of elapse playing time. The RNG peripheral will be used to randomly select one of the 7 blocks in the gameplay screen. The game logic will be in the *Tetris_Logic* fileset. The overall project has:

- 3 screens on the LCD: the "start game" screen, the "gameplay" screen, and the "game over" screen
    - "Start Game" Screen: has a 'start' button and displays all 7 tetrominoes
    - "Gameplay" Screen: the actual gameplay
    - "Game Over" Screen: displays elapsed game time in seconds
- A 10x12 unit game field (10 units wide, 12 units tall)

- 7 unique rotatable tetrominoes (a button press is a 90-degree clockwise rotation)
- Touching the screen on the left/right side will shift the live tetromino left/right
- Tetrominoes will fall one level every second (approved by Xavion). 3 seconds per level felt slow.
- Rotating a tetromino that was flush with a wall or set tetromino utilizes the wall-kick mechanism, space permitting
- Levels cleared according to the mechanics of *Tetris*

The timeline was originally as follows:

- Week of 11/18: Configure and initialize all necessary peripherals
- Week of 11/25: 3 screens designed, screen navigation, and main gameplay logic
- Week of 12/02: Stress test all features, add extra credit feature
- Week of 12/09: Wrap up final testing and documentation revising

I did the majority of the work over the Fall Break, so the timeline ended up as follows:

- Week of 11/18: Understand *LCD_Driver* fileset and integration, initial planning
- Week of 11/25: 3 screens designed, screen navigation, and main gameplay logic
- Week of 12/02: Configure and integrate peripherals
- Week of 12/09: Clean up final version, write documentation

Each week involved testing and optimizing features as I implemented them.

Work Breakdown

The majority of the work took place in the *Tetris_Logic* fileset. There were many helper functions necessary for a smooth experience which covered all of the edge cases. The testing was intensive, because the edge cases that arose early in development would effectively “crash” the game (e.g. the current tetromino would phase through a previously set one in a very specific case, and the gameboard would be completely ruined). Testing was also difficult, since I began with the game logic without integrating the peripherals. I had to create various inefficient and highly specific scenarios to test general functionality and to find the edge cases.

I faced the most difficulty with integrating the peripherals. After I got one working in isolation, integrating functionality with the rest of the working code involved copious amounts of debugging and adjusting of other peripherals to make them all work at the same time. Even now, the touchscreen does not fully work, as the timer sometimes hangs upon a touchscreen input. Interestingly, if you add a couple more touchscreen inputs, things will work perfectly again. Debugging the problems with the hardware when I had integrated everything took a lot more time than I expected and gave rise to some very weird issues.

Testing Strategy

I adopted the Agile development framework. After each feature or critical function written, I ran a round of testing which ensured expected/desired behavior. This was done by writing short test functions and running the code, or by reusing old projects' implementations of similar features, depending on the code under test. This is unit-testing. I debugged as we have been, using the IDE's built-in debugging tools. Each week was about 40% development and 60% testing, and I didn't move onto the next feature until the current one was perfected. I originally tried to use the Waterfall method, but since I ultimately decided to start with the game logic, which has inherently highly inter-dependent functions, I switched to Agile.

Use Cases

The use cases for this application are:

- Playing *Tetris*
- Aiming of and launching ICBMs (Intercontinental Ballistic Missiles) at air and ground targets **(requires additional programming)**

Struggles and Obstacles

This board was built on a budget. Every sensor on it is cheap and the HAL needs some fine tuning. *Tetris* is a complicated program, and this board is not well-built to handle it. My implementation required various matrix operations, such as transposes and matrix multiplication. After I integrated all of the peripherals (button, LCD, timer, RNG, and touchscreen), the reliability of the program took a nosedive. I had to strip away protective layers, such as HAL status checking, to get the touchscreen to work even sometimes. Thankfully, I got all the other peripherals working reliably. The other major obstacle I faced was testing the game logic and operations without having the peripherals configured and implemented. In hindsight, I should have started with the peripherals first and then worked out the game logic. It took me a considerable amount of extra time in the beginning to devise ways to test my game logic.

What I Learned

I learned a lot about software development and the importance of planning. This project took me considerably longer to complete because I did not properly plan out the development process. As mentioned in the previous section, I developed the game logic first, and then began integrating the peripherals, which made testing the game logic as I developed difficult.

I also learned that hardware has actual limitations that aren't just theoretical. Up until this point, I've programmed in Swift (iOS language), Java, and Python. These are higher level languages in which one can basically assume the device has infinite memory. This is not the case in hardware, and I learned that the hard way.

What I Would Do Differently

I would not use this board or write this program in C. *Tetris* is a complicated program, and a developer would benefit from taking advantage of higher-level OOP languages. For example, in C, you cannot have a function which returns a 2D array, unlike in languages like Python. In order to get around that, I defined a new struct called *Board* which had only one member: a 2D array named *Field*. Then, the functions that I needed to return a 2D array will instead return an object of type *Board*.

I would also start by integrating the peripherals first, and then developing the game logic. This would make testing my game logic as I developed much easier and more efficient.

Improvements

The best way to improve this final project is to instead program *Tetris* using a higher-level OOP language and running it on a PC or laptop. However, given more time, I would add the ability to rotate a tetromino into a tight space, like modern versions of *Tetris*. Additionally, I would incorporate a score, which is added to with level clears and time elapsed. Currently, the touchscreen does not shift the tetrominoes left/right, it slides them. I would figure out why this is happening and rectify it. I think it would be interesting to instead use the onboard gyro sensor to shift the live tetromino left and right. Finally, there are a couple functions which can be combined into one, which would clean up the code.