Paul McElroy
EECS 388 Lab Project 6

Project Description:

My project makes use of the matrix keyboard, which has numbers 0-9, * and #; the keyboard has 3 column attachment wires and 4 row attachment wires.

The project I designed makes use of this keyboard to play a simple game where the player controls a Momma Bird and collects enough worms to feed her Bird. The player moves the Momma Bird (@) around the screen to collect worms (~), she will also have to find some hidden worms in some dirt patches (#) and while collecting worms, the Momma Bird may be moved to the Baby Bird (&) to drop off the worms so far collected. To move the Momma Bird, the player uses the 8 number keys surrounding the 5 key to move the Bird in the appropriate direction. The dirt patches are "pecked" at by pressing the * asterisk key on top of them; worms are dropped off top of the Baby Bird by pressing the # pound key when the player has at least 1 worm. The program provides helpful dialogue during play to assist the player in what to do next.

The important components of the program are the main file, the buttons module, the game module and the OLED module. The system and LED modules are left untouched and function like they did in previous projects. FreeRTOS is used to schedule the tasks instead of the Polling method.

The main file's only change has been the addition of the buttons and game tasks. The memory usage and priority of the tasks has been updated to suit the tasks.

The buttons module controls the behavior of the buttons task. The 3 column attachment wires are attached to Port E, while the 4 column attachment wires are attached to Port B:

Pin Layout (the left numbers are the I/O pins the board):
Row 0 to 19
Row 1 to 18
Row 2 to 17
Row 3 to 16

Col 0 to 15
Col 1 to 14
Col 2 to 13

The buttons task enables all the proper ports and pins. Then in its while loop, it turns all the columns to their HIGH state and then one by one turns them to low and checks each Row for a LOW output. Any Rows with a LOW output are being pushed and this is recorded. Then that Column is set to HIGH and the process repeats with the next Column.

The button task then determines which buttons were released during the current tick by comparing the buttons pushed this tick to the buttons be pushed last tick. Actions in the program occur when a button has been released during the current tick. Multiple button presses are handled for the direction buttons by prioritizing the directions in this order: UP, UP-RIGHT, RIGHT, DOWN-RIGHT, DOWN, DOWN-LEFT, LEFT, UP-LEFT (though these are described using the Cardinal direction names in the program code).

The game task's first sets up the game board by placing all of the pieces and the player character (@). When it enters the while loop, it takes the array of buttons released this tick and translates them into action. If a direction was pressed, it moves the (@) in that direction if it is a valid move and a worm is picked up if one is present and the player's number of worms is updated. Standing on top of a unique tile with the (@) is handled by labelling every combination with a unique integer. Depending on the integer, unique actions are taken (picking up worms, not overwriting Dirt or the Baby with the @, or updating helpful messages at the bottom of the screen).

Five helper functions are used by the game task. getPlayerX and getPlayerY are getter functions that return the player's column and row coordinates as integers. setPlayer handles moving the player to a new position, using the isValid function to check if the move is valid (doesn't leave the playing area), and also to update the messageFlag to an appropriate message. Worms are also picked up here and this is also where game manages the overwrite protections of secondary game tiles.

If the player collects enough worms, the game tells the player to drop them off at the Baby Bird, and when the Baby has all the worms the game ends with a message "Baby's Full! You Win!"

The OLED task updates the screen and handles the game layout as well as the helpful messages to the player. First the OLED runs the intro screen which has been updated to show what each tile means. Upon playing, it uses the drawGrid function to draw a faint grey grid to help the player tell where each space is. It uses the drawPieces function every tick to update the playing area with the current piece layout and it uses the drawMessage function to draw the helpful messages set by the messageFlag in the game task.

My experiences with this Project:

I had a lot of trouble with the buttons task because I had the pins laid out all wrong and the inputs and outputs switched around compared to how I had set up my Pin types (GPIO_PIN_TYPE_STD_WPU vs GPIO_PIN_TYPE_STD). I spent nearly 3 to 4 hours hashing that particular mistake out until I finally realized what I had done.

I had originally planned a much simpler game, where you move an X to a O, picked it up and dropped it at another O to finish the "game". My current project grew out of the desire to make something that was actually interesting.

A few improvements would be to make graphic representations of the Baby Bird, Momma Bird, Dirt and Worms using the DrawImage function, and to make the game have a random layout of a variable number of pieces at the beginning of a new game.

I learned a lot I didn't know about implementing game logic, which I suppose will apply to other endeavors. I used the Stellaris Manual a lot for Pin numbers but didn't use the API at all since I already learned what I'd use in past labs.