UCS505 Computer Graphics Project Report on

Mini Tetris – A User Interactive Game

End-Semester Evaluation

Submitted by:

GAUTAM THAREJA, 101816005, CSE1
&
ABHIMANYU PARASHAR, 101816011, CSE1

(BE Third Year)

Submitted to:

Ms. Kudratdeep Aulakh, Lab Teacher

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department TIET, Patiala

10th June, 2021

1 PROJECT OVERVIEW

1.1 Introduction of the Project

In this project, we have used several Computer Graphics Concepts to make an interactive game. We have taken inspiration from the famous game Tetris. In Tetris, the aim is to complete the rows with the shapes that fall down. The type of shape and its colour is chosen at random. In our project, what the user has to do to earn points, is to match the blocks falling down with the same colour blocks. The user can match the blocks in either vertical or horizontal direction (or both). This will result in those blocks disappearing and the user will get points. The blocks that are falling down are of random colour and at a random position on the horizontal line. To make things even more interesting, the speed at which the blocks fall down increases gradually as the game progresses.

- **Aim of the Game**
  The aim of MiniTetris is to match as many colour-blocks as possible and not let the stack of blocks reach the top. When that happens, the game is over and all the blocks turn to white colour indicating that the game is over.

- **Controls**
  The controls of the game are simple, use the right/left arrow keys to move the colour block in the horizontal direction. Use the down arrow key to increase the speed at which the block falls down. Press the "Home" key to restart the game.

- **Scoring**
  The total score at the end is the total number of same colour blocks that the user has connected throughout the game.

2 COMPUTER GRAPHICS CONCEPTS USED

2.1 Computer Graphics Concepts Used

In this project, we have made use of several Computer Graphics concepts. One of the most basic but most important concept is the plotting of a point on the window, using the glVertex2f() function. Each block is one point so we had to make sure that the correct colour block was plotted at the correct position on the window.

Since our project is based on colour matching of different blocks, another concept that we have used is reading the colour from the screen. This was done using the glReadPixels() function. It was used to read the colour of pixels on left, right and the bottom side.

The concept of using arrow keys to move the block has also been added. This is what makes the game user interactive.

Some important openGL functions used:

- glColor() - set the current colour by taking a pointer to an array that contains red, green, blue, and (sometimes) alpha values.
- glVertex() - specify a vertex by taking a pointer to an array of two, three, or four elements. The elements of a two-element array are x and y; of a three-element array, x, y, and z;
- glReadPixels() - read a block of pixels from the frame buffer. It takes multiple parameters like window coordinates, width, height., format, type. But we have only specified the window coordinates of the first pixel that is read from the frame buffer. This location is the lower left corner of a rectangular block of pixels.
- glutPostRedisplay() - marks the current window as needing to be redisplayed.
- glutSpecialFunc() - sets the special keyboard callback for the current window. Takes a callback function.
- glutIdleFunc() - sets the global idle callback.

3 USER DEFINED FUNCTIONS

3.1 User Defined Functions

- **drawCurrentBlock()**
  This Function draws the current block. The current block can either be a new block with random colour and horizontal position or an existing block which is falling down.

- **drawGrid()**
  This Function draws the complete grid of the game. Whenever the falling block settles at the bottom or over another block of different colour, it is added to the grid.

- **shiftDownByOne(…)**
  This Function decreases the height of specific tower of blocks by one. Also, it calls the checkSurroundings() function to see if the new tower has a matching colour block in its surrounding.

- **shiftDownByTwo(…)**
  This Function decreases the height of specific tower of blocks by two. Also, it calls the checkSurroundings() function to see if the new tower has a matching colour block in its surrounding.

- **checkBottom(…)**
  This Function checks whether the colour of the current block matches with its bottom block's colour or not. If matched, the score is updated, shiftDownByTwo() is called on the lower block and 1 is returned, else 0 is returned.

- **checkLeft(…)**
  This Function checks whether the colour of the current block matches with its left block's colour or not. If a match is found, the score is updated, shiftDownByOne() is called both for the current as well as the left tower and 1 is returned, else 0 is returned.

- **checkRight(…)**
  This Function checks whether the colour of the current block matches with its right block's colour or not. If a match is found, the score is updated, shiftDownByOne() is called both for the current as well as the right and 1 is returned, else 0 is returned.

- **checkSurroundings(…)**
  This Function calls above 3 functions, checkBottom(), checkLeft() and checkRight(), to check whether there is any block which has same colour as of the current block. If any side matches, the score is updated.

- **drop()**

  This Function simulates the animation for dropping of the block from the top by reducing the y coordinate of the current block. The gradual increase of speed is also handled by this function.

- **specialInput(…)**

  This Function contains the actions to be taken for the keyboard inputs:

  - Key “down” - Drops the block faster.
  - Key “right” - Shifts the block by one column to the right.
  - Key “left” - Shifts the block by one column to the left.
  - Key “home” - Restarts the game.

  It is called from the glutSpecialFunc() in main().

- **restart()**

  This functions initialized the ht[] array to the initial values and erases all the data on the grid. It is called when the “Home” key is pressed.

- **game()**

  It is the primary function which is called from the glutDisplayFunc() in main().It contains all the functionality of the game and draws all the parts of the game by using other different function.

- **myInit()**

  This function initializes the details such as point size, background colour, etc. The grid is also initialized in this function.

4 CODE

4.1 Libraries

```
#include<GL/glut.h>
#include<iostream>
#include <math.h>
#include<windows.h>
#include<vector>
#include<ctime>
#include<unordered_map>
#include<map>
```

4.2 Global Variables Used

```
int score = 0;
int ptsz = 43;
bool lt = false;
bool rt = false;
bool down = false;
bool r = false;

int y;
float rd, gr, bl;
map<pair<int, int>, vector<float>> grid;
int ht[9] = { 50,50,50,50,50,50,50,50,50 };
int colCount = 5;
float red[5] = { 1.0,1.0,1.0,0.45098,0.0117647 };
float green[5] = { 0.121569,0.588235,0.839216,0.8,0.25098 };
float blue[5] = { 0.0705882,0.1009804,0,0.231373,0.678431 };
int tow, towi, col;
int blockNo = 0;
bool newBlock = true;
bool gameOver = false;

//Function Declaration
void checkSurroundings(int[], float, float, float, int, int, int);
```

4.3 User Defined Functions

```
void drawCurrentBlock() {
      if (gameOver)
            return;

      if (newBlock) {
            cout << "Score = " << score << endl;
            cout << "drawing new block\n";
            blockNo++;
            y = 600;
            srand((unsigned)time(0) + blockNo);
            col = rand() % colCount;
            towi = rand() % 9;
            tow = (towi + 1) * 45;

            rd = red[col], gr = green[col], bl = blue[col];
```

```
			cout << tow << " " << y << endl;
			cout << rd << " " << gr << " " << bl << endl;
			glBegin(GL_POINTS);
			glColor3f(rd, gr, bl);
			glVertex2f(tow, y);
			glEnd();
			glFlush();
			newBlock = false;
		}
		else {
			//cout << "drawing current block\n";
			tow = (towi + 1) * 45;
			glBegin(GL_POINTS);
			glColor3f(rd, gr, bl);
			glVertex2f(tow, y);
			glEnd();
			glFlush();
		}
}

void drawGrid() {
		//cout << "drawing grid\n";
		if (gameOver)
			return;
		glBegin(GL_POINTS);
		for (int i = 45; i < 450; i += 45) {
			for (int j = 50; j < 600; j += 45) {
				float rd1 = grid[{i, j}][0];
				float gr1 = grid[{i, j}][1];
				float bl1 = grid[{i, j}][2];
				glColor3f(rd1, gr1, bl1);
				glVertex2f(i, j);
			}
		}
		glEnd();
		glFlush();
}

void shiftDownByOne(int ht[], int towi, int tow, int y) {
		float color1[3];
		for (int i = y; i < ht[towi]; i += 45) {
			glReadPixels(tow, i + 45, 1.0, 1.0, GL_RGB, GL_FLOAT, color1);
			grid[{tow, i}] = { color1[0], color1[1], color1[2] };
			if (color1[0] == 0 && color1[1] == 0 && color1[2] == 0)
				break;
		}

		drawGrid();

		for (int i = y; i < ht[towi]; i += 45) {
			glReadPixels(tow, i, 1.0, 1.0, GL_RGB, GL_FLOAT, color1);
			if (color1[0] == 0 && color1[1] == 0 && color1[2] == 0)
				break;
			checkSurroundings(ht, color1[0], color1[1], color1[2], towi,
tow, i);
		}
}

void shiftDownByTwo(int ht[], int towi, int tow, int y) {
	float color1[3];
	int count = 0;
	for (int i = y; i < ht[towi]; i += 45) {
		glReadPixels(tow, i + 2 * 45, 1.0, 1.0, GL_RGB, GL_FLOAT,
color1);
		grid[{tow, i}] = { color1[0], color1[1], color1[2] };
		if (color1[0] == 0 && color1[1] == 0 && color1[2] == 0) {
			count++;
			if (count == 2)
				break;
		}
	}

	drawGrid();

	for (int i = y; i < ht[towi]; i += 45) {
		glReadPixels(tow, i, 1.0, 1.0, GL_RGB, GL_FLOAT, color1);
		if (color1[0] == 0 && color1[1] == 0 && color1[2] == 0)
			break;
		checkSurroundings(ht, color1[0], color1[1], color1[2], towi,
tow, i);
	}
}

int checkBottom(int ht[], float rd, float gr, float bl, int towi, int tow,
int y) {
	if (y > 51) {
		float color[3];
		glReadPixels(tow, y - 45, 1.0, 1.0, GL_RGB, GL_FLOAT, color);

		color[0] = (int)(color[0] * 100);
		color[1] = (int)(color[1] * 100);
		color[2] = (int)(color[2] * 100);

		if (color[0] == (int)(rd * 100) && color[1] == (int)(gr * 100)
&& color[2] == (int)(bl * 100)) {
			cout << "bottom matched\n";
			score++;
			shiftDownByTwo(ht, towi, tow, y - 45);
			ht[towi] -= 45;
			/*if (ht[towi] < 50)
				ht[towi] = 50;*/
			return 1;
		}
	}
	return 0;
}
```

```
int checkRight(int ht[], float rd, float gr, float bl, int towi, int tow,
int y) {
      if (towi > 9) {
            return 0;
      }
      float color[3];
      glReadPixels(tow + 45, y, 1.0, 1.0, GL_RGB, GL_FLOAT, color);

      color[0] = (int)(color[0] * 100);
      color[1] = (int)(color[1] * 100);
      color[2] = (int)(color[2] * 100);

      if (color[0] == (int)(rd * 100) && color[1] == (int)(gr * 100) &&
color[2] == (int)(bl * 100)) {
            cout << "right matched\n";
            score++;
            shiftDownByOne(ht, towi, tow, y);
            shiftDownByOne(ht, towi + 1, tow + 45, y);
            ht[towi + 1] -= 45;
            /*if (ht[towi + 1] < 50)
                  ht[towi + 1] = 50;*/
            return 1;
      }
      return 0;
}

int checkLeft(int ht[], float rd, float gr, float bl, int towi, int tow,
int y) {
      if (towi < 0) {
            return 0;
      }
      float color[3];
      glReadPixels(tow - 45, y, 1.0, 1.0, GL_RGB, GL_FLOAT, color);

      color[0] = (int)(color[0] * 100);
      color[1] = (int)(color[1] * 100);
      color[2] = (int)(color[2] * 100);

      if (color[0] == (int)(rd * 100) && color[1] == (int)(gr * 100) &&
color[2] == (int)(bl * 100)) {
            cout << "left matched\n";
            score++;
            shiftDownByOne(ht, towi, tow, y);
            shiftDownByOne(ht, towi - 1, tow - 45, y);
            ht[towi - 1] -= 45;
            if (ht[towi - 1] < 50)
                  ht[towi - 1] = 50;
            return 1;
      }
      return 0;
}

void checkSurroundings(int ht[], float rd, float gr, float bl, int towi,
int tow, int y) {
	int bt = checkBottom(ht, rd, gr, bl, towi, tow, y);
	int rt = checkRight(ht, rd, gr, bl, towi, tow, y);
	int lt = checkLeft(ht, rd, gr, bl, towi, tow, y);

	if (bt || rt || lt) {
		score++;
		ht[towi] -= 45;
	}
}

void drop() {
	if (y > ht[towi]) {
		y--;
		for (int i = 0; i < 10000 - blockNo * 100; i++)
			Sleep(0.9);
		glutPostRedisplay();
	}
}

void specialInput(int key, int x, int y)
{
	switch (key) {
	case GLUT_KEY_DOWN:
		//cout << "from SpecialInput: down pressed\n";
		down = true;
		glutPostRedisplay();
		break;
	case GLUT_KEY_LEFT:
		//cout << "from SpecialInput: left pressed\n";
		lt = true;
		glutPostRedisplay();
		break;
	case GLUT_KEY_RIGHT:
		//cout << "from SpecialInput: right pressed\n";
		rt = true;
		glutPostRedisplay();
		break;
	case GLUT_KEY_HOME:
		//cout << "from SpecialInput: home pressed\n";
		r = true;
		glutPostRedisplay();
		break;
	}
}

void restart() {
	for (int i = 45; i < 450; i += 45) {
		ht[i / 45 - 1] = 50;
		for (int j = 50; j < 600; j += 45) {
			grid[{i, j}] = { 0.0,0.0,0.0 };
		}
	}
}
```

```
void game() {
	glClear(GL_COLOR_BUFFER_BIT);
	drawGrid();
	drawCurrentBlock();

	if (rt) {
		if (towi < 8 && ht[towi + 1] < y) {
			towi++;
			tow += 45;
		}
		rt = false;
	}
	if (lt) {
		if (towi > 0 && ht[towi - 1] < y) {
			towi--;
			tow -= 45;
		}
		lt = false;
	}
	if (down) {
		y -= 45;
		if (y < ht[towi]) {
			y = ht[towi];
		}
		down = false;
	}
	if (r) {
		cout << "\nDo you want to restart the game? [y/n]\n";
		char ans;
		cin >> ans;
		if (ans == 'y') {
			cout << "\n\nRestarting the game.\n";
			restart();
			newBlock = true;
			score = 0;
			blockNo = 0;
		}
		r = false;
	}

	if (y == ht[towi]) {
		cout << "\nprev tower height = " << ht[towi] << endl;
		cout << "block dropped\n";
		grid[{tow, ht[towi]}] = { rd,gr,bl };
		drawGrid();
		ht[towi] += 45;
		checkSurroundings(ht, rd, gr, bl, towi, tow, y);
		cout << "new tower height = " << ht[towi] << endl << endl;
		newBlock = true;
		glutPostRedisplay();
	}

	if (ht[towi] > 600) {
		cout << "Game Over!\n";
```

```
            cout << "Score = " << score << endl;
            glBegin(GL_POINTS);
            for (int i = 45; i < 450; i += 45) {
                  for (int j = 50; j < 600; j += 45) {
                        if (!(grid[{i, j}][0] == 0.0 && grid[{i, j}][1] ==
0.0 && grid[{i, j}][2] == 0.0)) {
                              glColor3f(1.0, 1.0, 1.0);
                              glVertex2f(i, j);
                        }
                  }
            }
            glEnd();
            glFlush();
            gameOver = true;
      }

}
```

4.4 Init Function

```
void myinit() {
      //initializing grid
      for (int i = 45; i < 450; i += 45) {
            for (int j = 50; j < 600; j += 45) {
                  grid[{i, j}] = { 0.0,0.0,0.0 };
            }
      }

      glClearColor(0.0, 0.0, 0.0, 0.0);
      glPointSize(ptsz);
      glMatrixMode(GL_PROJECTION);
      gluOrtho2D(0.0, 450, 0.0, 600);

}
```

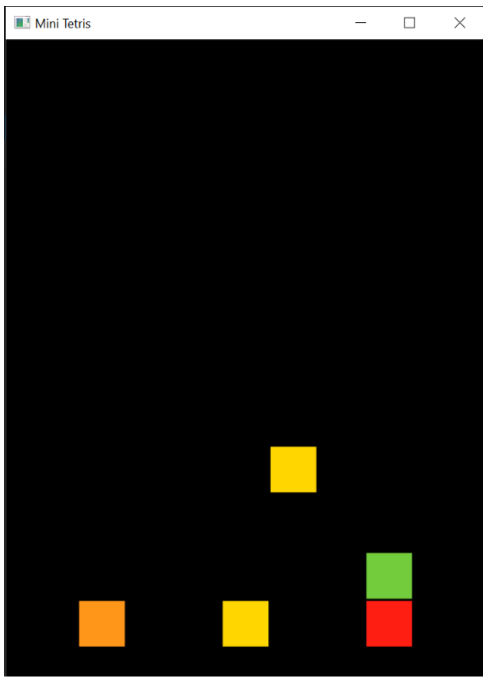
4.5 Main Function

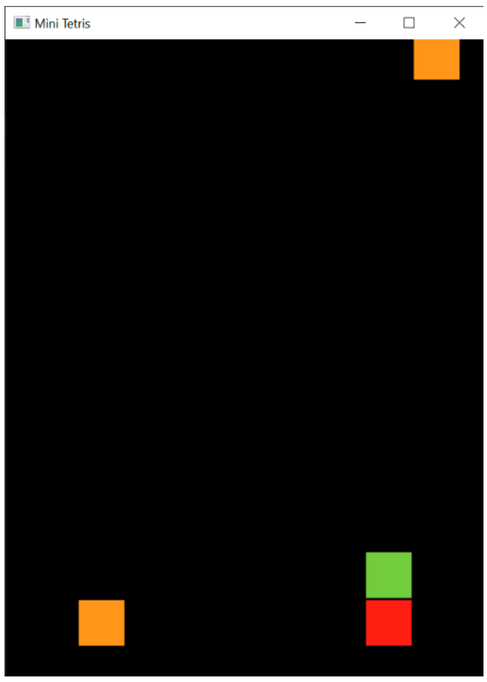
```
int main(int argc, char** argv) {
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(450, 600);
      glutInitWindowPosition(0, 0);
      glutCreateWindow("Mini Tetris");
      glutDisplayFunc(game);
      glutSpecialFunc(specialInput);
      glutIdleFunc(drop);
      myinit();
      glutMainLoop();

}
```

5 OUTPUT SCREENSHOTS

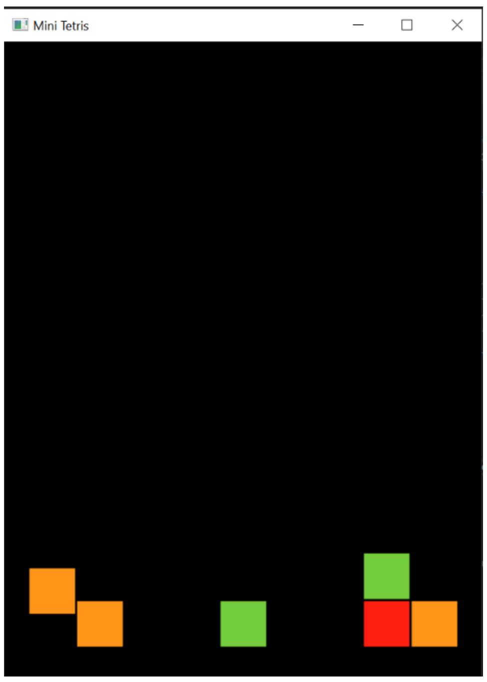
5.1 Output Screenshots

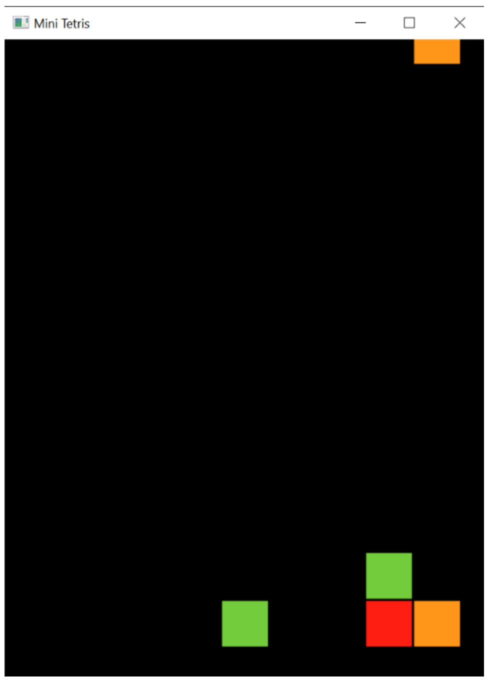
- **Case when the right-side block matches with the current block:**



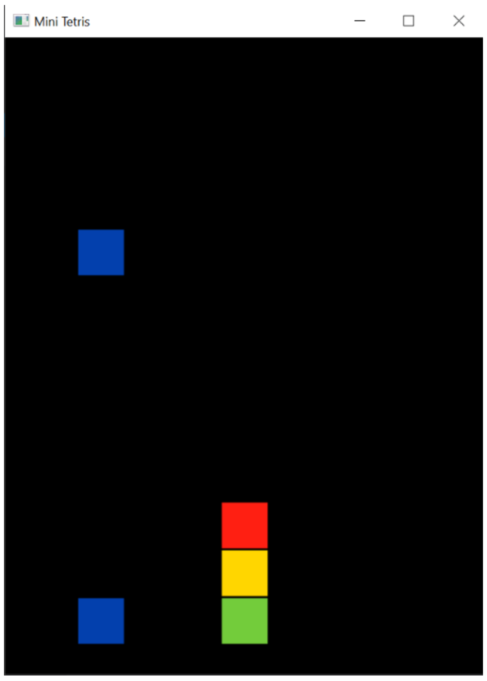


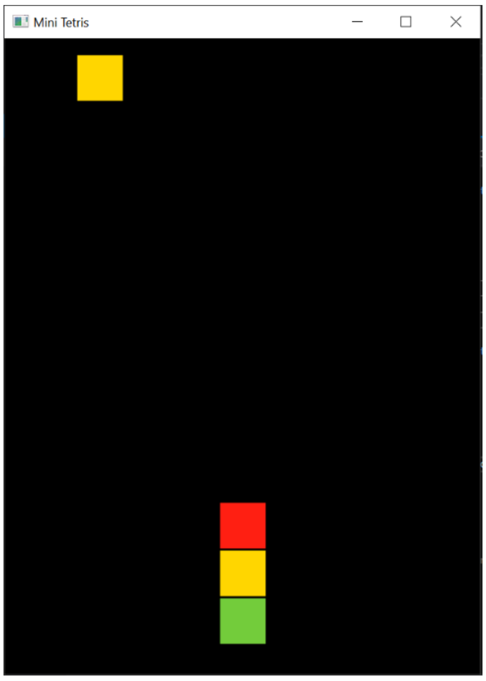
- **Case when the left-side block matches with the current block:**



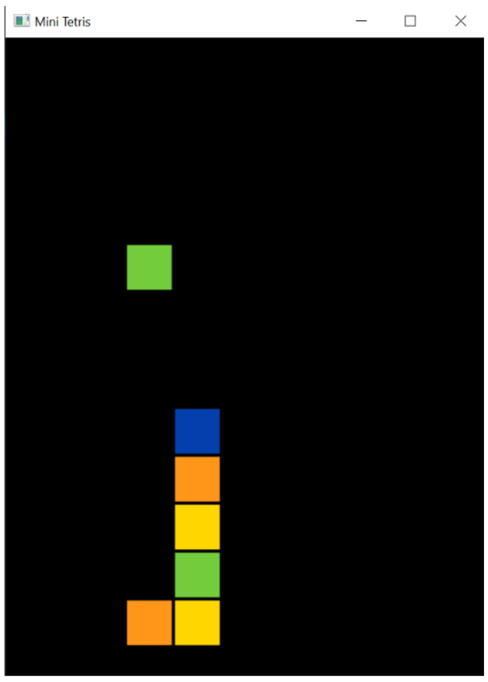


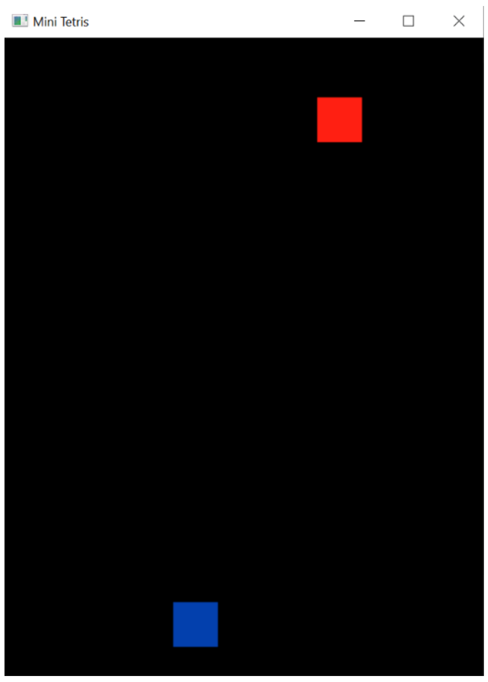
- **Case when the bottom-block matches with the current block:**



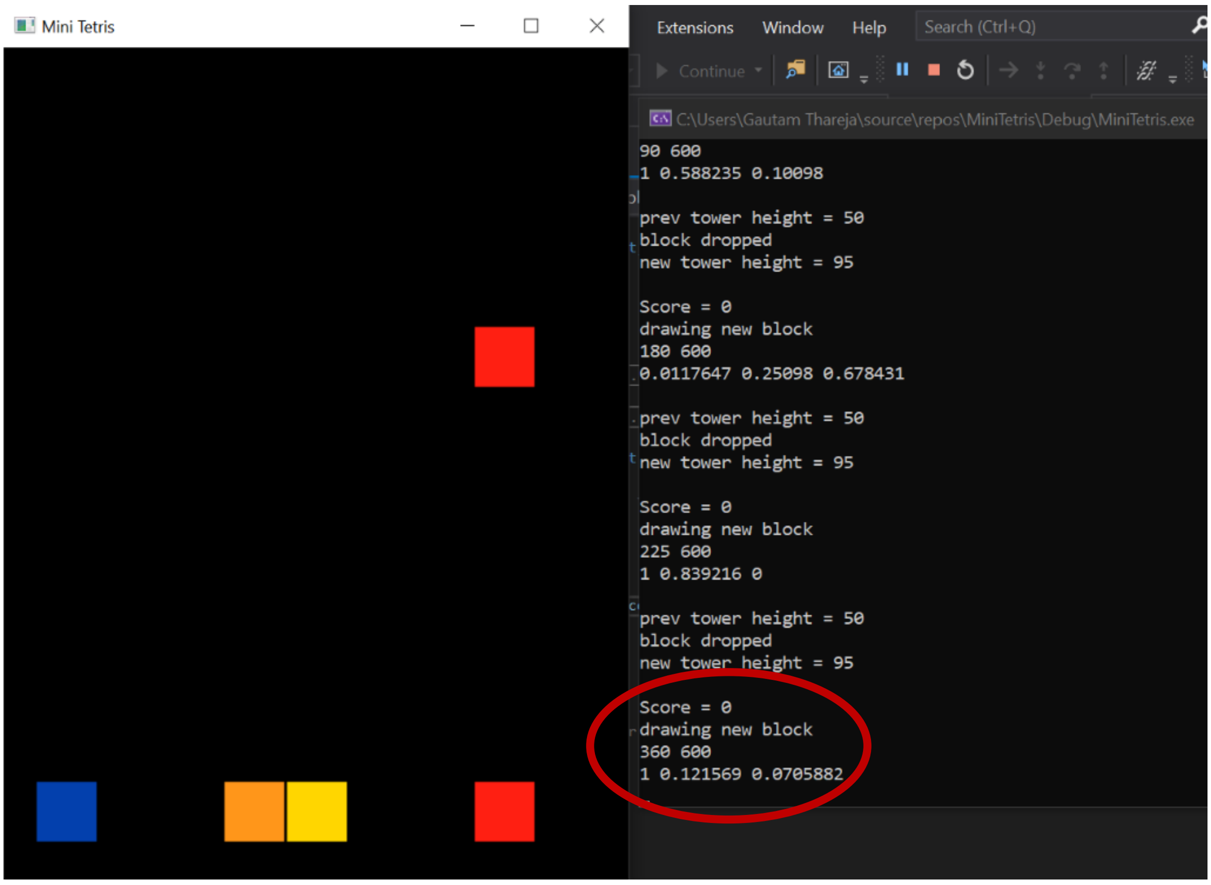


- **Multiple matchings of blocks consecutively:**

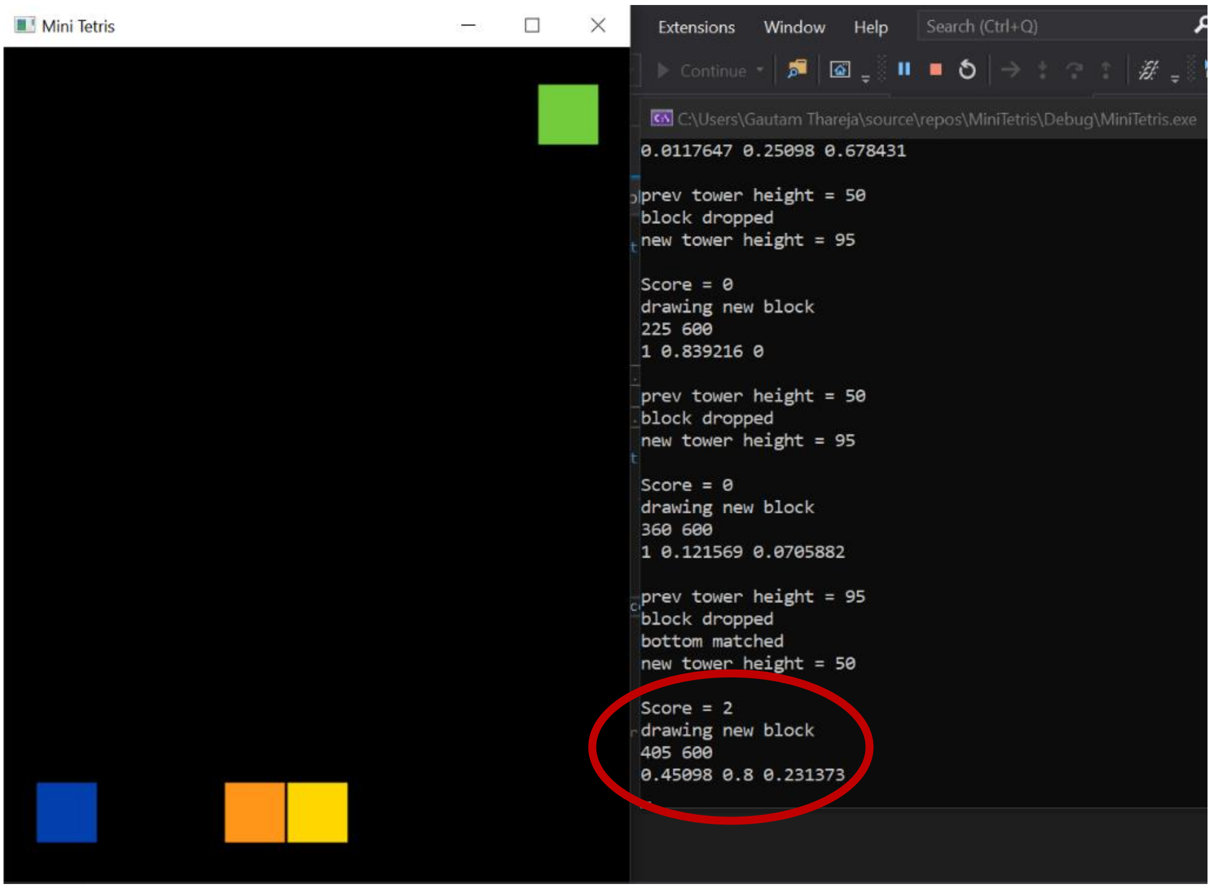




- **Scoring (Case when 2 blocks match)**

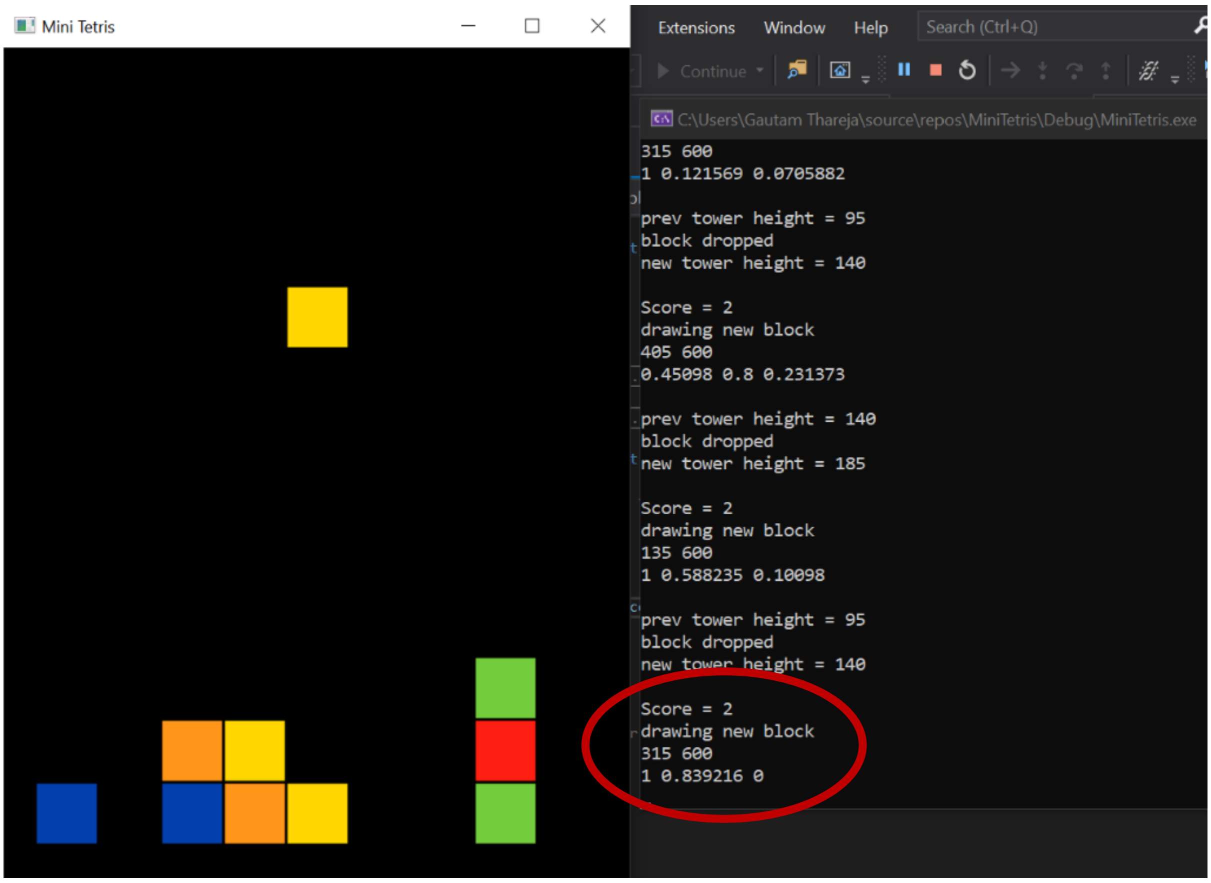

Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
90 600
1 0.588235 0.10098

prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
180 600
0.0117647 0.25098 0.678431

prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
225 600
1 0.839216 0

prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
360 600
1 0.121569 0.0705882
```

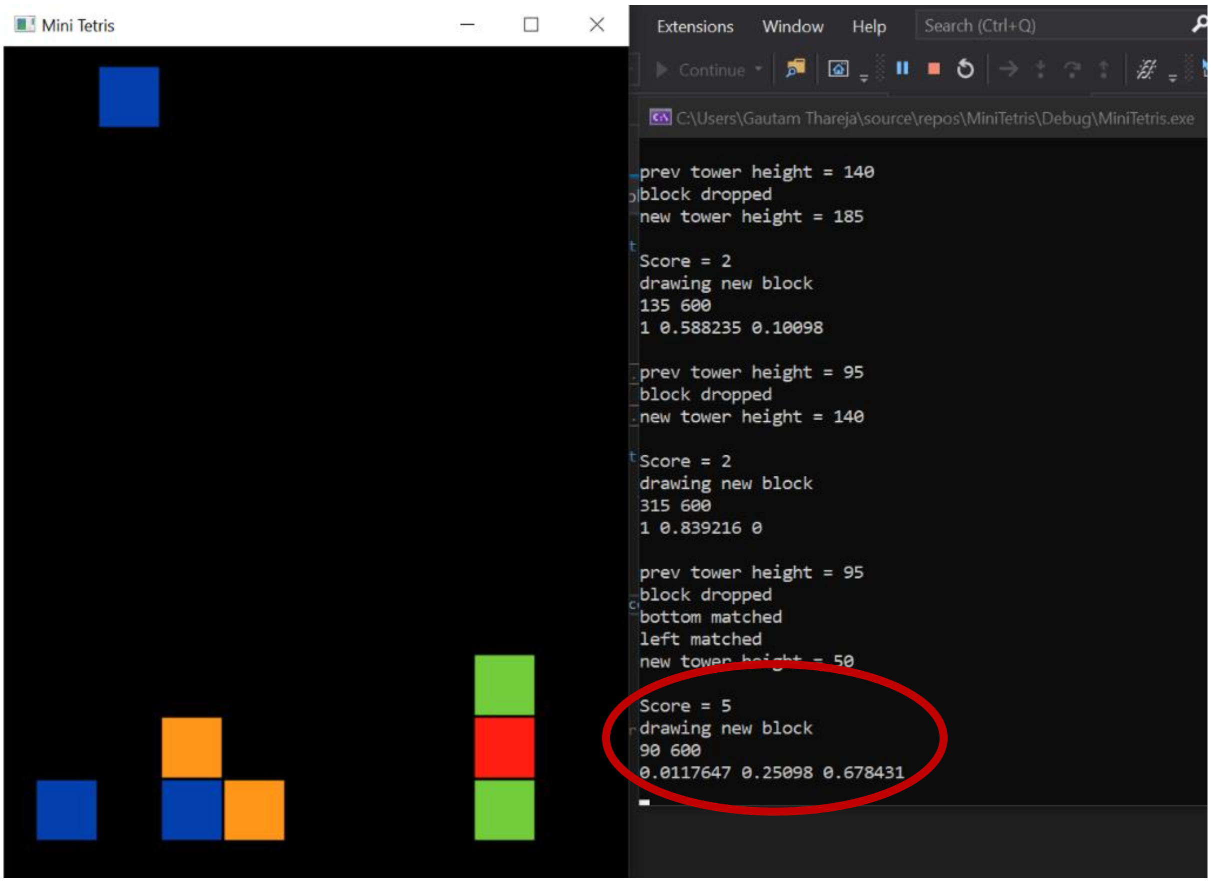

Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
0.0117647 0.25098 0.678431

prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
225 600
1 0.839216 0

prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
360 600
1 0.121569 0.0705882

prev tower height = 95
block dropped
bottom matched
new tower height = 50

Score = 2
drawing new block
405 600
0.45098 0.8 0.231373
```

- **Scoring (Case when 3 blocks match)**

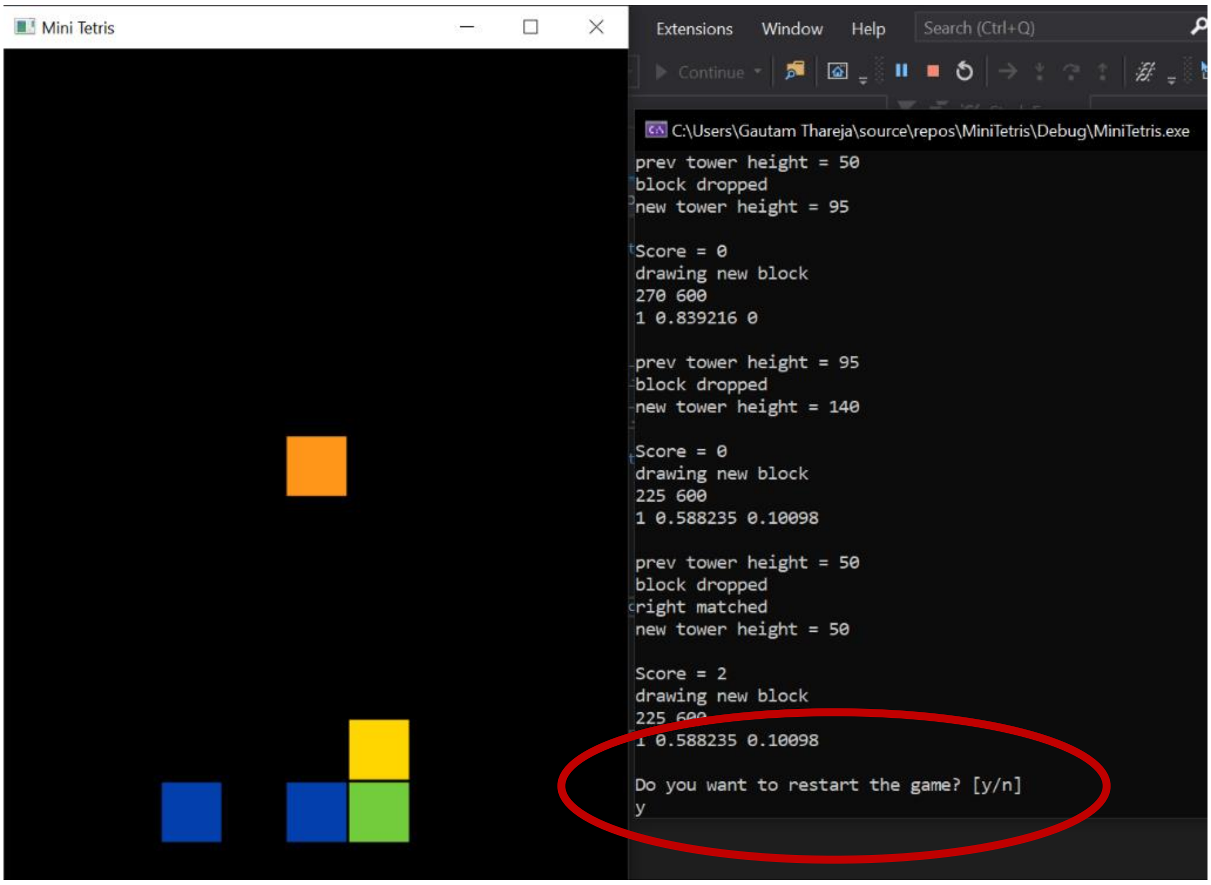

Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
315 600
1 0.121569 0.0705882

prev tower height = 95
block dropped
new tower height = 140

Score = 2
drawing new block
405 600
0.45098 0.8 0.231373

prev tower height = 140
block dropped
new tower height = 185

Score = 2
drawing new block
135 600
1 0.588235 0.10098

prev tower height = 95
block dropped
new tower height = 140

Score = 2
drawing new block
315 600
1 0.839216 0
```

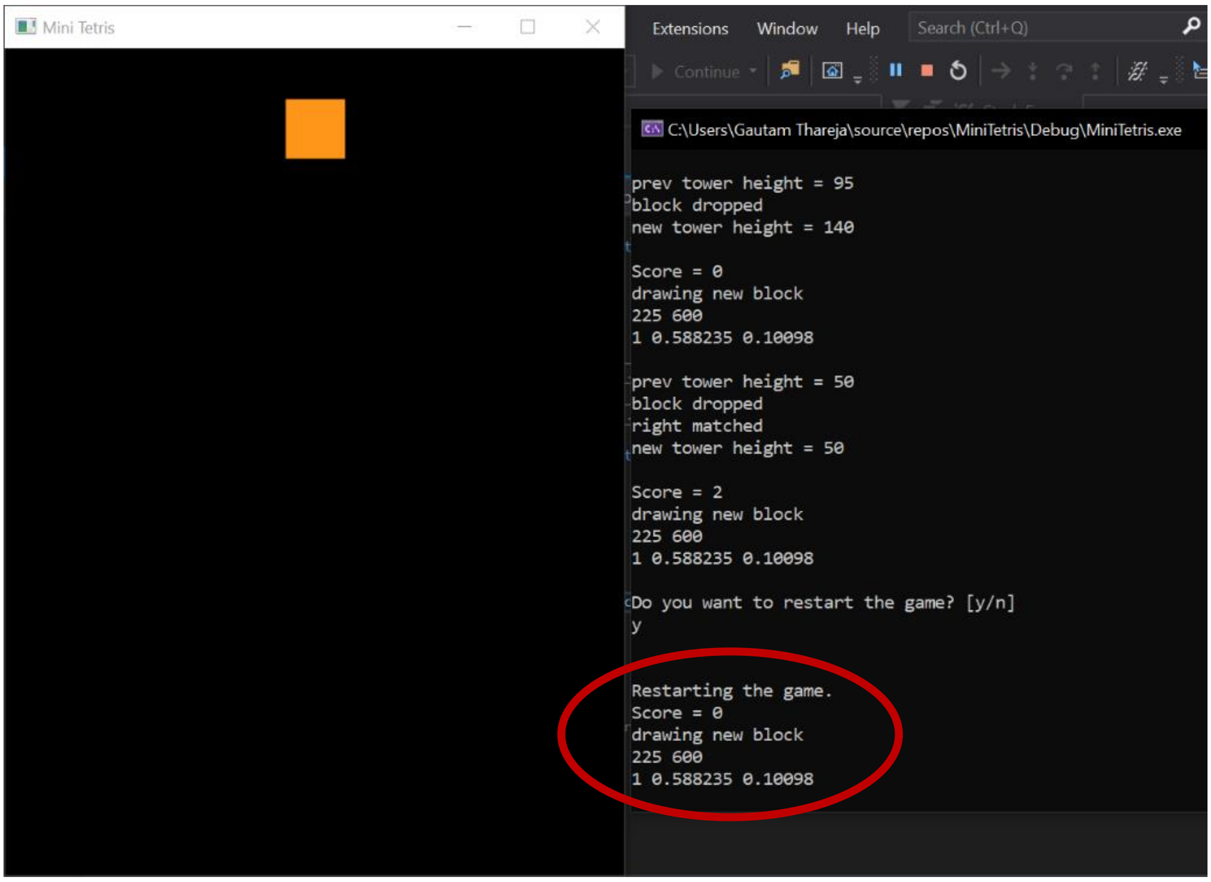

Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
prev tower height = 140
block dropped
new tower height = 185

Score = 2
drawing new block
135 600
1 0.588235 0.10098

prev tower height = 95
block dropped
new tower height = 140

Score = 2
drawing new block
315 600
1 0.839216 0

prev tower height = 95
block dropped
bottom matched
left matched
new tower height = 50

Score = 5
drawing new block
90 600
0.0117647 0.25098 0.678431
```

- **Case when the game restarts:**

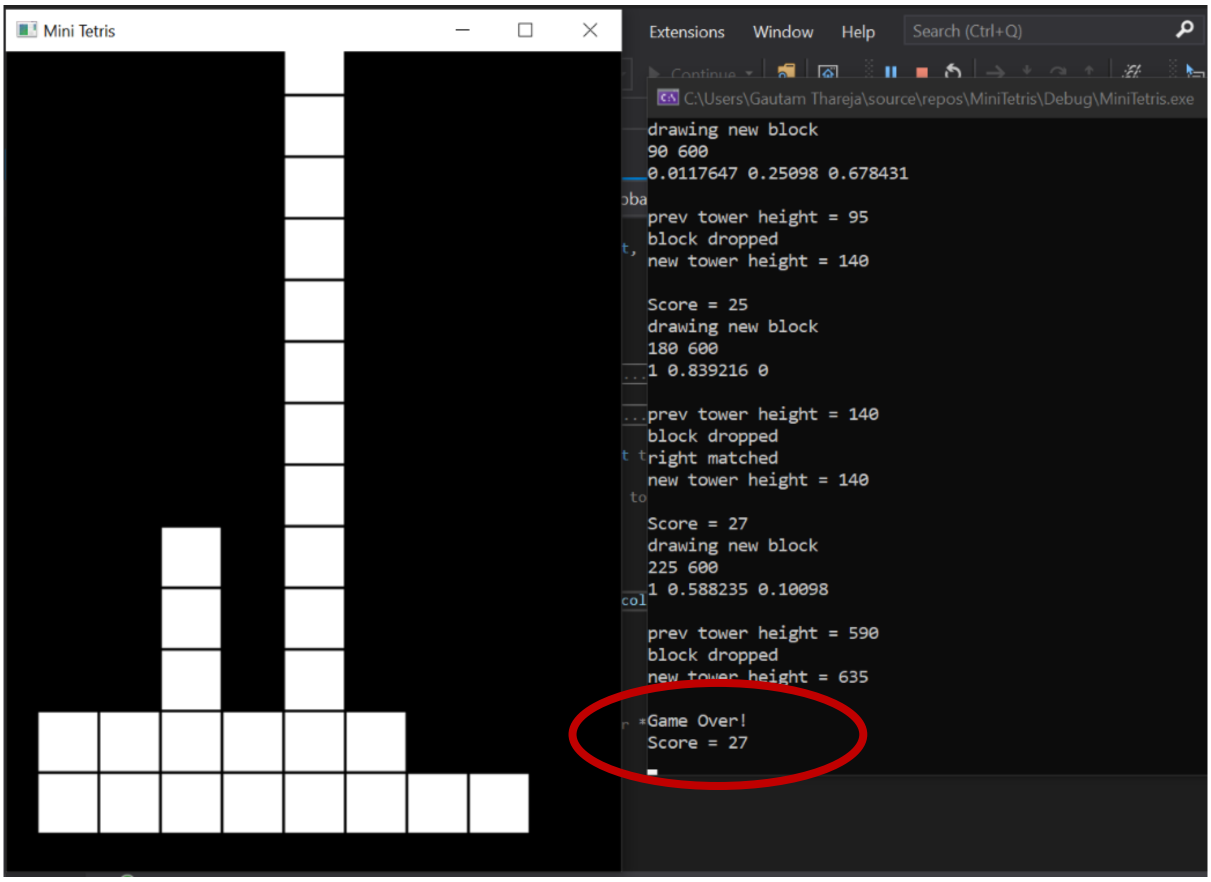

Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
prev tower height = 50
block dropped
new tower height = 95

Score = 0
drawing new block
270 600
1 0.839216 0

prev tower height = 95
block dropped
new tower height = 140

Score = 0
drawing new block
225 600
1 0.588235 0.10098

prev tower height = 50
block dropped
right matched
new tower height = 50

Score = 2
drawing new block
225 600
1 0.588235 0.10098

Do you want to restart the game? [y/n]
y
```


Mini Tetris
Extensions
Window
Help
Search (Ctrl+Q)
Continue
C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
prev tower height = 95
block dropped
new tower height = 140

Score = 0
drawing new block
225 600
1 0.588235 0.10098

prev tower height = 50
block dropped
right matched
new tower height = 50

Score = 2
drawing new block
225 600
1 0.588235 0.10098

Do you want to restart the game? [y/n]
y

Restarting the game.
Score = 0
drawing new block
225 600
1 0.588235 0.10098
```

- **Case when the game ends:**



Mini Tetris

Extensions Window Help Search (Ctrl+Q)

C:\Users\Gautam Thareja\source\repos\MiniTetris\Debug\MiniTetris.exe

```
drawing new block
90 600
0.0117647 0.25098 0.678431

prev tower height = 95
block dropped
new tower height = 140

Score = 25
drawing new block
180 600
1 0.839216 0

prev tower height = 140
block dropped
right matched
new tower height = 140

Score = 27
drawing new block
225 600
1 0.588235 0.10098

prev tower height = 590
block dropped
new tower height = 635

Game Over!
Score = 27
```