Tetris 1v1 Battle

By: Isaac Rivera, Philippe Rerolle, Jason Li

1. Device Functionality

We created two separate devices that consist of a Teensy 3.2, LCD screen, radio, and wii nunchuk that were able to run a Tetris game. The game goes as any other tetris would run, but when lines were cleared on one screen, subsequent lines would be added to the bottom of the other player's screen. This makes the game a competition between both players until one loses. The controls are done through the wii nunchuk remote and the systems communicate using radios.

2. Hardware Components

The following hardware components will be used for this project:

(a) 2, 3.2-inch LCD displays: ($24,https://www.amazon.com/Display-Module-240320-4-Wire-Screen/dp/B07KPD4DHD/ref=sr_1_10?keywords=3.2+tft&qid=1584240565&sr=8-10)

(b) 2, Wii nunchuks (Wii, Nintendo, Used or $15 online)

(c) 2, Wireless Transmitters/Receivers: NRF24L01 Wireless Transceiver(ECE shop, $18 for both)

(d) 2, Teensy 3.2 boards ($22, https://www.amazon.com/PJRC-Teensy-3-2/dp/B07NWY44K6)

3. Design Timeline

(a) Week 5: Purchase components, draw out schematics, and start coding.

(b) Week 6: A failure on the side of the shipping company that was responsible for our previous LCD screen forced our work to potentially be pushed back for one week. We decided to continue working on schematics and code.

(c) Week 7: Having trusted the shipping company we anticipated the screen to come this week. Just in case we ordered another screen from amazon. We continued to code and make sure other parts, such as the controller and radio worked.

(d) Week 8: We have received the screen from amazon, gave up on previous shipment. We began coding to make sure the screen was capable of running visuals with our code. Pseudocode to draw/erase.

(e) Week 9: We began to finalize the controls and code of the game. Eventually we were able to complete one game set and rushed to finish the twin set by the end of the week.

(f) Week 10: We fixed bugs and made sure the game was completely functional. The game system was finished and they were able to compete against each other. In the final days of the week though, we broke one of the controllers, so at the moment, we do not have the system completely functional until we get the replacement controller. (Will be done by presentations).

4. Software Design

Tetromino pieces that have already been dropped, and are static on the screen, are represented in memory, as a two-dimensional array where each index can either be empty or have a tetromino.

The tetromino that the user controls (called “active tetromino” in the code), is stored as a struct containing its shape (I, J, L, S, T, O, Z), the coordinates of its center of rotation, and its current orientation, (“Normal”, “CounterClockwise”, “Clockwise”, “Flipped”).

A method called “blocks” takes as arguments the shape of the tetromino and its orientation, and returns the positions of the four blocks of the tetromino relative to its center of mass.

Using this method, paired with the coordinates for the center of rotation, the game calculates the actual coordinates of each block of the active tetromino, and determines if the tetromino is in a legal position (each block of the tetromino must be in a coordinate that is empty on the grid of static tetrominos).

Moving the tetromino left or right, as well as falling down one line, consists of changing the center of rotation of the active tetromino in the corresponding direction and verifying that the active tetromino is still in a legal position.

If a call to “game_fall”, returns false, then that means the tetromino cannot fall anymore as it would put it in an illegal position. As such, the next tetromino in the up next queue becomes the active tetromino, and the coordinates of the active tetromino become marked as filled.

To determine the next tetrominoes, a bag style randomizer is used, in keeping with other Tetris games.

In the “loop” method, the teensy constantly polls the Wii controller for the current states of its inputs. If the x-value for the analog controller is below a certain threshold, “game_move_left” gets called moving the active tetromino, left. If the value is above a certain threshold “game_move_right” gets called. If the y-value of the analog controller is below a certain threshold, the game will do a fast fall meaning that the teesy will call “game_fall” every 66 milliseconds, making the piece fall down faster than it usually would. If the c button is pressed, “game_rotate_clockwise” will be called, which changes the orientation value of the active tetromino. If the z button is pressed, the game executes a hard drop, which calls “game_drop” until the method returns false, because it can no longer move down anymore.

To draw the game onto the screen, “game_visible_grid” is called which returns a two dimensional array with the color at each coordinate. For each coordinate, “draw_tet” is called which takes the x-coordinate, the y-coordinate, and the color as parameters, internally it uses “gfx.fillRect”, from the ILI9341_t3 library, which abstracts away the SPI communication used to communicate with the display driver. Additionally gfx.fillRect is also used to display the tetromino in the “hold” box. And the three tetrominos in the “up next” box.

At the end of each turn, the “game_finish_turn” method is responsible for adding the coordinates of the active tetromino into the static grid as “Filled”, generating a new active tetromino from the bag style randomizer, and clearing lines from the static grid that are entirely full. “game_finish_turn” returns the amount of lines that ended up being cleared.

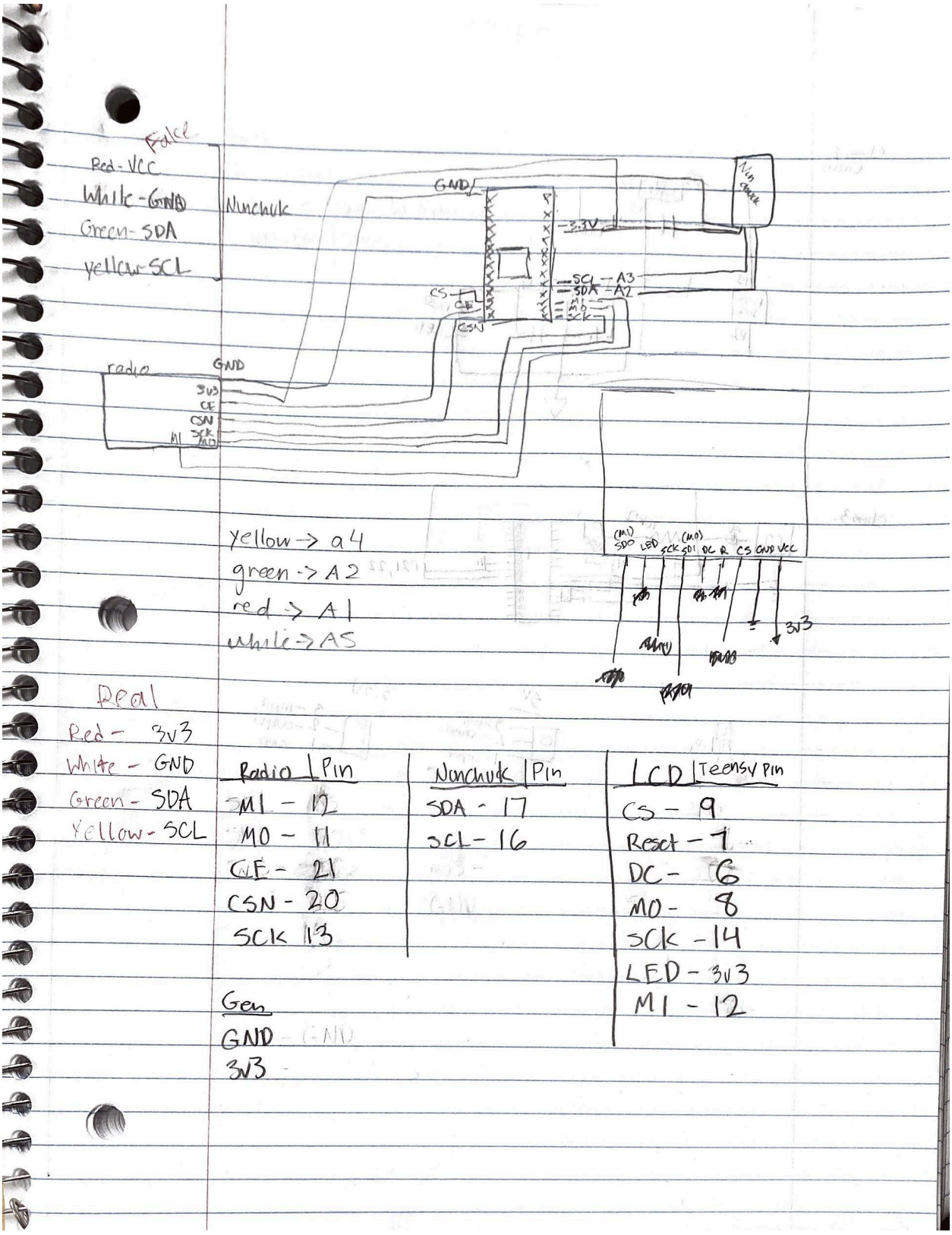
Every 100ms in the “loop” method, the teensy would check if the radio had received a message (using radio.available(), and radio.read()). A message is one byte long, with the first bit set to 1 if the other game has lost, and the last 2 bits corresponding to the amount of lines to add as “garbage” to the bottom.

If the amount of lines cleared at game finish_turn is greater than 1, the teensy uses radio.write() to send a message to the other game, sending it n - 1 lines.

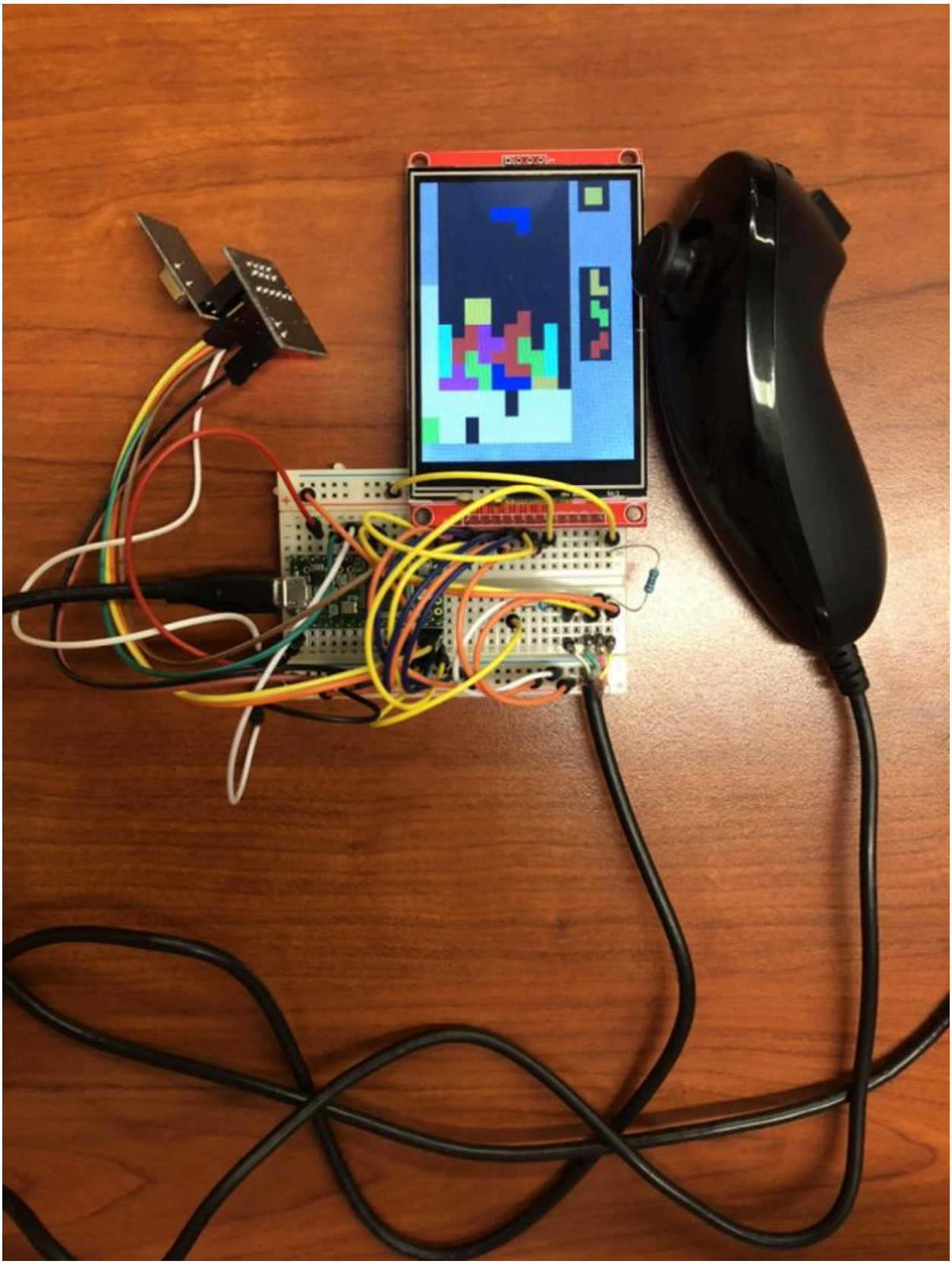
If the new active tetromino generated by game finish_turn is immediately in an illegal position upon generation, then the player has lost the game, sending a message to the other radio with the first bit set as 1.

The game calls game_fall in the loop every 1 second initially, however, the fall speed increases every 10 line clears.

5. Circuit Schematics



6. Circuit Prototype



7. Testing

Controller: Wii nunchuk I2C libraries allowed us to read inputs from the controllers. The positions of buttons and data received from the controller was used to implement the controls within the game. We eventually used an optimized I2C library that allowed the controllers to run smoothly in response to inputs.

Screen: Ran an example code from the TFT library that printed different colors on the screen. The screen was capable of drawing multiple colors. We then attempted to draw/erase the shapes of the tetromino. The screen was capable of running a smooth game.

Radio: Pseudocode made both radios transceivers where each were given the functions of sending and receiving code. The code made each radio send the “millis()” value as the program ran. Each radio worked as expected.

Final Overall:

In order to test that the final system was functioning, we cleared lines on one controller and made sure the other recieved the signal to create another for the player. We also tested that the controllers always worked reliably and did not create any problems for the user.

In the end, both systems worked completely, but we ended up breaking one of the controllers, so in the meantime the system does not work until we find a replacement. The game works to all our expectations