CS/EE 120B Custom Laboratory Project Report

Brick Breaker

Javier Delgado

December 8, 2025

Introduction

Brick Breaker is a two-dimensional game where a player tries to obtain the highest score by breaking the bricks by bouncing the ball off the paddle. The player controls the paddle by moving it right or left in order to keep the ball from falling into the void and losing a life. Each point is obtained by breaking a brick, so the bricks will disappear as more bricks are hit with the ball. The game ends and the player wins when all bricks are broken or when the player loses by failing to hit the ball with the paddle. The game becomes more challenging by making it harder for the player to hit the ball with the paddle and thus harder to hit the bricks.

Known Bugs

1. Moving the joystick all the way to the left can cause the paddle to go off-screen and hinder the player's ability to correctly hit the ball with the paddle. The error is due to the values read from the joystick being so inconsistent that it became difficult to constantly update the code to accommodate the new, different values read from the joystick.
2. The ball sometimes appears to have fallen past the paddle due to the ball speed being increased, making the player possibly think that they lost the game, but as long as the player correctly places the paddle underneath the ball, then the game continues.
3. While not a game limitation, there is some noticeable black flickering, typically when the ball hits the sides of the bricks. The reason for this is that the code clears the ball's old position, and the area being cleared overlaps with the bricks being rendered.

Build-Upons

- Successfully Implemented
    - EEPROM to save Game State (1/2 complexity)
    - 16x2 LCD from parts kit display the player's score & game messages (1/2 complexity)
    - Using a HiLetgo ST7735 LCD screen to display the basic shapes like squares that act as the background for the player's gameplay (1 complexity)
    - Using a HiLetgo ST7735 LCD screen to display custom sprites (1 complexity)
- Not Implemented
    - Support for 2-player mode (1 complexity)
    - Add a shift register to give more available pins for outputs for the parts kit LCD screen (1 complexity)
    - Complex game logic (1 complexity)

User Guide

- Game Controls
    - Ensure that the system has power by plugging in the ATmega328P
    - The push button resets the entire game to the start screen
    - The joystick controls the paddle that is at the bottom of the screen, used for gameplay
    - The joystick button is used only to begin the game when at the start screen
    - The 16x2 LCD is used to display the player's score and messages relating to the game
    - The potentiometer is used to adjust the brightness of the 16x2 LCD
    - The entire gameplay is displayed on the ST7735 128x128 LCD
- Game Rules
    - To win the game the player must guide the ball in the direction of bricks and breaking all 25 bricks wins the game.
    - Losing the game means that the player let the ball fall past the paddle or rather the paddle was not underneath the ball in time as it fell towards the bottom of the gameplay screen. The highest score after losing is kept and displayed on the 16x2 LCD.
    - Using the joystick, the player can change the direction that the ball travels by aiming for the ball as the ball is falling down to hit the middle of the paddle, the right edge of the paddle, or left edge of the paddle. If the ball hits the middle region of the paddle then the ball travels straight up. If the ball hits the right edge then the ball travels diagonally right. If the ball hits the left edge then the ball travels diagonally left.
    - The ball will stay within the gameplay screen and thus bounce off the walls.
    - Every 5 bricks that the player breaks will increase the speed.
- Game Features
    - If power is lost at any point during the game, once the user plugs power back in the EEPROM will load in previous game data before power was lost such as bricks already broken, ball's speed, and player's score.
    - If power is lost after the game ends, when power is restored the player's score will be saved and the game messages on the 16x2 LCD and 128x128 LCD will be displayed again.

Hardware Components

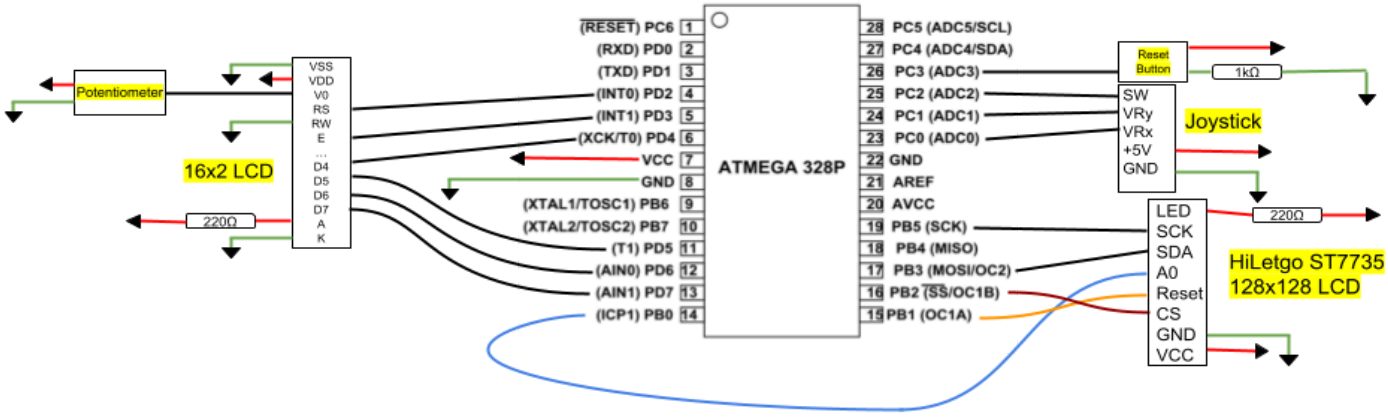
- ATmega328P Board
- Joystick
- Push Button
- Potentiometer
- HiLetgo ST7735 128x128 LCD

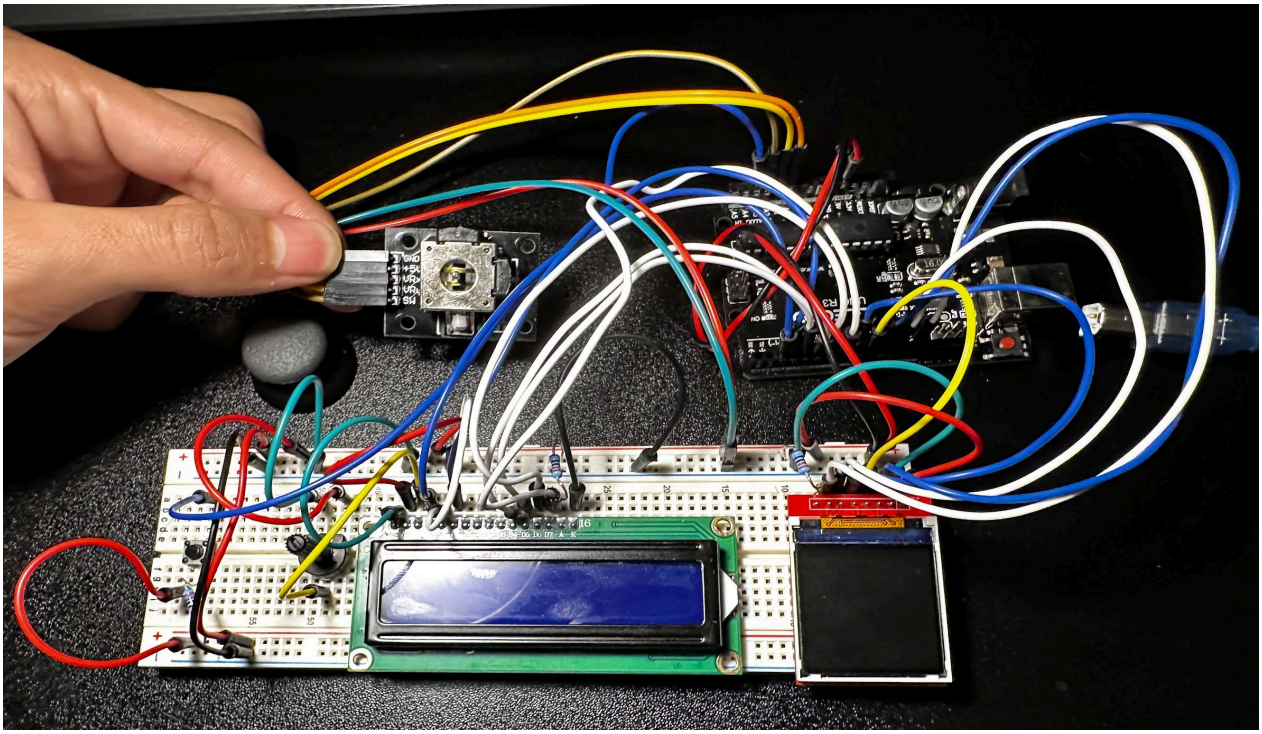
- ELEGOO UNO R3 Super Starter Kit 16x2 LCD screen

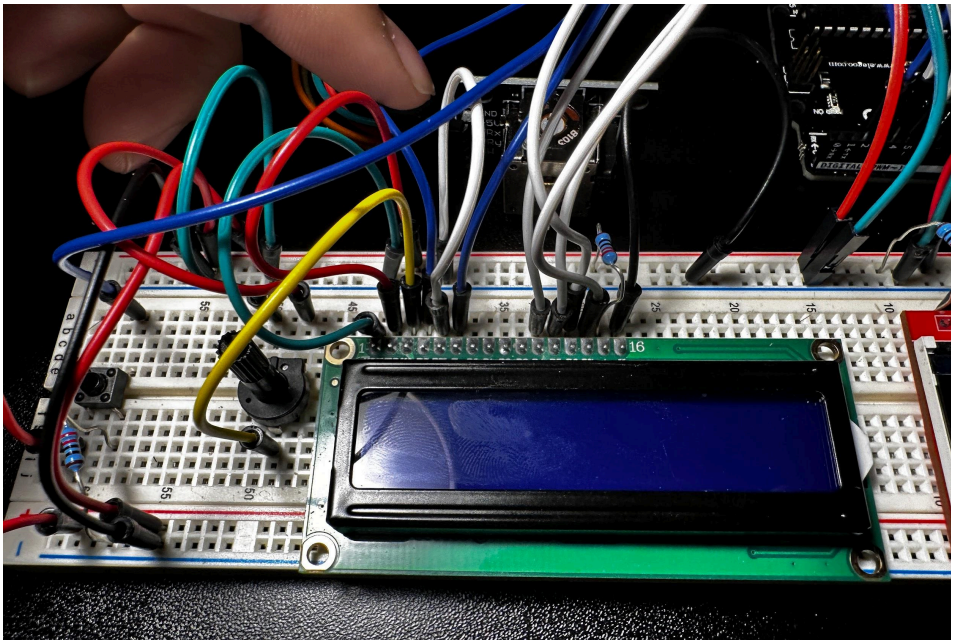
Software Libraries

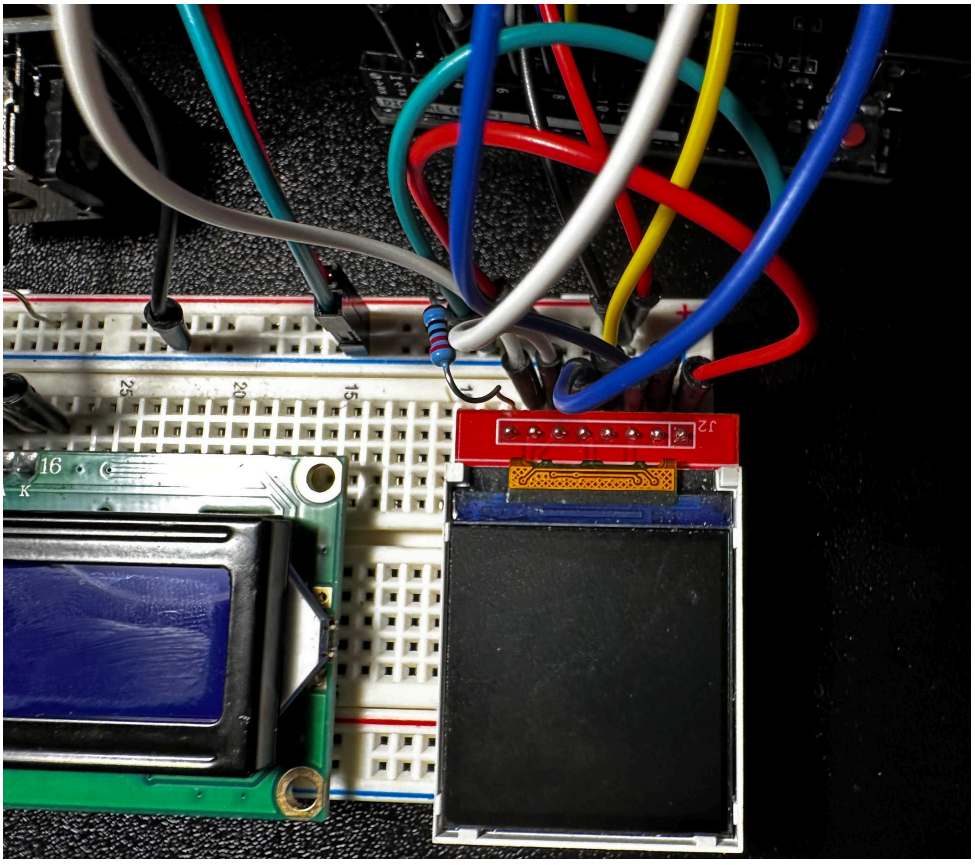
- <avr/pgmspace.h>
    - I used this library to store sprite data since the amount of bytes used to write custom sprites to the ST7735 LCD exceeds the SRAM limit of 2 kilobytes. Using ATmega328P Program Memory allowed me to store up to 32 kilobytes of custom sprite data.
- <stdio.h>
    - To convert the player's score from an integer variable to a string that would allow the score to be printed to the 16x2 LCD.
- "EEPROM.h"
    - Using EEPROM as a build-upon required me to include this library, but I researched different functions included in the libary to see which was best for my code, which ended up being writing to the ATmega328P in blocks instead of bytes.

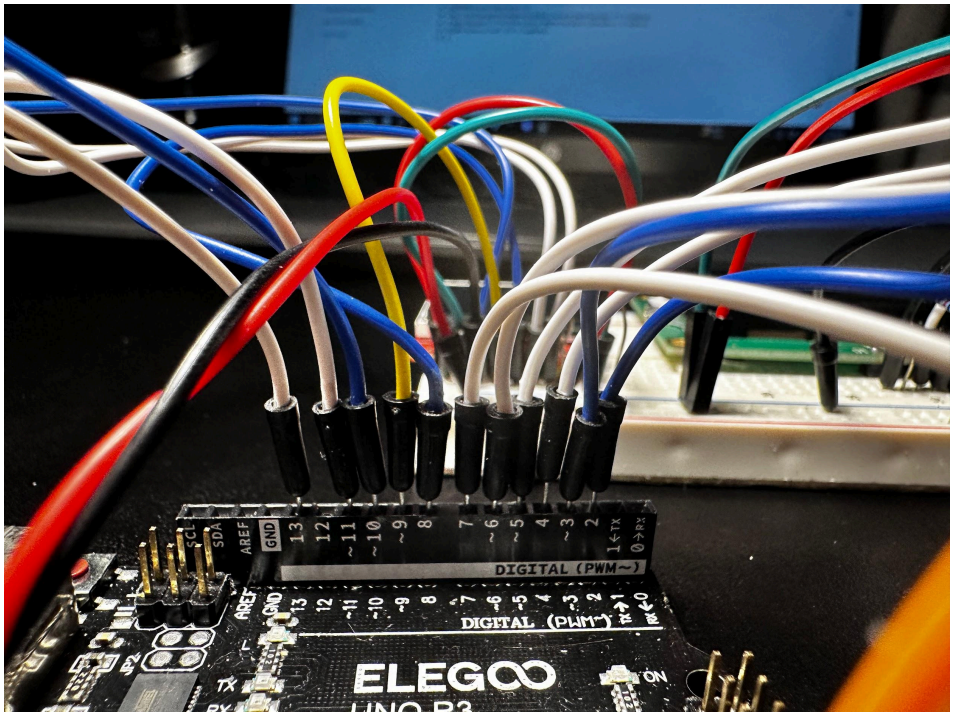
Wiring Diagram

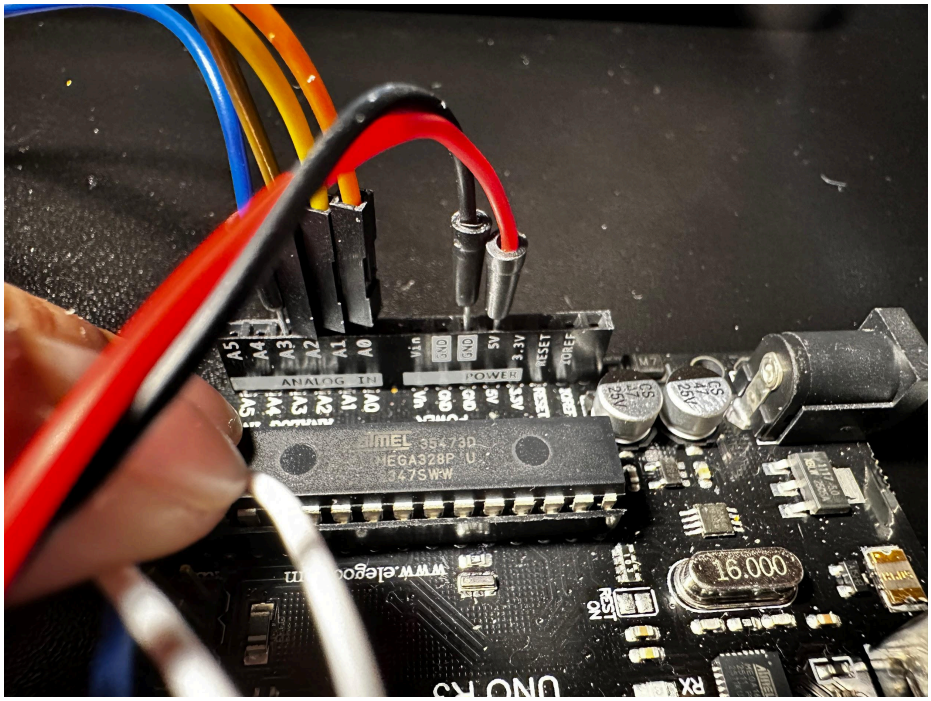




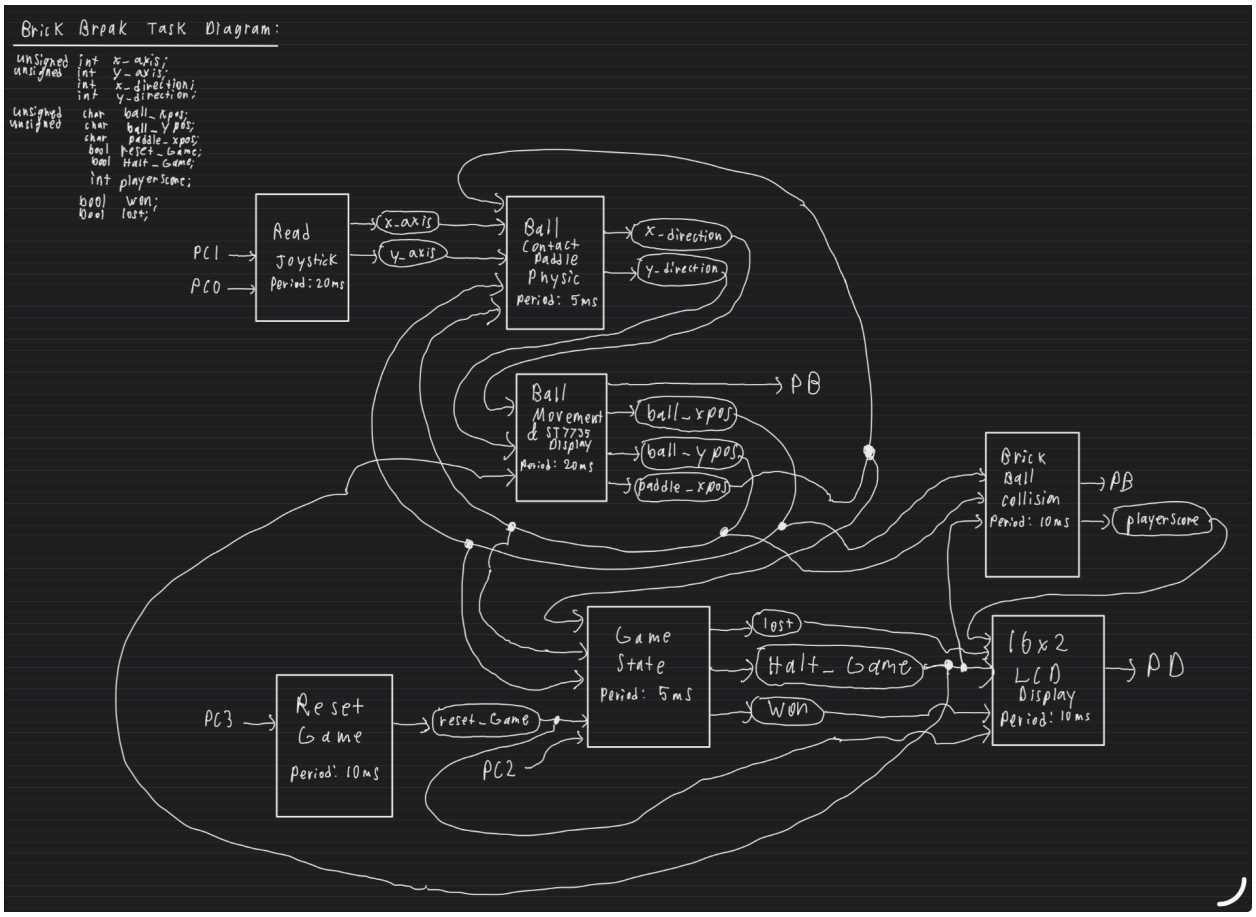




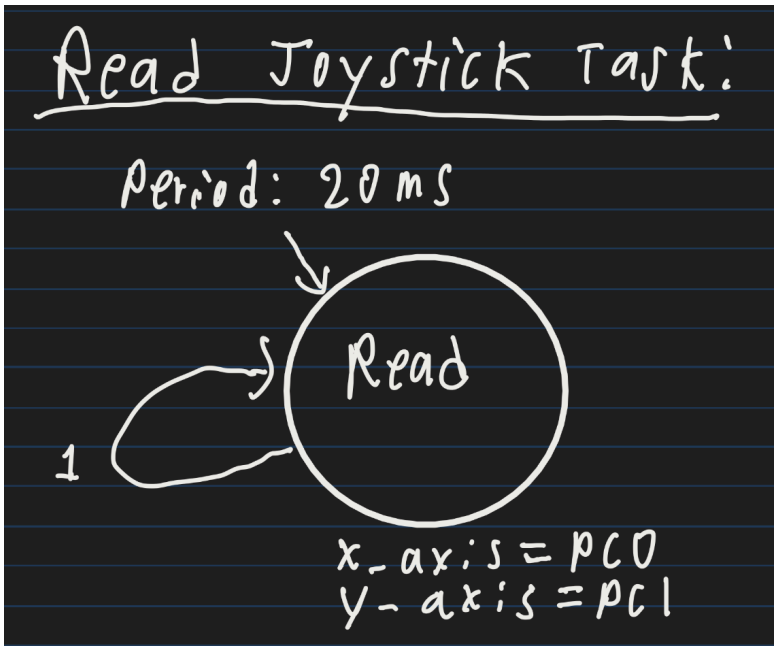

DIGITAL (PWM~)
ELEGOO
ON


ANALOG IN
POWER
16.000

Task Diagram



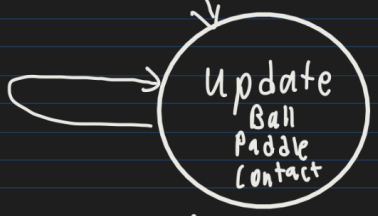
Synch SM Diagrams



Ball Contact Paddle Physics Task:

Period: 5 ms

```
bool hitMiddle;
bool hitLeft;
bool hitRight;

x_direction=-1;
y_direction=1;
```

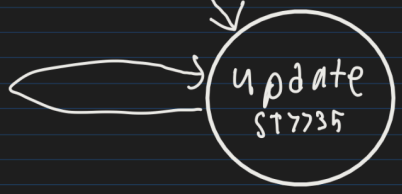



```
/* Code to compare
ball_xpos & ball_ypos
w/ paddle_xpos to
set hitMiddle, hitLeft,
& hitRight accordingly then
based on the bool flags
output correct
x_direction & y_direction */
```

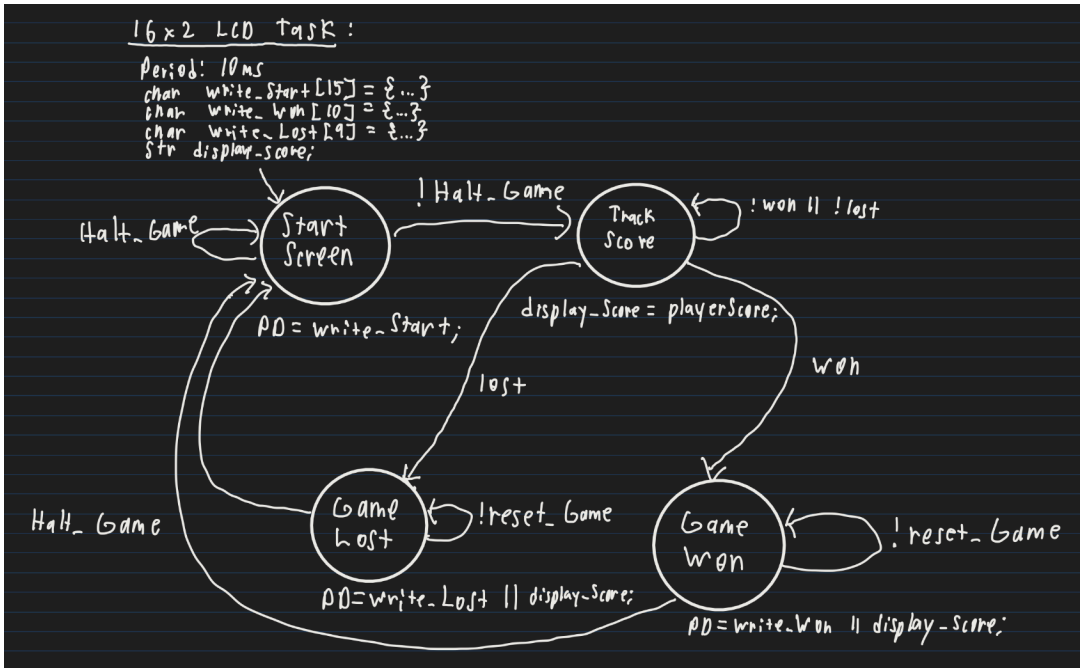
Ball Movement & ST7735 Display Task:

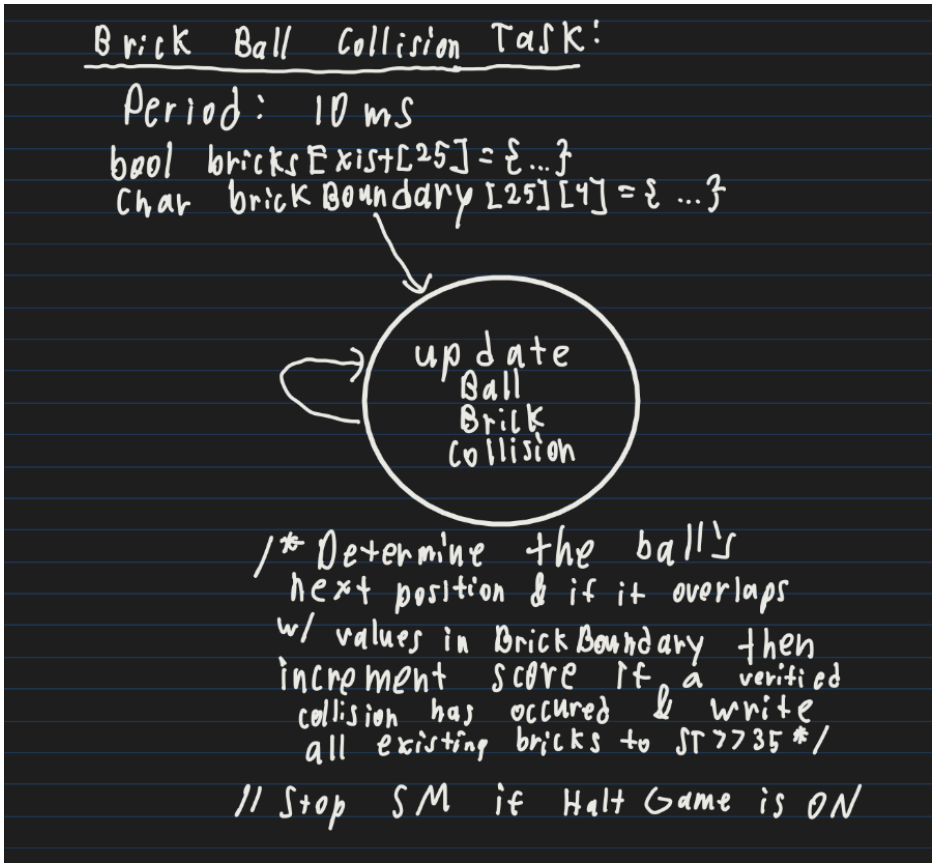
Period: 20 ms

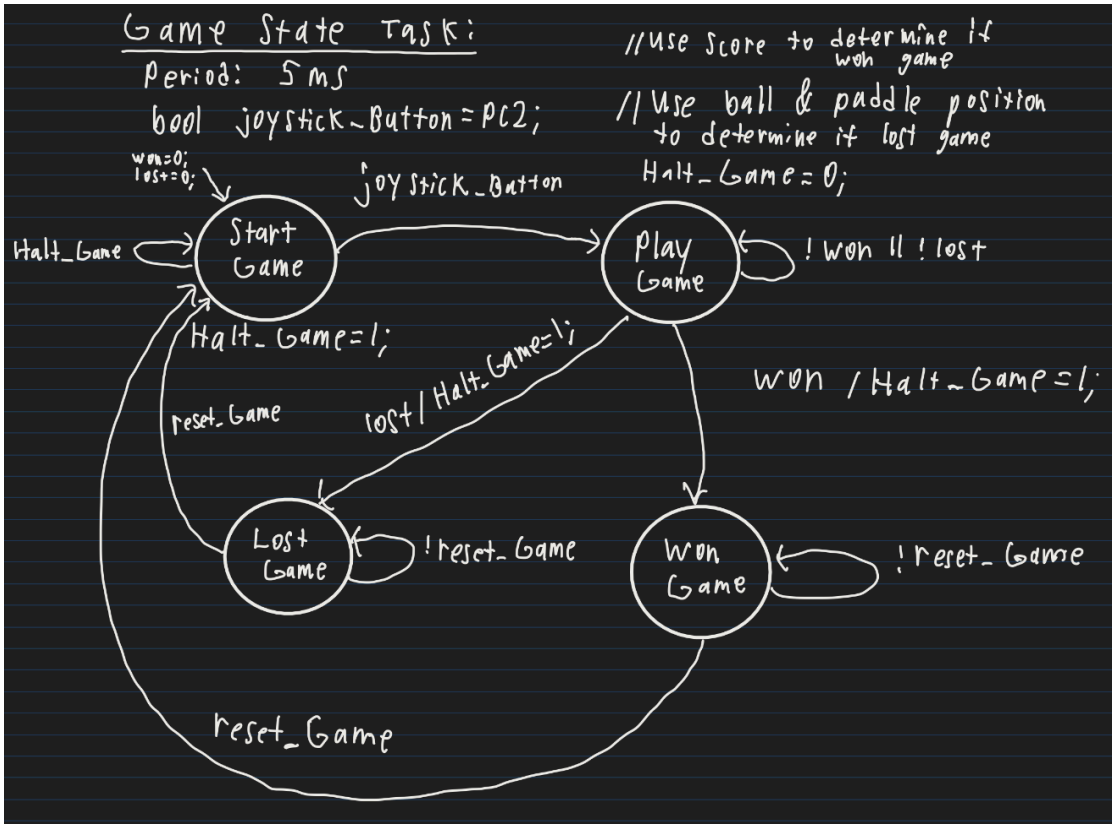
```
unsigned char old_ballx;
unsigned char old_bally;
int ball_xspeed;
int ball_yspeed;
```

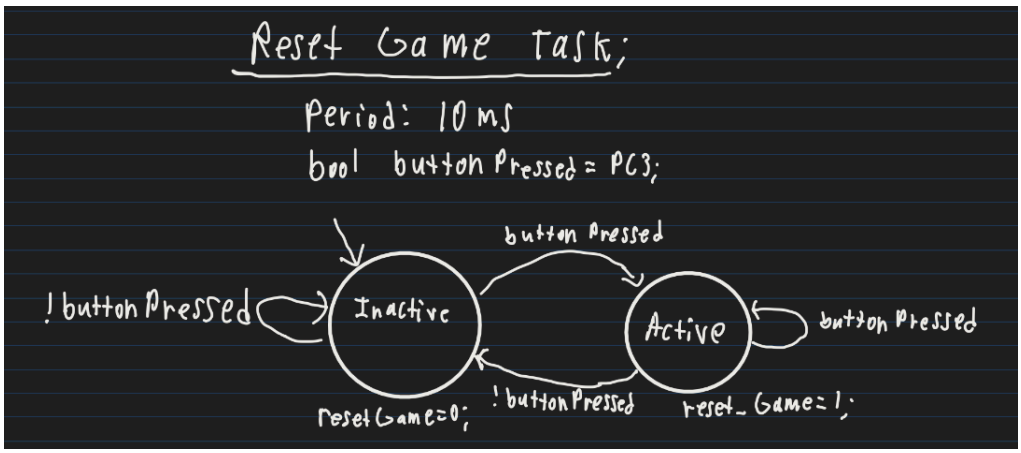



```
/* Clear old ball position
& write to LCD the new
ball depending on the
speed & direction it is
supposed to travel */

/* Stop all display on LCD
if the game has been halted */

// Write everything to PB
for ST7735 LCD
```


16 x 2 LCD Task :
Period: 10 ms
char write_Start[15] = {...}
char write_Won[10] = {...}
char write_Lost[9] = {...}
Str display_score;
Start Screen
Halt_Game
!Halt_Game
Track Score
!won || !lost
PD = write_Start;
display_Score = playerScore;
lost
won
Game Lost
!reset_Game
Halt_Game
PD=write_Lost || display_score;
Game Won
!reset_Game
PD = write_Won || display_score;


Brick Ball Collision Task:
Period: 10 ms
bool bricksExist[25] = {...}
Char brickBoundary[25][4] = {...}
update Ball Brick Collision
/* Determine the ball's next position & if it overlaps w/ values in BrickBoundary then increment score if a verified collision has occured & write all existing bricks to ST7735 */
// Stop SM if Halt Game is ON


Game State Task:
Period: 5ms
bool joyStick_Button = PC2;
won=0;
lost=0;
joyStick_Button
//use score to determine if won game
//use ball & paddle position to determine if lost game
Halt_Game=0;
Halt_Game
Start Game
Play Game
!won || !lost
Halt_Game=1;
reset_Game
lost / Halt_Game=1;
won / Halt_Game=1;
Lost Game
!reset_Game
Won Game
!reset_Game
reset_Game


Reset Game Task;
Period: 10ms
bool buttonPressed = PC3;
buttonPressed
!buttonPressed
Inactive
Active
buttonPressed
resetGame=0;
!buttonPressed
reset_Game=1;