Grab What's Green

EK 131 E2

Nafis Abeer and Hannah Gold

<u>Motivation behind the Game</u>

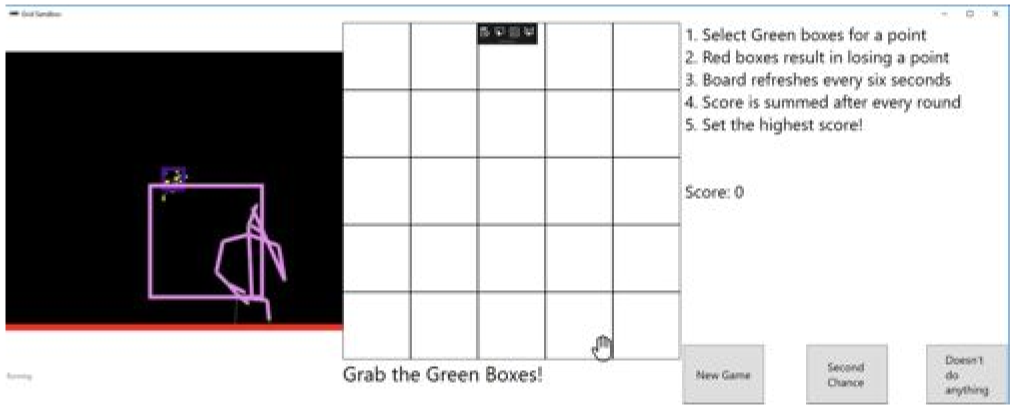
When thinking of an original game, we were first drawn to the idea of a "Whack-a-Mole", where moles would randomly pop up from a set of holes and players have to quickly react and whack the mole before it goes back into the hole; and there would a penalty for hitting something that's not a mole (in some versions). We modified the game and used green spaces instead of the moles and we used red spaces as the penalty. Instead of randomly having the green boxes pop up one at a time, we present users with a 5 by 5 grid of randomly selected red and green boxes and allow the user six seconds to react to a new grid and grab as many green boxes as they can. The player would know if they successfully grabbed a box based on the fact that a box would turn white after its been grabbed. The game goes on for five rounds and the player is presented with a newly randomized grid for each round. The name of our game is *Grab what's Green*. The second part of our game was actually a mistake we made by not re-randomizing the board when starting a new game, which led to the placement of the green and red and white boxes to remain the same after the first five rounds finished.

<u>*Grab what's Green* Game Objective</u>

Our game requires a certain degree of dexterity, since one does not want to hit the incorrect box. The parts of the game are simple:

1. Start with an empty grid with instructions for the game and the options "New Game" and "Second Chance".

2. Pressing new game will fill each box with either red or green and you have to close your hand over as many green boxes as you can.

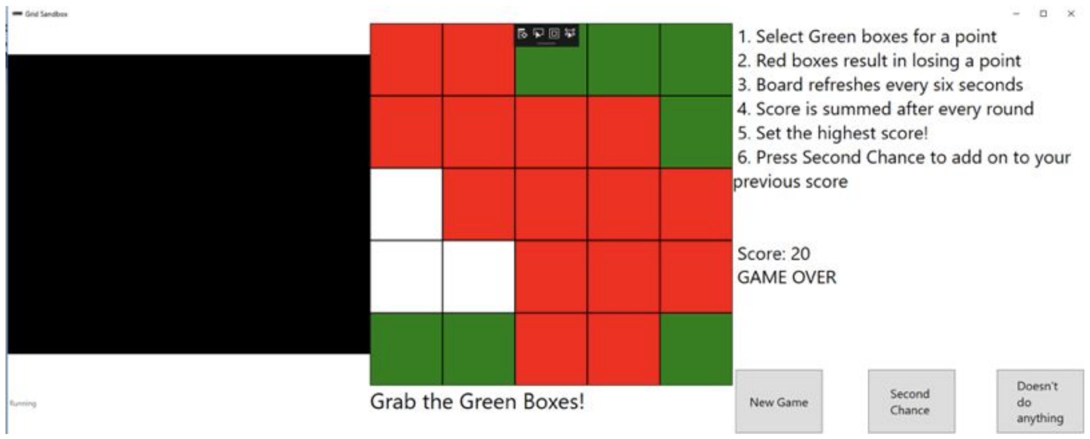


1. Select Green boxes for a point
2. Red boxes result in losing a point
3. Board refreshes every six seconds
4. Score is summed after every round
5. Set the highest score!

Score: 0

Grab the Green Boxes!

3. You have 6 seconds before the board refreshes and a new set of red and green boxes is presented to you.
4. Close your hand over the green spaces to gain a point.
5. Close your hand over the red spaces and you will lose a point.
6. For every box you close your hand over will change to white.
7. The game will have 5 different grids (making the game last a total of 30 seconds).
8. Score is consistently displayed.
9. When the 30 seconds are up the game stops and displays "GAME OVER" and an additional instruction in the instruction section of the game.



1. Select Green boxes for a point
2. Red boxes result in losing a point
3. Board refreshes every six seconds
4. Score is summed after every round
5. Set the highest score!
6. Press Second Chance to add on to your previous score

Score: 20
GAME OVER

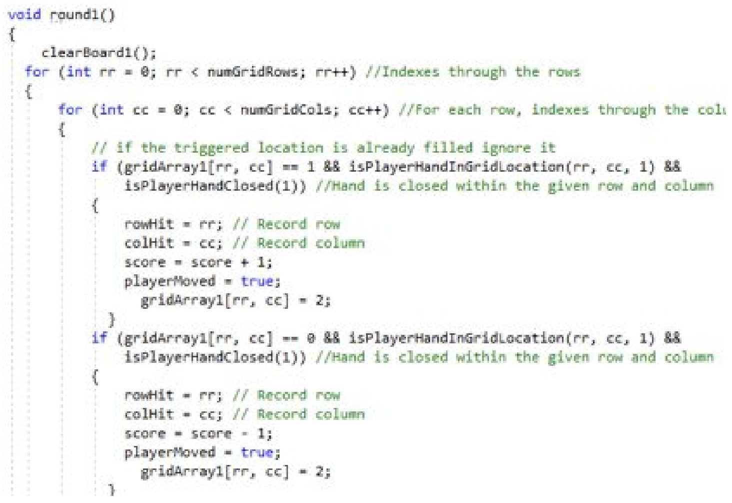
Grab the Green Boxes!

This is for the normal game; however, we have included a part two to the game which we like to call "Second Chance." This allows a player to go back to the grids which they had initially set during the first five rounds and have the ability to get more points without resetting the positions of the green and red boxes that was chosen for the first five rounds. The score would add on to whatever the score was set to be after the first five rounds and the white boxes from the first five rounds would hold its spot after pressing second chance. This could allow users to see how many times they'd have to press "second chance" in order to select all the green boxes set by the first five rounds and also how accurately they can select the green boxes based on the fact that every time a user selects a red box in the process, their final score would be lowered. For example, randomizing the board for the first five rounds would mean randomizing the color of 125 different boxes and we are allowed to assume that 50 percent of the boxes would be green meaning around 62.5 boxes would be green. Now, a player could possibly achieve a score of 62 or 63 by repeatedly pressing second chance and attempting to grab all the possible green boxes but a less skilled player would grab red boxes in the process and lower the possibility of them achieving a score close to 62. Also, a more skilled player could achieve a score of 62 by pressing second chance the least amount of time. This component of the game is what tests a player's dexterity.

How the Program Works

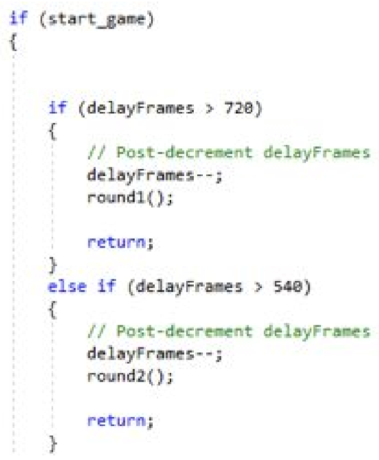
- In order for the game to begin the "New Game" button must first be clicked. The button essentially works by setting the delay frames to 900 and score to zero and also it runs the "RandGridArray" (the functions of which is explained later) function. It also sets a boolean variable called "start_game" to be true which initiates the process of running through the if statements. The reason why the delay frames is at 900 is because over the course of 5 rounds (5 grids filled randomly with red or green boxes), delay frames constantly decrements by 1 and the delay will initiate a new round every 180 frames (a total of 6 seconds since the student work section is called 30 times per second).

- We initialized 5 arrays called “gridArray1, gridArray2…” Calling the function “RandGridArray”, generates 25 * 5 random binary numbers, one for each of the elements in the arrays. The elements represent each of the boxes visible to the player. Then a different set of functions “clearBoard1, clearBoard2…” uses a “for” loop, that goes over all the boxes paints them either red if it is a “0” or green if it is a “1.” The round functions call for the corresponding “clearBoard” function to randomize the colors of the grid array first. It then loops through each element of the corresponding grid array and tracks the players hand motion at the same time. If a player’s hand closes over a certain element, the program recognizes one of the if-statement to be true and therefore, the player is either rewarded a point or has a point taken away based on the number (or color) of the element that the player closed their hand over. The value of the element then changes to 2 and the function that sets the colors of the grid array changes the selected box to white.



```
void round1()
{
    clearBoard1();
  for (int rr = 0; rr < numGridRows; rr++) //Indexes through the rows
  {
      for (int cc = 0; cc < numGridCols; cc++) //For each row, indexes through the col
      {
          // if the triggered location is already filled ignore it
          if (gridArray1[rr, cc] == 1 && isPlayerHandInGridLocation(rr, cc, 1) &&
              isPlayerHandClosed(1)) //Hand is closed within the given row and column
          {
              rowHit = rr; // Record row
              colHit = cc; // Record column
              score = score + 1;
              playerMoved = true;
                gridArray1[rr, cc] = 2;
            }
          if (gridArray1[rr, cc] == 0 && isPlayerHandInGridLocation(rr, cc, 1) &&
              isPlayerHandClosed(1)) //Hand is closed within the given row and column
          {
              rowHit = rr; // Record row
              colHit = cc; // Record column
              score = score - 1;
              playerMoved = true;
                gridArray1[rr, cc] = 2;
            }
```

- When a player first clicks “New Game”, delay frames is 900 or greater than 720 so the first condition is met and the function for round 1 is ran. The value of delay frames begins to decrease till it reaches 720 at which point it is greater than 540 and the next condition is met so the next round

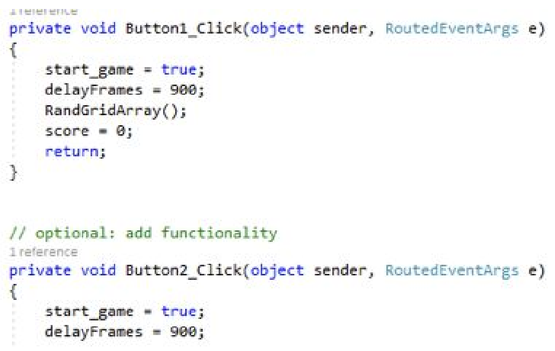


```
if (start_game)
{


    if (delayFrames > 720)
    {
        // Post-decrement delayFrames
        delayFrames--;
        round1();

        return;
    }
    else if (delayFrames > 540)
    {
        // Post-decrement delayFrames
        delayFrames--;
        round2();

        return;
    }
```

function is ran and so on. Once delay frames reaches a value of zero, the instruction texts function is rerun, and this time it includes a text that reads “GAME OVER” and informs the player that they can restart from round 1 with the same grid array and score by pressing the “Second Chance” button.

Encountered Difficulties

- We were having trouble thinking of a way to randomize the colors of the grid every six seconds and we solved it by splitting the grid up into five, one for every six second period.
- Figuring out how the player movement would constantly be tracked was kind of a challenge because the tic tac toe game had player moves based on turns while our player move was constant but the fact that the student works function runs at thirty frames per second helped solve that problem because any time a player closed their hand, the program would instantly react to it because of how rapidly it executes.
- We were having trouble with restarting the game after finishing the game once by pressing new game because the board would not refresh but we solved that by making delay frames equal 900 every time new game was pressed.



```
1 reference
private void Button1_Click(object sender, RoutedEventArgs e)
{
    start_game = true;
    delayFrames = 900;
    RandGridArray();
    score = 0;
    return;
}

// optional: add functionality
1 reference
private void Button2_Click(object sender, RoutedEventArgs e)
{
    start_game = true;
    delayFrames = 900;
```

Future Enhancements

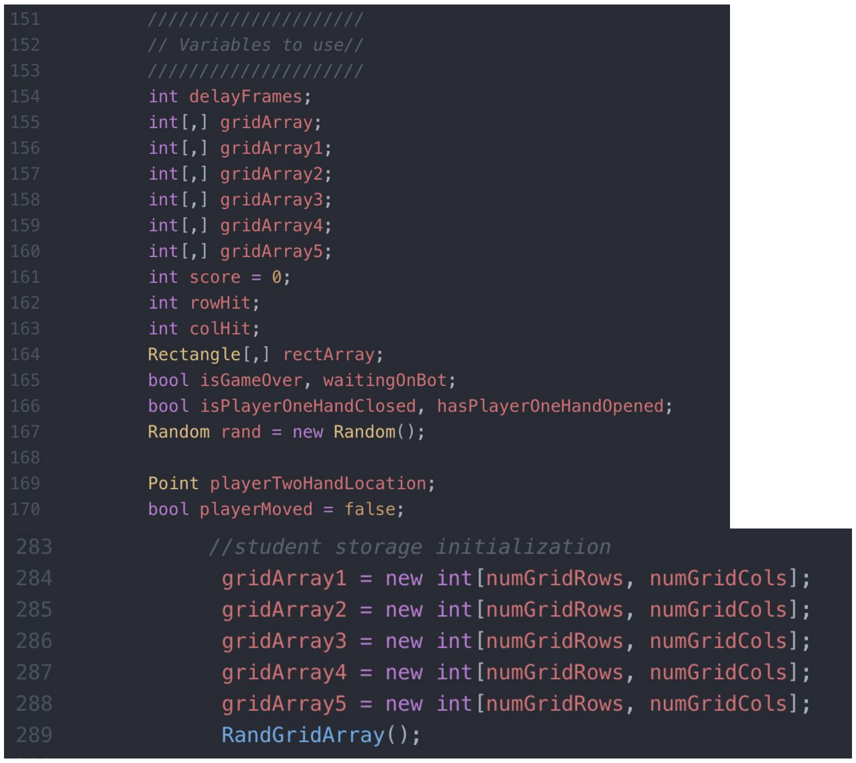
- We already attempted to introduce different difficulties into the game by increasing/decreasing the surface area and number of squares as more squares would mean smaller squares which present more of a challenge to the player.

- If studentwork () ran at a higher frame rate, our game would run smoother because the program would react to the player's hand motions and award/take-away points more efficiently.
- We initially thought we could make a two player game where the second player's objective would be to grab all the red boxes and avoid the green ones.
- Given more time, we could have set up a file where all the scores would be recorded to, and the program would read the top five scores set by any of the players at the conclusion of the game.

References

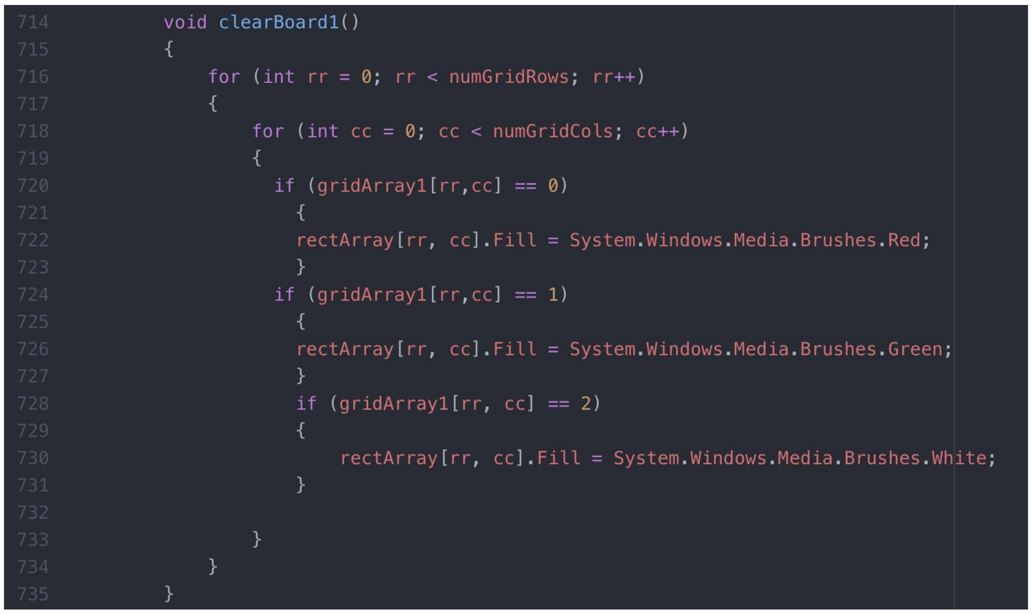
- https://www.tutorialspoint.com/csharp/index.htm

Appendix

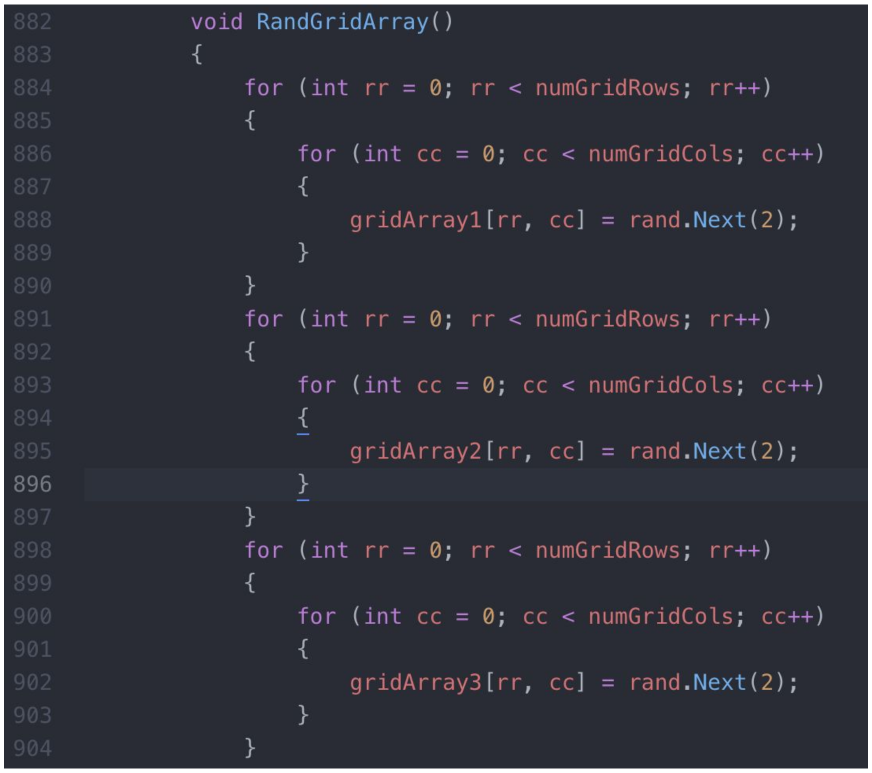


```
        ////////////////////
        // Variables to use//
        ////////////////////
        int delayFrames;
        int[,] gridArray;
        int[,] gridArray1;
        int[,] gridArray2;
        int[,] gridArray3;
        int[,] gridArray4;
        int[,] gridArray5;
        int score = 0;
        int rowHit;
        int colHit;
        Rectangle[,] rectArray;
        bool isGameOver, waitingOnBot;
        bool isPlayerOneHandClosed, hasPlayerOneHandOpened;
        Random rand = new Random();

        Point playerTwoHandLocation;
        bool playerMoved = false;
```

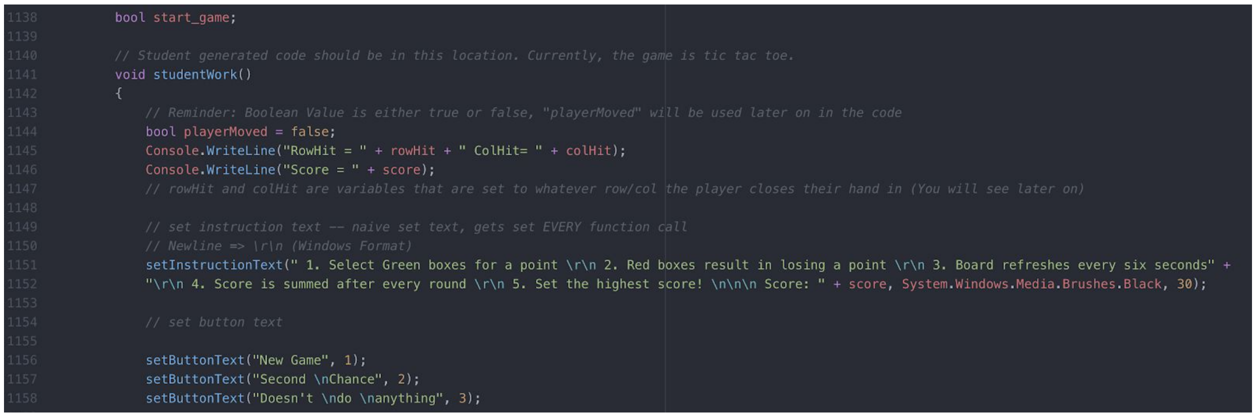
```
            //student storage initialization
             gridArray1 = new int[numGridRows, numGridCols];
             gridArray2 = new int[numGridRows, numGridCols];
             gridArray3 = new int[numGridRows, numGridCols];
             gridArray4 = new int[numGridRows, numGridCols];
             gridArray5 = new int[numGridRows, numGridCols];
             RandGridArray();
```



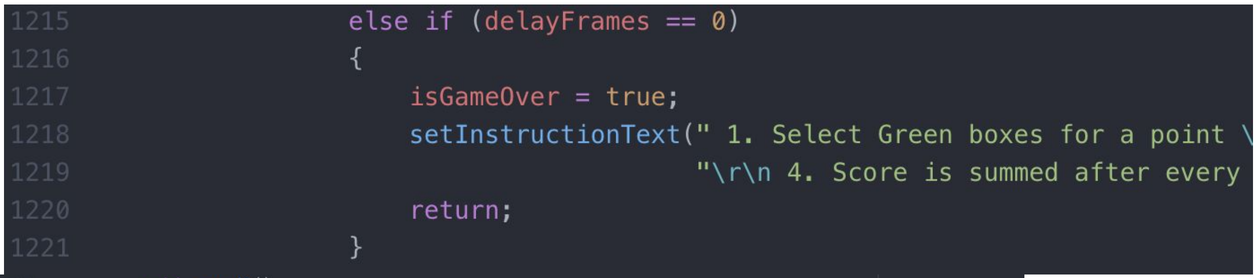
```
        void clearBoard1()
        {
            for (int rr = 0; rr < numGridRows; rr++)
            {
                for (int cc = 0; cc < numGridCols; cc++)
                {
                  if (gridArray1[rr,cc] == 0)
                    {
                    rectArray[rr, cc].Fill = System.Windows.Media.Brushes.Red;
                    }
                  if (gridArray1[rr,cc] == 1)
                    {
                    rectArray[rr, cc].Fill = System.Windows.Media.Brushes.Green;
                    }
                    if (gridArray1[rr, cc] == 2)
                    {
                        rectArray[rr, cc].Fill = System.Windows.Media.Brushes.White;
                    }

                }
            }
        }
```



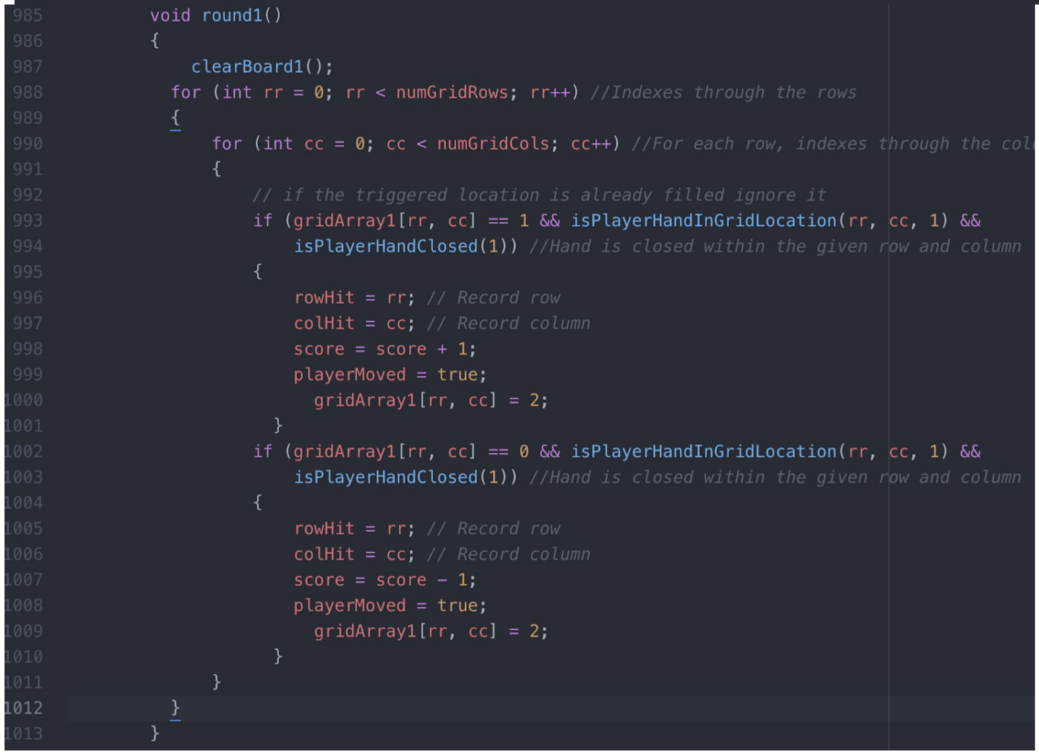
```
        void RandGridArray()
        {
            for (int rr = 0; rr < numGridRows; rr++)
            {
                for (int cc = 0; cc < numGridCols; cc++)
                {
                    gridArray1[rr, cc] = rand.Next(2);
                }
            }
            for (int rr = 0; rr < numGridRows; rr++)
            {
                for (int cc = 0; cc < numGridCols; cc++)
                {
                    gridArray2[rr, cc] = rand.Next(2);
                }
            }
            for (int rr = 0; rr < numGridRows; rr++)
            {
                for (int cc = 0; cc < numGridCols; cc++)
                {
                    gridArray3[rr, cc] = rand.Next(2);
                }
            }
```



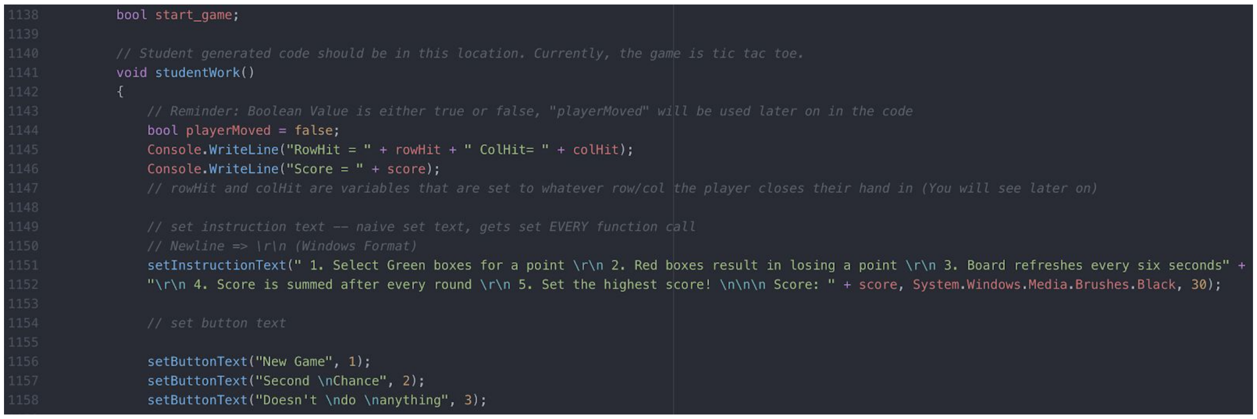
```
        bool start_game;

        // Student generated code should be in this location. Currently, the game is tic tac toe.
        void studentWork()
        {
            // Reminder: Boolean Value is either true or false, "playerMoved" will be used later on in the code
            bool playerMoved = false;
            Console.WriteLine("RowHit = " + rowHit + " ColHit= " + colHit);
            Console.WriteLine("Score = " + score);
            // rowHit and colHit are variables that are set to whatever row/col the player closes their hand in (You will see later on)

            // set instruction text -- naive set text, gets set EVERY function call
            // Newline => \r\n (Windows Format)
            setInstructionText(" 1. Select Green boxes for a point \r\n 2. Red boxes result in losing a point \r\n 3. Board refreshes every six seconds" +
            "\r\n 4. Score is summed after every round \r\n 5. Set the highest score! \n\n\n Score: " + score, System.Windows.Media.Brushes.Black, 30);

            // set button text

            setButtonText("New Game", 1);
            setButtonText("Second \nChance", 2);
            setButtonText("Doesn't \ndo \nanything", 3);
```



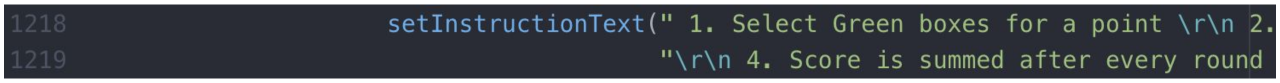
```
                else if (delayFrames == 0)
                {
                    isGameOver = true;
                    setInstructionText(" 1. Select Green boxes for a point \
                                        "\r\n 4. Score is summed after every 

                    return;
                }
```



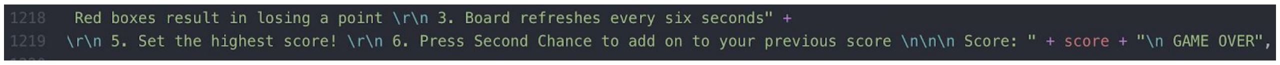
```
        void round1()
        {
            clearBoard1();
          for (int rr = 0; rr < numGridRows; rr++) //Indexes through the rows
          {
              for (int cc = 0; cc < numGridCols; cc++) //For each row, indexes through the col
              {
                  // if the triggered location is already filled ignore it
                  if (gridArray1[rr, cc] == 1 && isPlayerHandInGridLocation(rr, cc, 1) &&
                      isPlayerHandClosed(1)) //Hand is closed within the given row and column
                  {
                      rowHit = rr; // Record row
                      colHit = cc; // Record column
                      score = score + 1;
                      playerMoved = true;
                        gridArray1[rr, cc] = 2;
                    }
                  if (gridArray1[rr, cc] == 0 && isPlayerHandInGridLocation(rr, cc, 1) &&
                      isPlayerHandClosed(1)) //Hand is closed within the given row and column
                  {
                      rowHit = rr; // Record row
                      colHit = cc; // Record column
                      score = score - 1;
                      playerMoved = true;
                        gridArray1[rr, cc] = 2;
                    }
              }
          }
        }
```



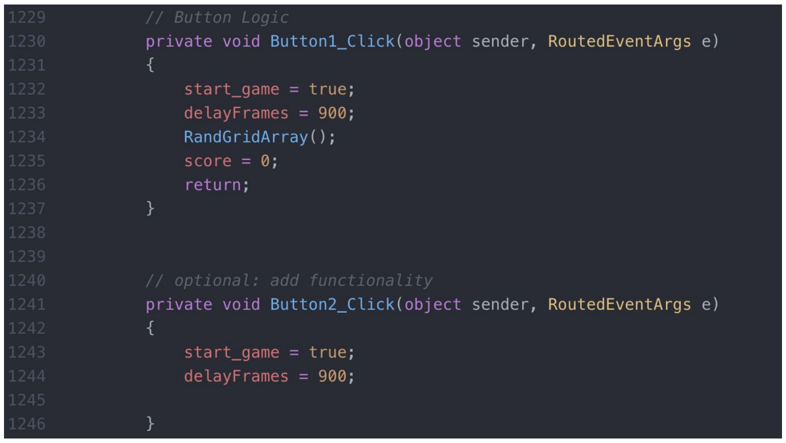
```
        bool start_game;

        // Student generated code should be in this location. Currently, the game is tic tac toe.
        void studentWork()
        {
            // Reminder: Boolean Value is either true or false, "playerMoved" will be used later on in the code
            bool playerMoved = false;
            Console.WriteLine("RowHit = " + rowHit + " ColHit= " + colHit);
            Console.WriteLine("Score = " + score);
            // rowHit and colHit are variables that are set to whatever row/col the player closes their hand in (You will see later on)

            // set instruction text -- naive set text, gets set EVERY function call
            // Newline => \r\n (Windows Format)
            setInstructionText(" 1. Select Green boxes for a point \r\n 2. Red boxes result in losing a point \r\n 3. Board refreshes every six seconds" +
            "\r\n 4. Score is summed after every round \r\n 5. Set the highest score! \n\n\n Score: " + score, System.Windows.Media.Brushes.Black, 30);

            // set button text

            setButtonText("New Game", 1);
            setButtonText("Second \nChance", 2);
            setButtonText("Doesn't \ndo \nanything", 3);
```



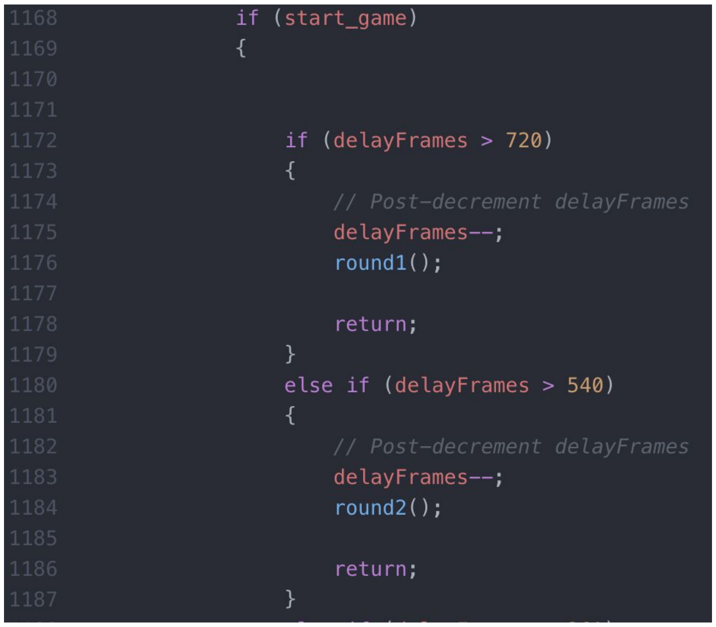
```
                    setInstructionText(" 1. Select Green boxes for a point \r\n 2.
                                   "\r\n 4. Score is summed after every round
```



```
 Red boxes result in losing a point \r\n 3. Board refreshes every six seconds" +
\r\n 5. Set the highest score! \r\n 6. Press Second Chance to add on to your previous score \n\n\n Score: " + score + "\n GAME OVER",
```



```
        // Button Logic
        private void Button1_Click(object sender, RoutedEventArgs e)
        {
            start_game = true;
            delayFrames = 900;
            RandGridArray();
            score = 0;
            return;
        }


        // optional: add functionality
        private void Button2_Click(object sender, RoutedEventArgs e)
        {
            start_game = true;
            delayFrames = 900;

        }
```



```
            if (start_game)
            {


                if (delayFrames > 720)
                {
                    // Post-decrement delayFrames
                    delayFrames--;
                    round1();

                    return;
                }
                else if (delayFrames > 540)
                {
                    // Post-decrement delayFrames
                    delayFrames--;
                    round2();

                    return;
                }
```

```
        // optional: add functionality
        private void Button3_Click(object sender, RoutedEventArgs e)
        {


            /* int numGridRows = 4, numGridCols = 4;
            uniGrid.Rows = numGridRows; uniGrid.Columns = numGridCols;
            //rows left->right
            rectArray = new Rectangle[numGridRows, numGridCols];
            for (int rr = 0; rr < numGridRows; rr++)
            {
                for (int cc = 0; cc < numGridCols; cc++)
                {
                    rectArray[rr, cc] = new Rectangle()
                    {
                        Stroke = System.Windows.Media.Brushes.Black,
                        // Height = (int)(gridHeight / numGridRows),
                        //Width = (int)(gridWidth / numGridRows),
                    };
                    uniGrid.Children.Add(rectArray[rr, cc]);
                }
            }

            //student storage initialization
            gridArray1 = new int[numGridRows, numGridCols];
            gridArray2 = new int[numGridRows, numGridCols];
            gridArray3 = new int[numGridRows, numGridCols];
            gridArray4 = new int[numGridRows, numGridCols];
            gridArray5 = new int[numGridRows, numGridCols];
            RandGridArray();

            start_game = true;
            delayFrames = 900;
            RandGridArray();
            score = 0;
            */


        }
```