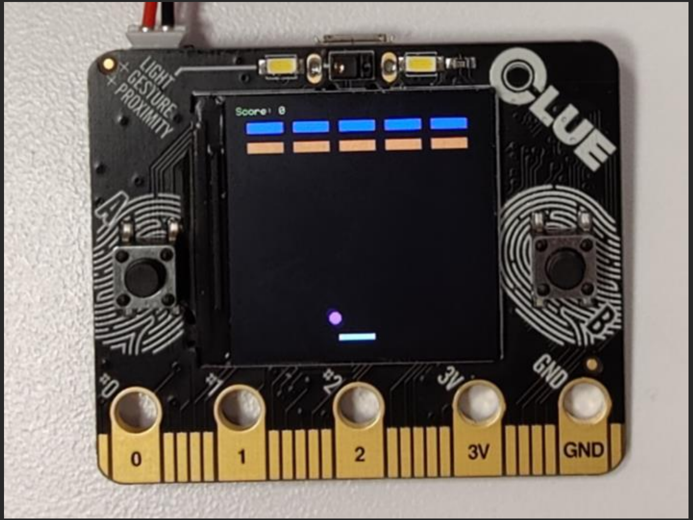
Breakout: Clue Edition



By Bleacher Creatures from Mars: Samuel Burchett, Andrew Dresen, Casey Kolbeck, Kaden Kotvis, and Blake Stumpf

Introduction

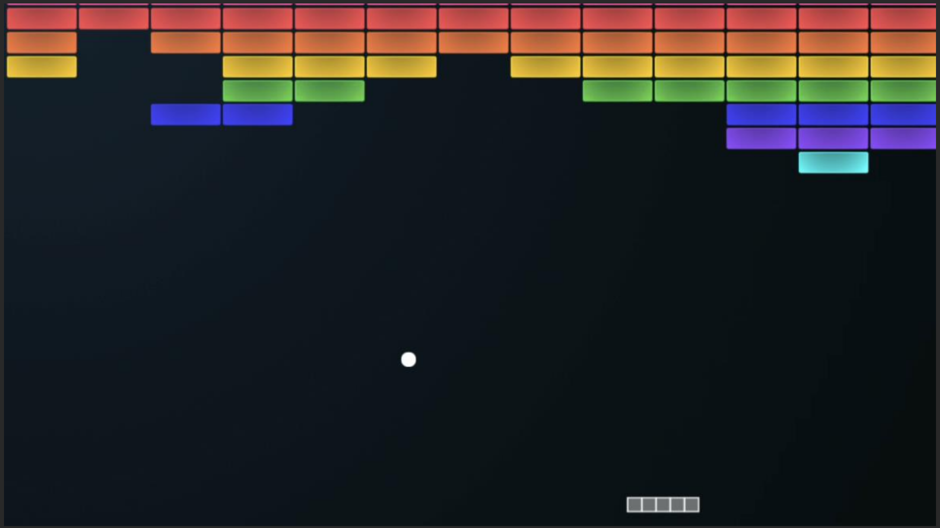
- Slow reaction is a problem in young children and growing adults
- Solution? – Breakout game- fun way to improve reaction time
- Small pocket-sized game that can be played anywhere!

Problem Statement

- Slow reaction is a problem in young children and growing adults
- There is a great demand for video game entertainment

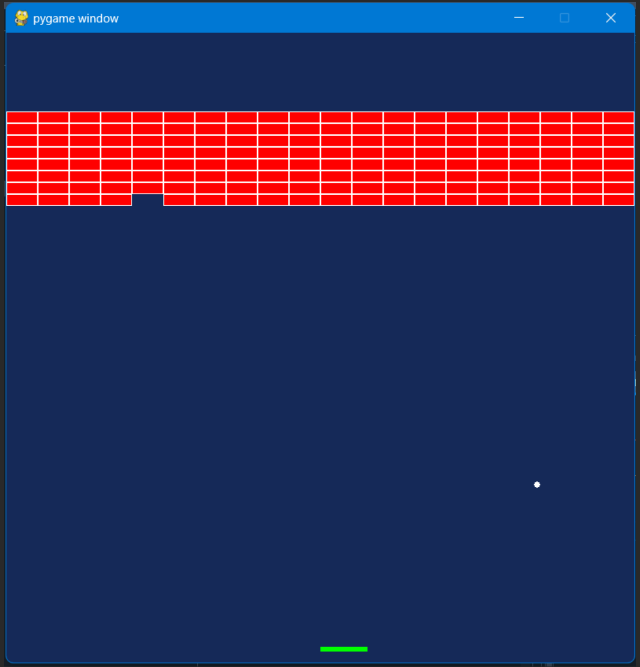
Solution to the Problem

- We created a game based of the 1976 classic Atari game "Breakout"
- Played on a Clue Breakout board



How to Play

- Using buttons on the board to move the bottom paddle
- Ball will bounce off the paddle and hit the blocks above
- The ball will continue to bounce around
- User must hit the ball around and aim to break all the blocks to win



Coding Help

- One Team member had a breakout game written in Python
- Team convert in C for Adafruit clue board
- Found and referenced a similar breakout project

Libraries

- Adafruit Arcada (compilation of a bunch of libraries)
    - Handles pin assignments
    - Handles button inputs
    - Has predefined 16 bit color codes
    - Main subsidiary libraries used
        - Adafruit GFX (handles graphics)
        - Adafruit ST7735 and ST7789 Library (interfaces the display to the microcontroller)

Process of Coding

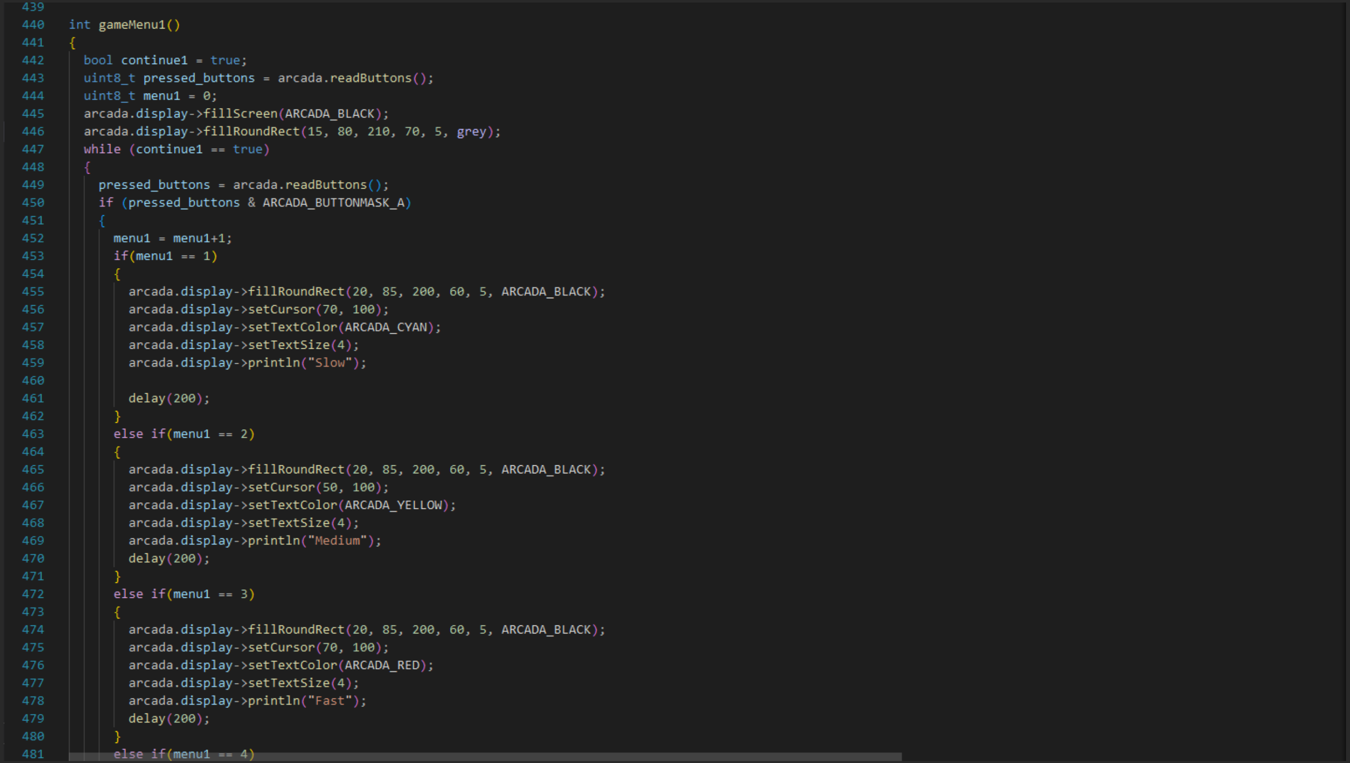
- Setup function
  - Starts display and fills completely black
- Main loop
  - Calls the game menu
  - Calls blocks function to draw the blocks
  - Reads button inputs
  - Calls the paddle function
  - Calls the ball function
- Game Menu Function
  - Creates a menu to select the game speed
- Score Board Function
  - Takes a value to add to the overall score
  - Displays the score in the upper left corner

Process of Coding

- Paddle Function
  - Updates graphics and position
  - Contains bounds interactions for paddle
  - Returns x position of the paddle
- Ball Function
  - Updates ball position and graphics
- Blocks Function
  - Generates Blocks
  - Checks for ball collision with blocks
  - Calls the score function
  - Makes noise with collision
- Bounds Function
  - Checks for collision with paddle or sides

Menu Function

- Initializes menu upon startup
- For each button a press, +1 is added to menu variable
- Displays box with colored difficulty
- Delay in between each input
- After menu equals 4 function resets to one with else menu = 0.

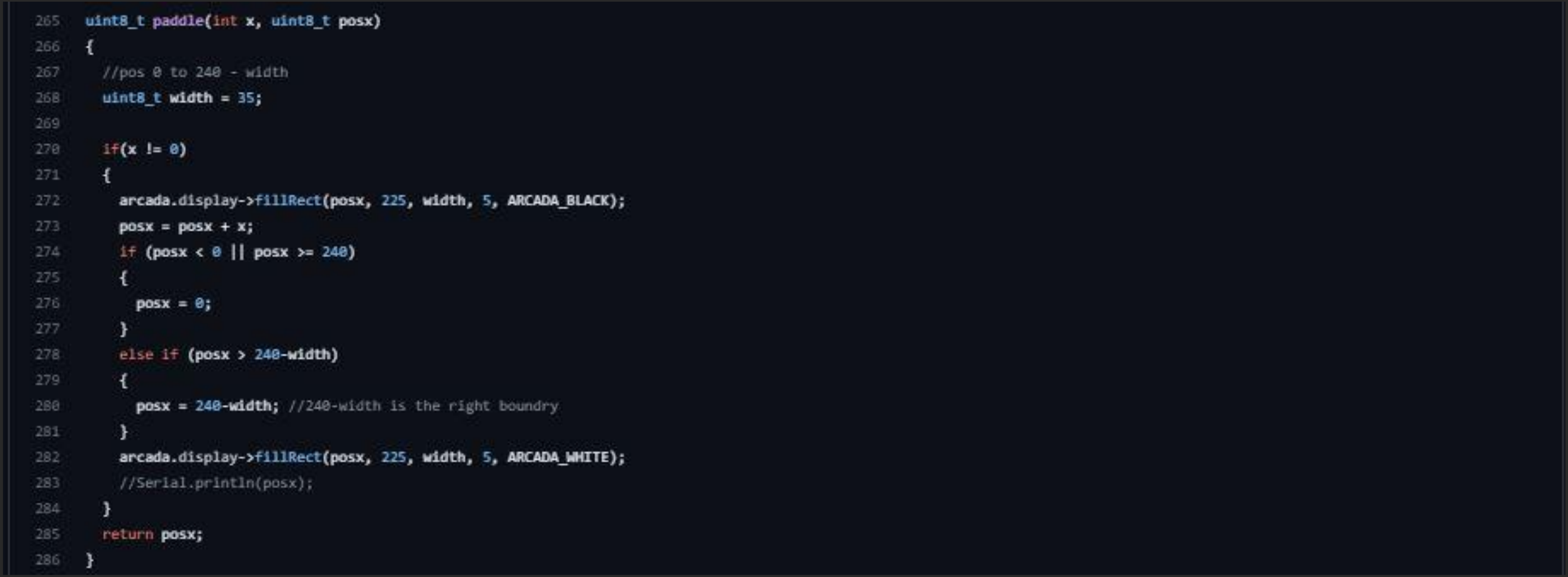


```
int gameMenu1()
{
  bool continue1 = true;
  uint8_t pressed_buttons = arcada.readButtons();
  uint8_t menu1 = 0;
  arcada.display->fillScreen(ARCADA_BLACK);
  arcada.display->fillRoundRect(15, 80, 210, 70, 5, grey);
  while (continue1 == true)
  {
    pressed_buttons = arcada.readButtons();
    if (pressed_buttons & ARCADA_BUTTONMASK_A)
    {
      menu1 = menu1+1;
      if(menu1 == 1)
      {
        arcada.display->fillRoundRect(20, 85, 200, 60, 5, ARCADA_BLACK);
        arcada.display->setCursor(70, 100);
        arcada.display->setTextColor(ARCADA_CYAN);
        arcada.display->setTextSize(4);
        arcada.display->println("Slow");

        delay(200);
      }
      else if(menu1 == 2)
      {
        arcada.display->fillRoundRect(20, 85, 200, 60, 5, ARCADA_BLACK);
        arcada.display->setCursor(50, 100);
        arcada.display->setTextColor(ARCADA_YELLOW);
        arcada.display->setTextSize(4);
        arcada.display->println("Medium");
        delay(200);
      }
      else if(menu1 == 3)
      {
        arcada.display->fillRoundRect(20, 85, 200, 60, 5, ARCADA_BLACK);
        arcada.display->setCursor(70, 100);
        arcada.display->setTextColor(ARCADA_RED);
        arcada.display->setTextSize(4);
        arcada.display->println("Fast");
        delay(200);
      }
      else if(menu1 == 4)
```

Menu Function

Paddle Function

- The inputs are how many units and the direction that the paddle is moving and its current x position
- Fills in old paddle with background color
- If the new paddle position is within the bounds of the screen the new paddle is displayed
- If the new paddle position is not within the bounds of the screen the position is updated so the whole paddle stays on screen
- Returns the updated position

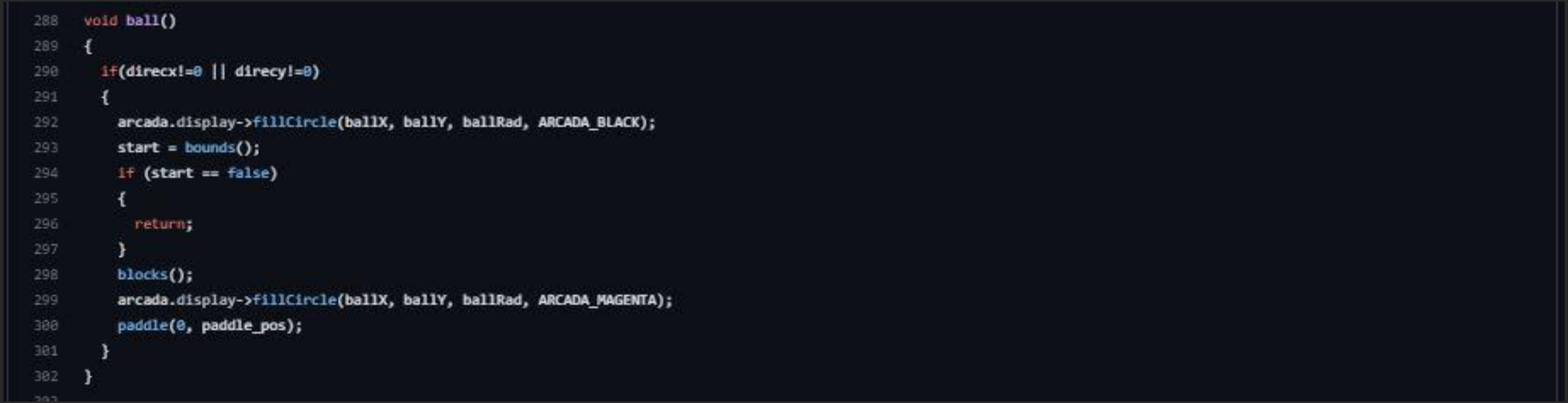


```
uint8_t paddle(int x, uint8_t posx)
{
  //pos 0 to 240 - width
  uint8_t width = 35;

  if(x != 0)
  {
    arcada.display->fillRect(posx, 225, width, 5, ARCADA_BLACK);
    posx = posx + x;
    if (posx < 0 || posx >= 240)
    {
      posx = 0;
    }
    else if (posx > 240-width)
    {
      posx = 240-width; //240-width is the right boundry
    }
    arcada.display->fillRect(posx, 225, width, 5, ARCADA_WHITE);
    //Serial.println(posx);
  }
  return posx;
}
```

Paddle Function

Ball Function

- Fills in old ball position to background color
- Calls bounds function so wall collisions and paddle collisions occur
- Calls blocks function so any block collisions occur
- Draws new ball

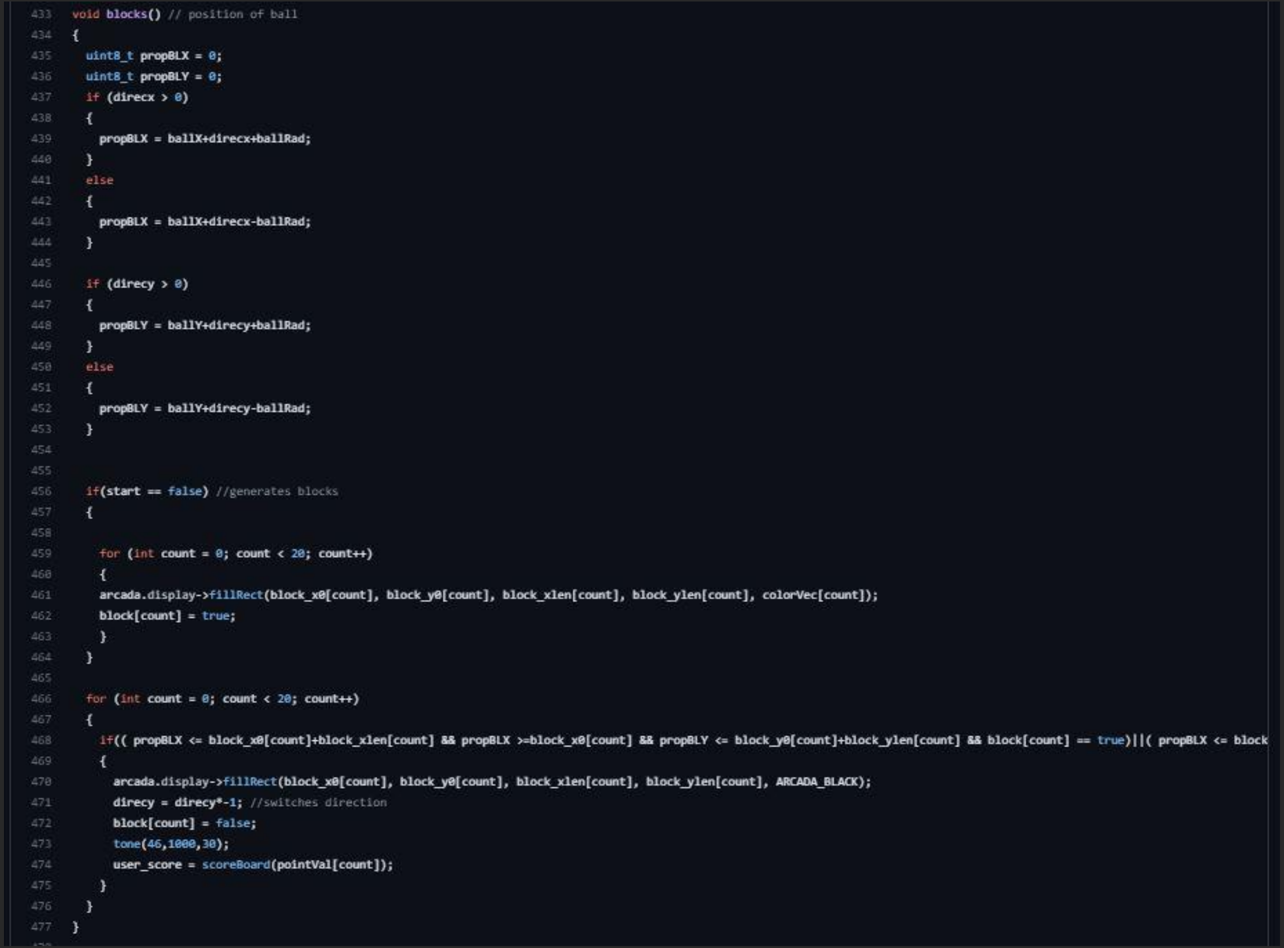


```
void ball()
{
  if(direcx!=0 || direcy!=0)
  {
    arcada.display->fillCircle(ballX, ballY, ballRad, ARCADA_BLACK);
    start = bounds();
    if (start == false)
    {
      return;
    }
    blocks();
    arcada.display->fillCircle(ballX, ballY, ballRad, ARCADA_MAGENTA);
    paddle(0, paddle_pos);
  }
}
```

Ball Function

Blocks Function

- Creates proposed positions for the ball based on the current position and the numbers to be added
- Checks proposed positions if they will get into the block's collision box
- If a collision is detected it deactivates the collision box and draws the block to the background color
- Reverses Y direction for the ball
- Generates the blocks when the global Boolean variable "start" is equal to false

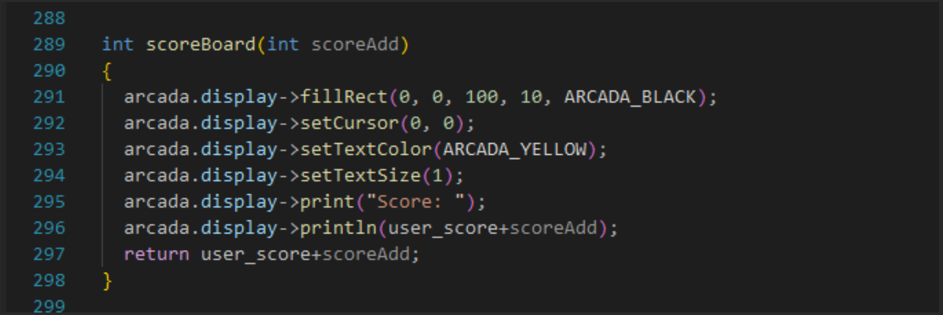


```cpp
void blocks() // position of ball
{
  uint8_t propBLX = 0;
  uint8_t propBLY = 0;
  if (direcx > 0)
  {
    propBLX = ballX+direcx+ballRad;
  }
  else
  {
    propBLX = ballX+direcx-ballRad;
  }

  if (direcy > 0)
  {
    propBLY = ballY+direcy+ballRad;
  }
  else
  {
    propBLY = ballY+direcy-ballRad;
  }


  if(start == false) //generates blocks
  {

    for (int count = 0; count < 20; count++)
    {
    arcada.display->fillRect(block_x0[count], block_y0[count], block_xlen[count], block_ylen[count], colorVec[count]);
    block[count] = true;
    }
  }

  for (int count = 0; count < 20; count++)
  {
    if(( propBLX <= block_x0[count]+block_xlen[count] && propBLX >=block_x0[count] && propBLY <= block_y0[count]+block_ylen[count] && block[count] == true)||( propBLX <= block
    {
      arcada.display->fillRect(block_x0[count], block_y0[count], block_xlen[count], block_ylen[count], ARCADA_BLACK);
      direcy = direcy*-1; //switches direction
      block[count] = false;
      tone(46,1000,30);
      user_score = scoreBoard(pointVal[count]);
    }
  }
}
```

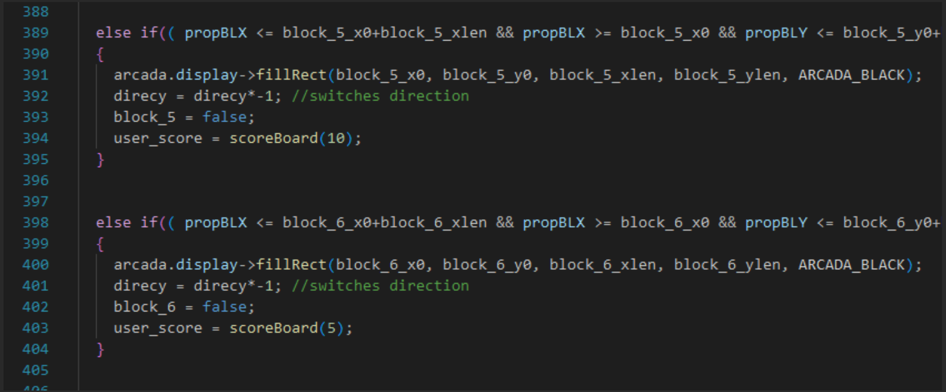
Blocks Function

Score Board Functions

- Initializes scoreboards
- Collision adds a point value
- Prints score



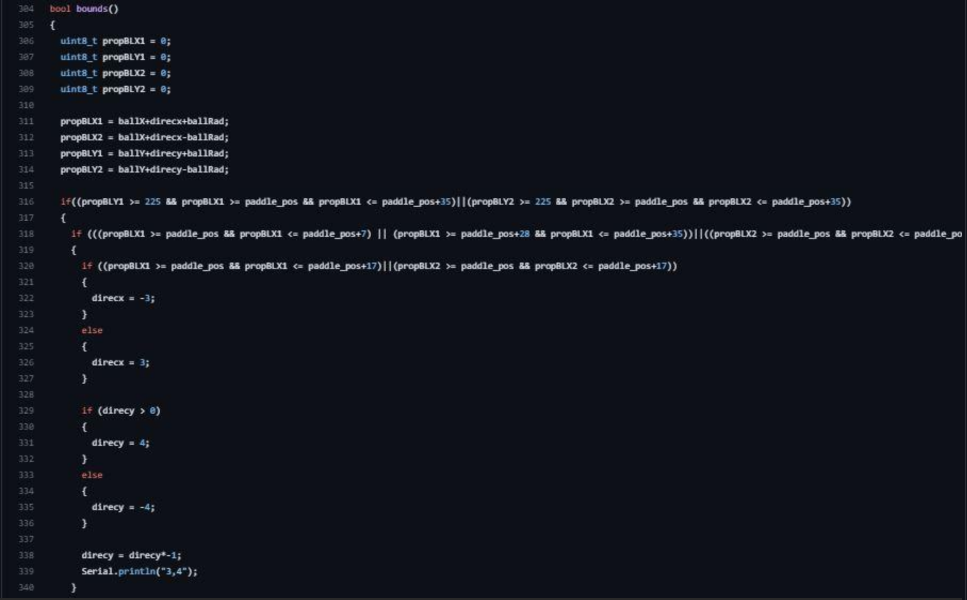
```
int scoreBoard(int scoreAdd)
{
  arcada.display->fillRect(0, 0, 100, 10, ARCADA_BLACK);
  arcada.display->setCursor(0, 0);
  arcada.display->setTextColor(ARCADA_YELLOW);
  arcada.display->setTextSize(1);
  arcada.display->print("Score: ");
  arcada.display->println(user_score+scoreAdd);
  return user_score+scoreAdd;
}
```



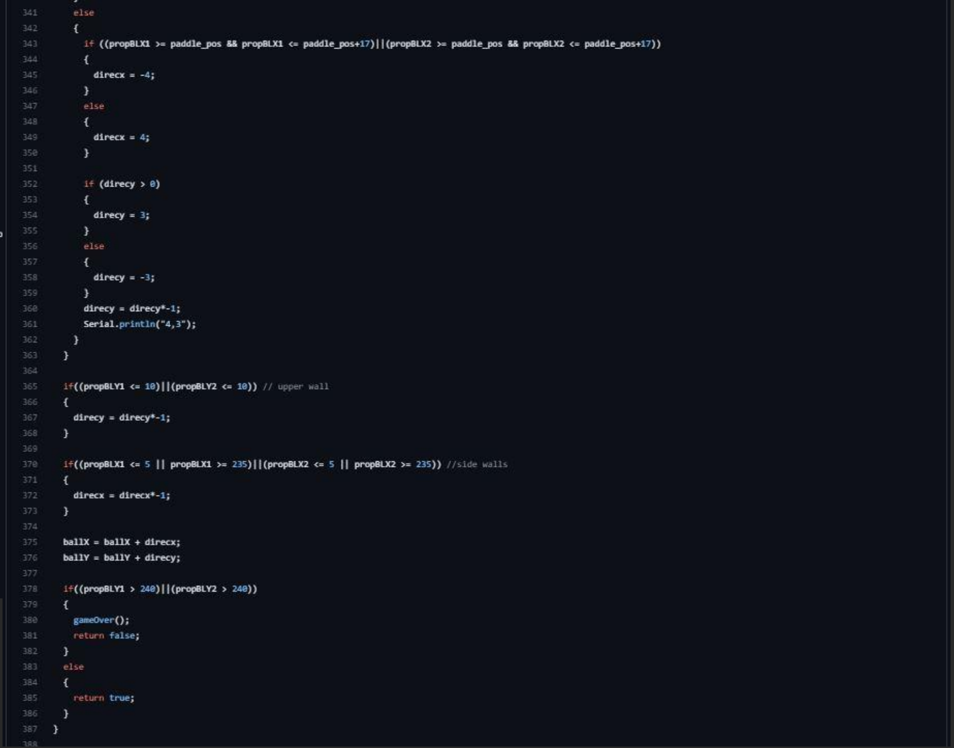
```
else if(( propBLX <= block_5_x0+block_5_xlen && propBLX >= block_5_x0 && propBLY <= block_5_y0+
{
  arcada.display->fillRect(block_5_x0, block_5_y0, block_5_xlen, block_5_ylen, ARCADA_BLACK);
  direcy = direcy*-1; //switches direction
  block_5 = false;
  user_score = scoreBoard(10);
}


else if(( propBLX <= block_6_x0+block_6_xlen && propBLX >= block_6_x0 && propBLY <= block_6_y0+
{
  arcada.display->fillRect(block_6_x0, block_6_y0, block_6_xlen, block_6_ylen, ARCADA_BLACK);
  direcy = direcy*-1; //switches direction
  block_6 = false;
  user_score = scoreBoard(5);
}
```

Bounds Function

- Generates proposed positions for the ball
- Checks for collision with paddle
  - If the ball hits far-right side of paddle the ball will bounce to the right
  - If the ball hits inside-right side of paddle the ball will bounce the right at a steeper angle
  - Same for left side but ball only goes to the left
- If the proposed location of the ball is going to hit the sides the X vector for the ball is reversed
- If the proposed location of the ball is going to hit the roof the Y vector for the ball is reversed
- If the ball's proposed position is going to hit the bottom the game ends
- Returns Boolean value to say if the game ends



```
bool bounds()
{
  uint8_t propBLX1 = 0;
  uint8_t propBLY1 = 0;
  uint8_t propBLX2 = 0;
  uint8_t propBLY2 = 0;

  propBLX1 = ballX+direcx+ballRad;
  propBLX2 = ballX+direcx-ballRad;
  propBLY1 = ballY+direcy+ballRad;
  propBLY2 = ballY+direcy-ballRad;

  if((propBLY1 >= 225 && propBLX1 >= paddle_pos && propBLX1 <= paddle_pos+35)||(propBLY2 >= 225 && propBLX2 >= paddle_pos && propBLX2 <= paddle_pos+35))
  {
    if (((propBLX1 >= paddle_pos && propBLX1 <= paddle_pos+7) || (propBLX1 >= paddle_pos+28 && propBLX1 <= paddle_pos+35))||((propBLX2 >= paddle_pos && propBLX2 <= paddle_po
    {
      if ((propBLX1 >= paddle_pos && propBLX1 <= paddle_pos+17)||(propBLX2 >= paddle_pos && propBLX2 <= paddle_pos+17))
      {
        direcx = -3;
      }
      else
      {
        direcx = 3;
      }

      if (direcy > 0)
      {
        direcy = 4;
      }
      else
      {
        direcy = -4;
      }

      direcy = direcy*-1;
      Serial.println("3,4");
    }
```



```
    else
    {
      if ((propBLX1 >= paddle_pos && propBLX1 <= paddle_pos+17)||(propBLX2 >= paddle_pos && propBLX2 <= paddle_pos+17))
      {
        direcx = -4;
      }
      else
      {
        direcx = 4;
      }

      if (direcy > 0)
      {
        direcy = 3;
      }
      else
      {
        direcy = -3;
      }
      direcy = direcy*-1;
      Serial.println("4,3");
    }
  }

  if((propBLY1 <= 10)||(propBLY2 <= 10)) // upper wall
  {
    direcy = direcy*-1;
  }

  if((propBLX1 <= 5 || propBLX1 >= 235)||(propBLX2 <= 5 || propBLX2 >= 235)) //side walls
  {
    direcx = direcx*-1;
  }

  ballX = ballX + direcx;
  ballY = ballY + direcy;

  if((propBLY1 > 240)||(propBLY2 > 240))
  {
    gameOver();
    return false;
  }
  else
  {
    return true;
  }
}
```

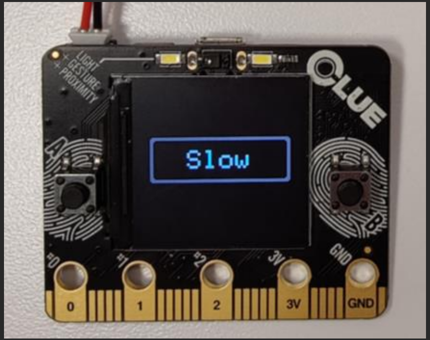
Bounds Function

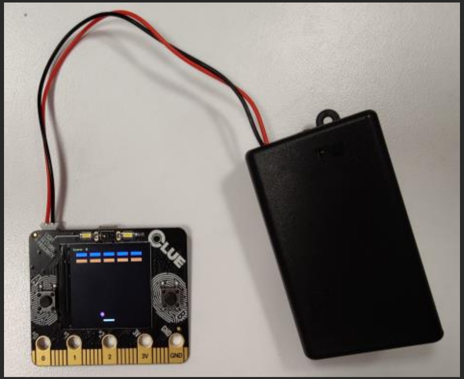
Coding Problems

- Game mechanics
  - Having x axis and y axis collisions with the same block
    - Still occurring
- C functions only having one return
  - Solved by using global variables
- Minimizing redraw flicker and speed issues
  - Solved by setting only the part that needs erased to black pixels
    - Example: setting the ball's old location to black pixels then redrawing

Game Constraints

- Limited visual size
  - Limited number of blocks
    - Much fewer than Python version
  - Limited playtime
- Small buttons
  - Not as intuitive as joystick
- Limited Replay value
  - Same pattern of blocks
  - No new blocks generated





Status

- Working Prototype: Completed
- More Maps: In Progress...
- Side Collisions: In Progress...
- Squashing Bugs: In Progress...
- Timer: In Progress...
- Ball speed increase: In Progress...
- Randomized Blocks: In Progress...

References

- https://www.coolmathgames.com/sites/default/files/Atari%20Breakout%20OG%20Image.png
- https://blog.sarasotabayclub.net/video-games-can-help-seniors

Questions??

Demonstration