Gregory Christ
12206975
ECE 4220
December 16, 2010

Tetris

Tetris is a game that uses shapes that stack on each other as they fall down the screen. These shapes also move around the screen by means of button pressing by the user. Since the movement during game play is continuous, it is imperative to check the next location of this shape. The game is shown on the computer screen using text characters so small amounts of graphics is involved in displaying results. Since there is so much movement going on in Tetris, the information needs to be continuously updated and displayed for the user.

This project utilizes the TS-7250 Single Board Computer and buttons on the I/O board that were used during labs. This board runs my program of Tetris and captures the button pressing on the board while displaying the game on the computer screen. The game is shown on the screen as text and is continuously updated according to buttons pressed by the user and the game play. Three buttons are used to move the shapes left, right, and one to rotate the shape.

The problem that was required to solve this version of Tetris was how to continuously update text on a screen while also incorporating movement of the shapes determined by the user. The uses of real time tasks were mandatory in the code to keep track of when and where shapes needed to be on the screen. Information from the buttons is obtained by the means of memory mapping. This information uses a First In First Out (FIFO) approach to send messages using mailboxes. A thread is used to run in the background of the rest of the code to keep track of the game timer. The information from the game is stored and printed from a simple 2D array.

Gregory Christ
12206975
ECE 4220
December 16, 2010

All the data used to play Tetris is contained in a 13x16, character array called *game_data[][]*. First of all, I wanted to place the game board into this array to define the bounds of the game. The sides are denoted by "*|*" which are placed on the far left and right three columns. Next, I wanted to create a real time task that would check for button pressing. This was accomplished by creating three child processes. Each child process will monitor a single button, left, right, and up, using a real time task. The buttons on the I/O board are connected to Port B of the TS-7250 board and a pointer that points to Port B's memory address is mapped into memory using *mmap()*. The direction of Port B is set so it is an input by ORing it with 0xF8. A structure called message that contains an integer value and other values for purposes of checking and debugging. A mailbox called *btn_mbox* is also set up for means of interprocess communication when the buttons are pressed. When a button is pressed and let go, it sets the integer value in the message to the integer value corresponding to that button. This message is now sent in the mailbox using *rt_mbx_send()* that is sent back to the *main()* that receives these messages using *rt_mbx_receive()* after a small wait period. The message is looked at and calls *left()* when it is 0, *right()* when it is 1, or *up()* when it is 2.

Next, I wanted to create a real time task that will act as the game play timer so the shapes will drop down one row every second. In order to create this timer, a thread is created using *pthread_create()* to run in conjunction with the rest of the program. When the time is up, the shape data is moved into the row located below it. Once this timer is setup, a random number called *cur_shape* is generated to choose the shape to be placed in the game. Another number called *cur_dir* is also generated to determine the orientation of this shape. The shapes that are created are the "I" having two orientations and the "T" having four orientations. The shapes are

defined by manually placing "O"s into the correct positions in the array. The top and bottom row and well as left and right columns and a center row or column is defined for each shape. These are defined in the program by *cur_top_row*, *cur_btm_row*, *cur_left_col*, *cur_right_col*, and *cur_cntr*. This is how the program knows where the current shape is located in the array. Incrementing the *cur_top_row* and *cur_btm_row* creates the illusion that the shape is falling down the screen.

Since the buttons are monitored, the shape will move left or right depending on the button. The illusion that the shape is moving left is created by decrementing *cur_left_col* and *cur_right_col* by one. The illusion that the shape is moving to the right is created by incrementing those same values by one. When the up button is pressed, it changes the orientation of that shape. In order to do this, the *cur_dir* value is incremented by one to change the orientation. If *cur_dir* reaches the maximum number in that shapes orientation then it set is back to 0 acting as a continuous circle. The *cur_top_row*, *cur_btm_row*, *cur_left_col*, *cur_right_col*, and *cur_cntr* values have to be redefined to allow the shape to look like it is rotating. While having the shape drop and moving the shape left and right is great, checking where the shape is going is a mandatory practice. The program makes sure that the location where the block will be moving to does not contain parts of other shapes and does not go out of bounds. To make sure it's not running out of bounds, it checks that the next location is not a "*" to the left or right depending on movement of the shape. Otherwise, it just makes sure there is a " " (blank space) in the next location and then it allows the shape to move. If the shape is no longer able to fall then it tells the program that a new shape must be created and put into play. Thus, a new random number is generated for *cur_shape* and *cur_dir*.

The program always calls a function called *print_data()* that prints the entire 2D array with the updated information anytime the shape moves. It uses the simple print statement called *printf()* in a *for()* loop to print the results. Before this information is displayed, the old data needs to be cleared from the screen using *printf("%c[1A", 27)* and *printf("%c[2K", 27)* which clears an entire row and then moves up a row on the screen. This is also put in a *for()* loop so it clears the multiple lines of old information.

Tetris is a game that is very hard to recreate using text shapes since the shapes need to move and rotate as a unit. I was able to create two different types of shapes and create the different orientations for each. Most of this results section should be considered while watching the included video since it is difficult to prove that this worked since this is a visual project, but was tested several times. These shapes were able to be created and rotated around while keeping its shape. This was tested several times as the shapes were created and rotated. I was also able to successfully keep the shapes within the boundaries of the game board limits. The dropping feature created the illusion that the shapes were falling down the screen, but actually just moving down the rows of the array. It also seemed to be falling at a constant rate. When I was moving the shapes left or right, I could press buttons multiple times while in between drop times and it would do what I told it to do. The rotation of these shapes was fluid as well. My code also acted like Tetris since the shapes would stop falling once the shape hit parts of other shapes already in the game of play. The included video shows multiple shapes being created that can be moved left and right or rotated and falls down the screen until it hits other shapes in play.

For the longest time, I had the hardest time making the buttons work in conjunction with the rest of the code. I was able to easily create a shape and place that into play and make that fall

down the screen. At first I was trying to use a module that would use interrupts to determine the button pressing. I was going to write to a FIFO that would relay a message to my main code, but when I compiled it, the compiler did not me opening and writing to a FIFO in a module. This made me turn towards creating children processes to monitor the buttons. Once this was working using children processes, the next hardest thing was to check the next location when moving left and right and rotating since there are multiple areas in the array that need to be checked. Also, converting the shape rows and columns to rotate into a different orientation was very hard to visualize but eventually accomplished.

Overall, my program is very close to acting like the game of Tetris. The next thing that I need to do is create the other shapes that are used in the game like the two types of L shapes, the two types of zigzag shapes, and the square as well as the different orientations for each shape. I would also have to figure out how to convert the shapes rows and columns to make them look like the orientation of the shapes are rotating when the up button is pressed. My code really doesn't act like Tetris since I do not check the lines for a full line when the shape finishes falling. I spent so much time working on the buttons and the rotating that I did not incorporate this into my program. Also, the next location of the shape when rotating needs to have more checking since I need to look at more areas of the array. This would make the program work smoother. My program also prints out junk in the bottom row of my array and I'm really not sure why so that is something that I will have to continue to look at in the future. I feel like this program is so close to being a Tetris game and shares the main idea of a Tetris game. A little more time on this project would help solidify the program as Tetris.