Game Engines

Task 2

Jeffrey Camenzuli BA IM YR2

Target Device

The game will be available to play on mobile, on Android and IOS.

The resolution is 720x405 at a 16:9, which will suit all the latest smartphones.

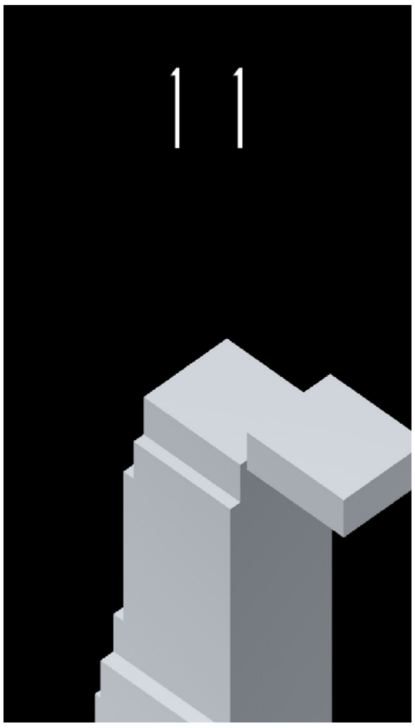
The game is played by using a touch screen, therefore, a mobile phone is perfect for the game.

Controls and Mechanics

The game stack is based around one control. Click on the screen when you think it is the appropriate time, and the tile will stop once you click and stack on top the tower.

If the tile isn't alligned with the previous tile, the parts of the tiles left hanging are cut, and the tile will shrink.

Once you successfully stack 3 tiles with no errors, if you keep stacking the tiles successfully the tile starts to increase in size once from the X-axis, and once from the Y-axis until the combo ends.

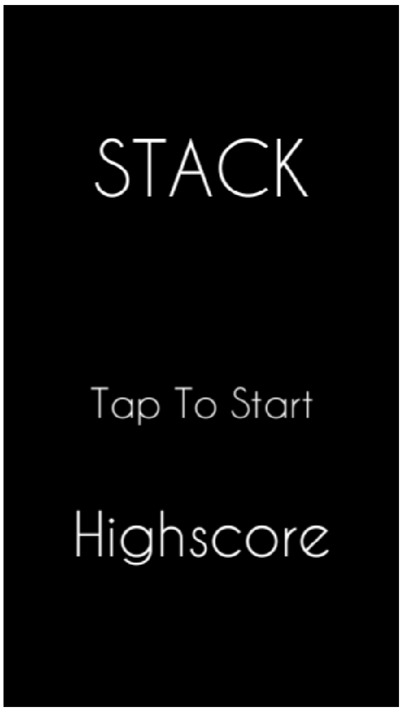


Game Screens

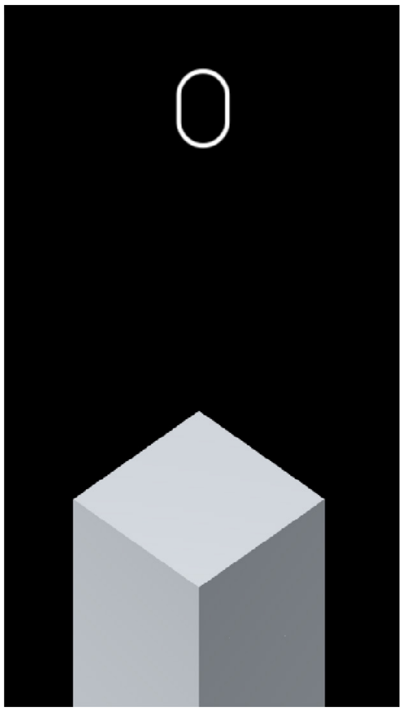
When you first enter the game the first screen you are presented with is the main menu. The menu only consists of 1 button (Tap to start), game title, and the high -score.

Once you tap the screen, the game starts. With every tile placed the score will go up by one point until you fail to stack the tile and lose the game.

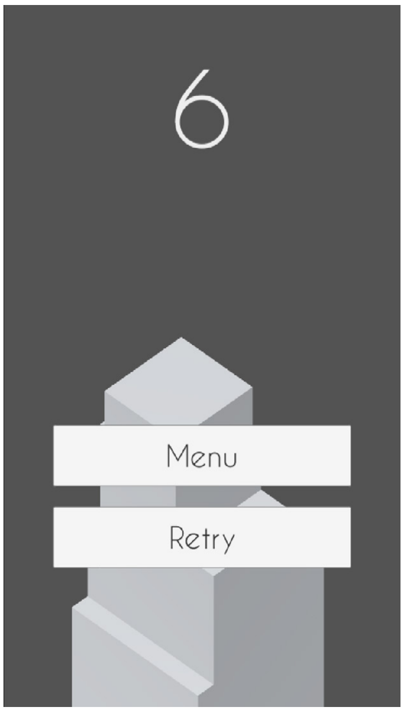
When you lose the game, you are presented with 2 buttons. One is the menu button to go back to the menu screen, and the retry button that once pressed you can play the game again.



Menu

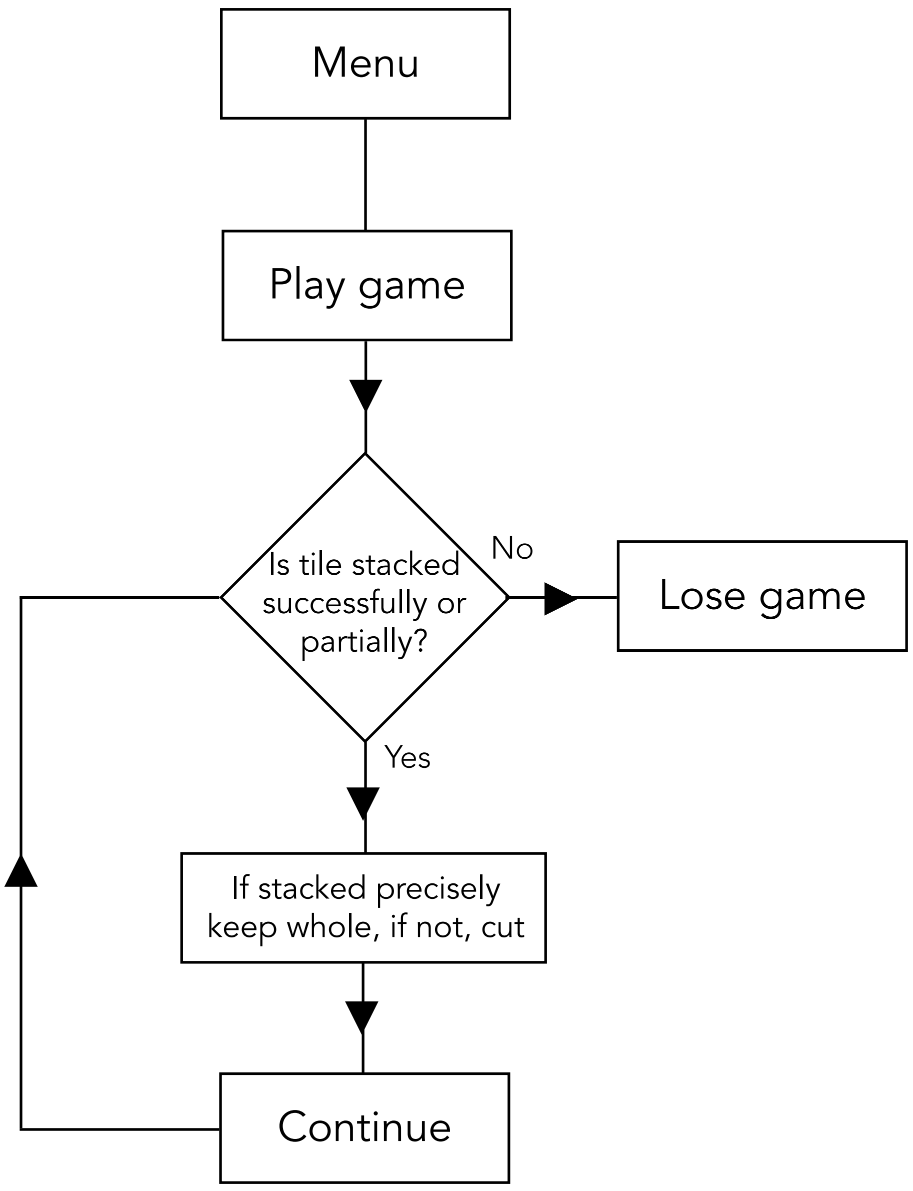


Play Game



End Game
Menu

Game Flowchart


Menu
Play game
Is tile stacked successfully or partially?
No
Lose game
Yes
If stacked precisely keep whole, if not, cut
Continue

Objective

The objective of the game is simple, keep on stacking the tiles until you misplace a tile and lose.

Visual Assets/UI Elements

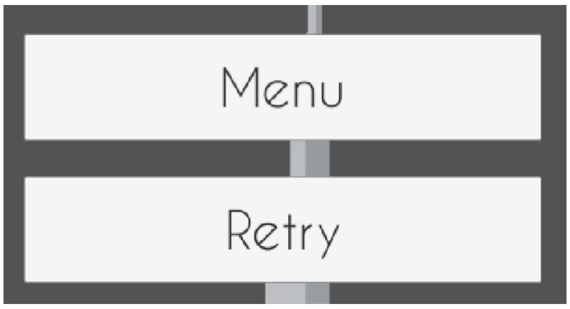
Typeface

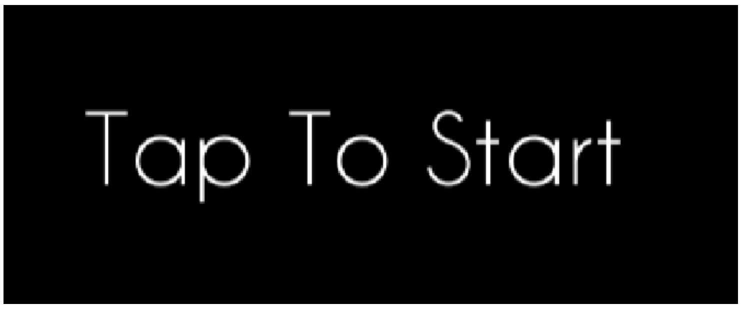
Caviar Dreams

Colour Scheme

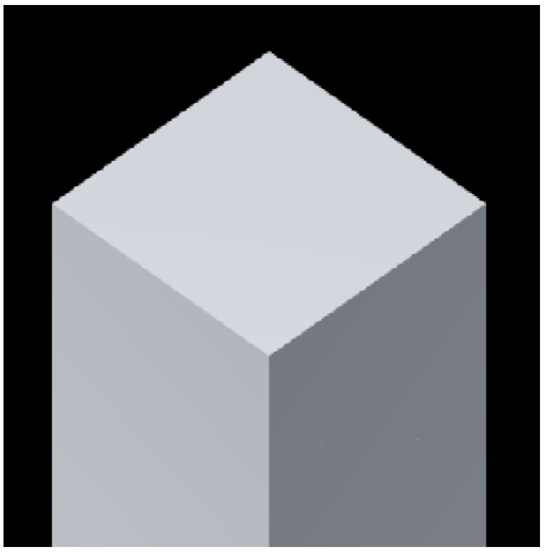
● #000000 ● #777C83 ● #B6BAC0 ● #D2D7DD

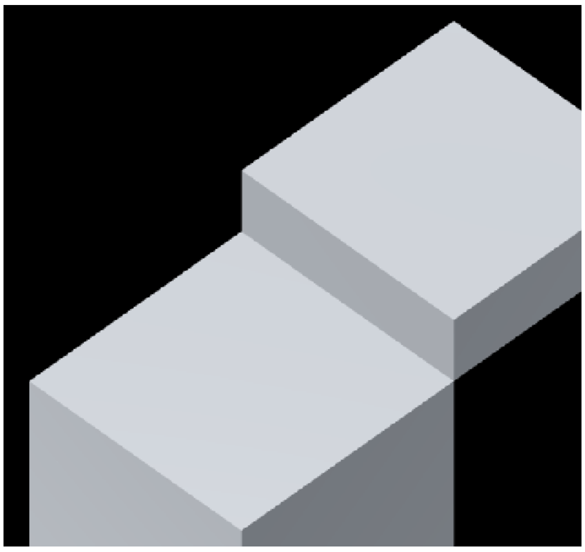
Buttons





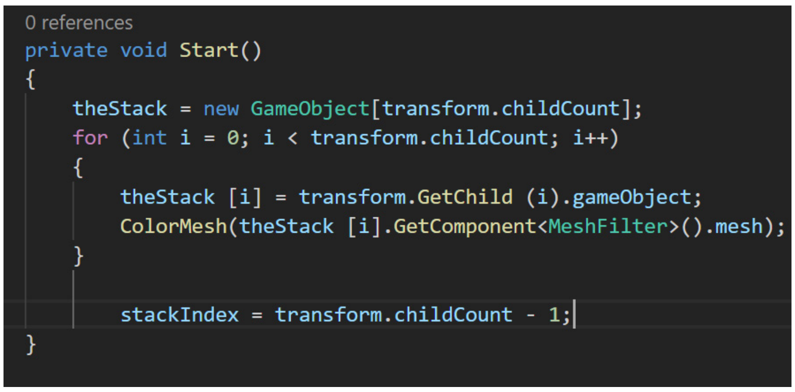
Tiles



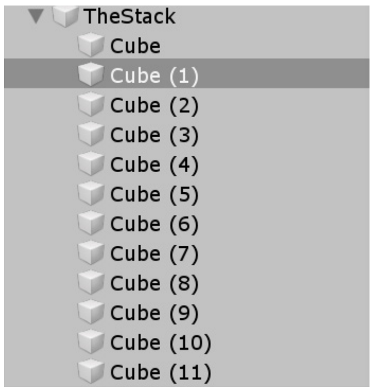


Timeline

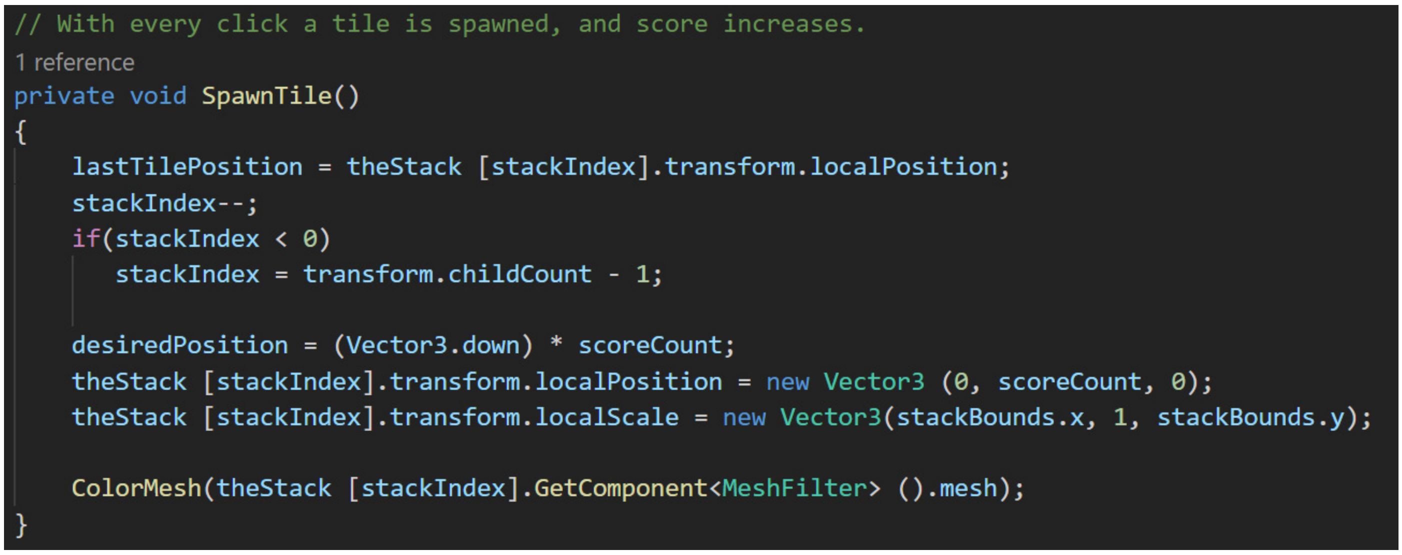
The first step to programming Stack was to set up the assets in Unity. 12 tiles were constructed separately and placed on top of each other. Once this was done, work on the tile movement started. The programming for the tile to come from the X-axis was done first, and since I wanted the tiles to also come from the Y-axis, the code was replicated and altered to have tiles coming from both sides.



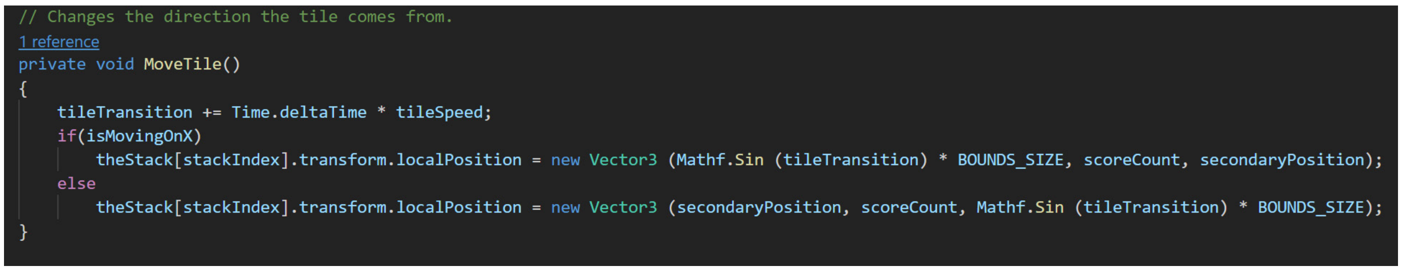
```
0 references
private void Start()
{
    theStack = new GameObject[transform.childCount];
    for (int i = 0; i < transform.childCount; i++)
    {
        theStack [i] = transform.GetChild (i).gameObject;
        ColorMesh(theStack [i].GetComponent<MeshFilter>().mesh);
    }

        stackIndex = transform.childCount - 1;
}
```



```
▼ TheStack
    Cube
    Cube (1)
    Cube (2)
    Cube (3)
    Cube (4)
    Cube (5)
    Cube (6)
    Cube (7)
    Cube (8)
    Cube (9)
    Cube (10)
    Cube (11)
```



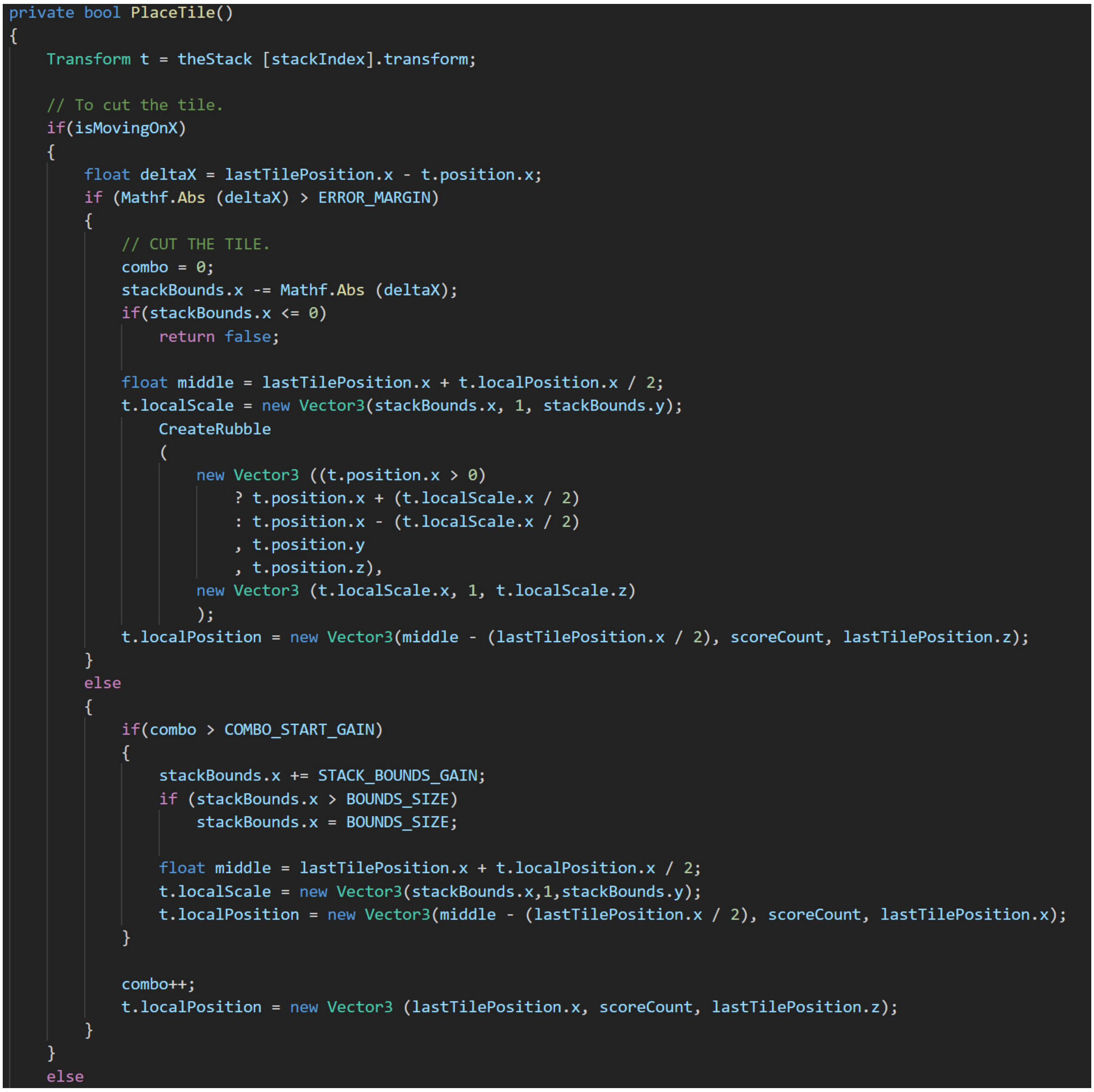
```
// With every click a tile is spawned, and score increases.
1 reference
private void SpawnTile()
{
    lastTilePosition = theStack [stackIndex].transform.localPosition;
    stackIndex--;
    if(stackIndex < 0)
        stackIndex = transform.childCount - 1;

    desiredPosition = (Vector3.down) * scoreCount;
    theStack [stackIndex].transform.localPosition = new Vector3 (0, scoreCount, 0);
    theStack [stackIndex].transform.localScale = new Vector3(stackBounds.x, 1, stackBounds.y);

    ColorMesh(theStack [stackIndex].GetComponent<MeshFilter> ().mesh);
}
```



```
// Changes the direction the tile comes from.
1 reference
private void MoveTile()
{
    tileTransition += Time.deltaTime * tileSpeed;
    if(isMovingOnX)
        theStack[stackIndex].transform.localPosition = new Vector3 (Mathf.Sin (tileTransition) * BOUNDS_SIZE, scoreCount, secondaryPosition);
    else
        theStack[stackIndex].transform.localPosition = new Vector3 (secondaryPosition, scoreCount, Mathf.Sin (tileTransition) * BOUNDS_SIZE);
}
```

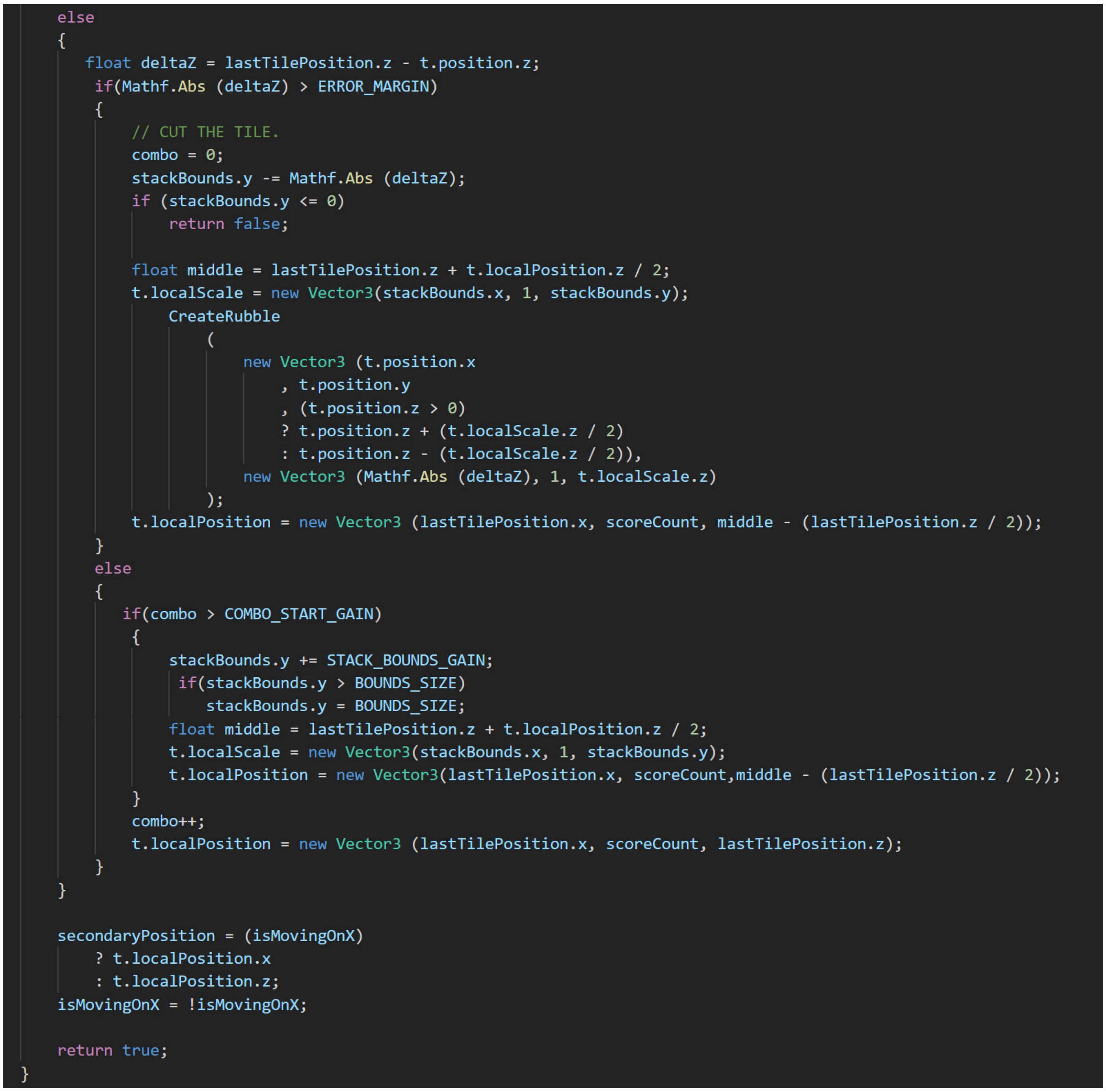
Once the movement of the tile was implemented, work on the tile being placed started. Again, I worked on the tile coming from the X-axis first, then replicated the same code and altered it for the Y-Axis. Once the mechanic of placing the tile was done, I started working on cutting the parts of the tile that aren't aligned to the previous tile, reducing the size of the tile and making it harder for the player. This part was the most intensive to program, compared to the other mechanics. With every 3 successfully stacked tiles, the tiles will start to increase in size until the combo ends.

For the X-axis



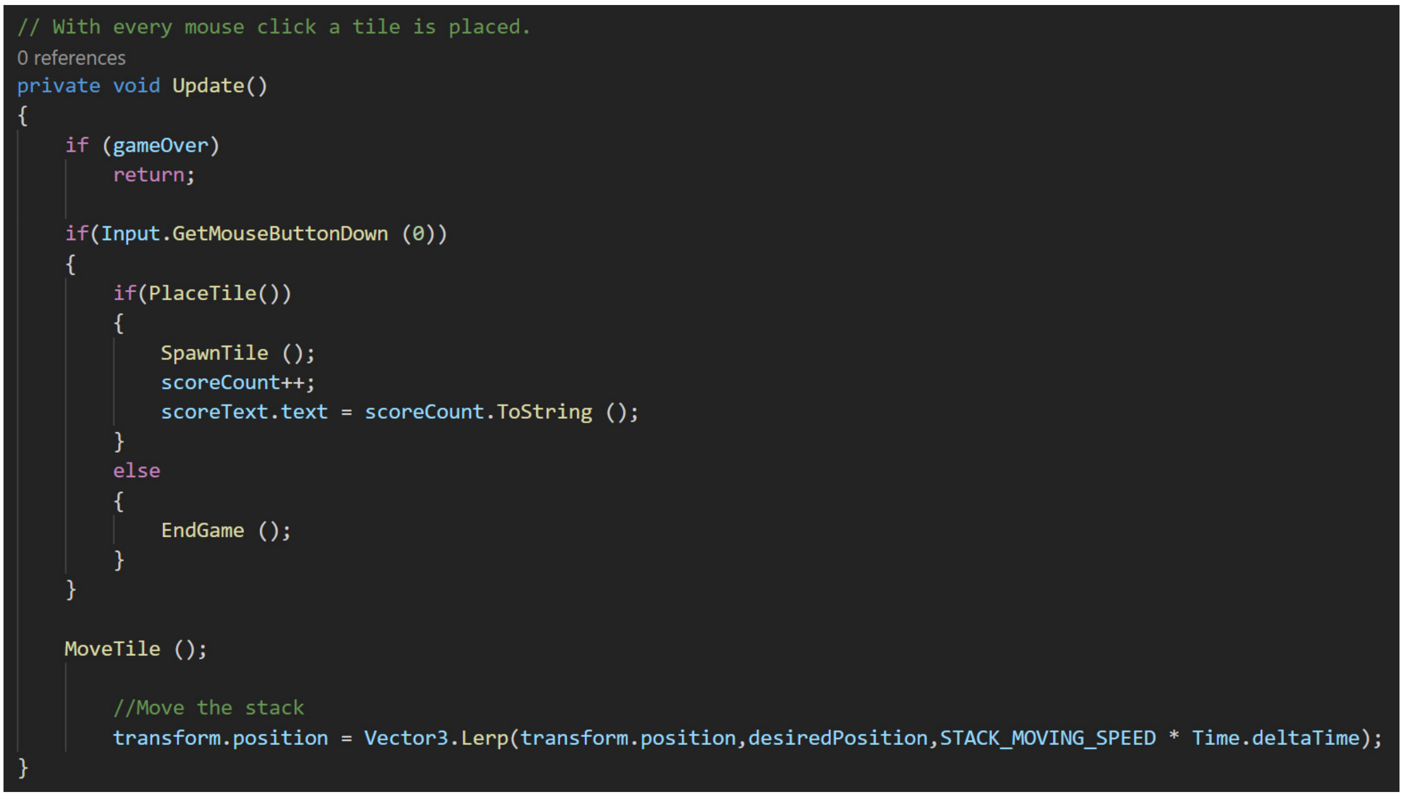
```
private bool PlaceTile()
{
    Transform t = theStack [stackIndex].transform;

    // To cut the tile.
    if(isMovingOnX)
    {
        float deltaX = lastTilePosition.x - t.position.x;
        if (Mathf.Abs (deltaX) > ERROR_MARGIN)
        {
            // CUT THE TILE.
            combo = 0;
            stackBounds.x -= Mathf.Abs (deltaX);
            if(stackBounds.x <= 0)
                return false;

            float middle = lastTilePosition.x + t.localPosition.x / 2;
            t.localScale = new Vector3(stackBounds.x, 1, stackBounds.y);
                CreateRubble
                (
                    new Vector3 ((t.position.x > 0)
                        ? t.position.x + (t.localScale.x / 2)
                        : t.position.x - (t.localScale.x / 2)
                        , t.position.y
                        , t.position.z),
                    new Vector3 (t.localScale.x, 1, t.localScale.z)
                    );
            t.localPosition = new Vector3(middle - (lastTilePosition.x / 2), scoreCount, lastTilePosition.z);
        }
        else
        {
            if(combo > COMBO_START_GAIN)
            {
                stackBounds.x += STACK_BOUNDS_GAIN;
                if (stackBounds.x > BOUNDS_SIZE)
                    stackBounds.x = BOUNDS_SIZE;

                float middle = lastTilePosition.x + t.localPosition.x / 2;
                t.localScale = new Vector3(stackBounds.x,1,stackBounds.y);
                t.localPosition = new Vector3(middle - (lastTilePosition.x / 2), scoreCount, lastTilePosition.x);
            }

            combo++;
            t.localPosition = new Vector3 (lastTilePosition.x, scoreCount, lastTilePosition.z);
        }
    }
    else
```

For the Y-axis.



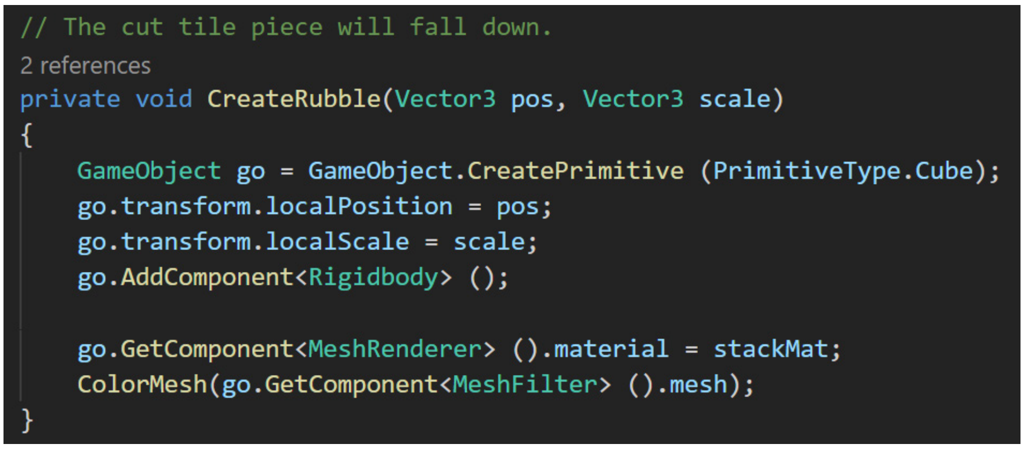
```
    else
    {
       float deltaZ = lastTilePosition.z - t.position.z;
        if(Mathf.Abs (deltaZ) > ERROR_MARGIN)
        {
            // CUT THE TILE.
            combo = 0;
            stackBounds.y -= Mathf.Abs (deltaZ);
            if (stackBounds.y <= 0)
                return false;

            float middle = lastTilePosition.z + t.localPosition.z / 2;
            t.localScale = new Vector3(stackBounds.x, 1, stackBounds.y);
                CreateRubble
                    (
                        new Vector3 (t.position.x
                            , t.position.y
                            , (t.position.z > 0)
                            ? t.position.z + (t.localScale.z / 2)
                            : t.position.z - (t.localScale.z / 2)),
                        new Vector3 (Mathf.Abs (deltaZ), 1, t.localScale.z)
                    );
            t.localPosition = new Vector3 (lastTilePosition.x, scoreCount, middle - (lastTilePosition.z / 2));
        }
        else
        {
           if(combo > COMBO_START_GAIN)
            {
                stackBounds.y += STACK_BOUNDS_GAIN;
                 if(stackBounds.y > BOUNDS_SIZE)
                    stackBounds.y = BOUNDS_SIZE;
                float middle = lastTilePosition.z + t.localPosition.z / 2;
                t.localScale = new Vector3(stackBounds.x, 1, stackBounds.y);
                t.localPosition = new Vector3(lastTilePosition.x, scoreCount,middle - (lastTilePosition.z / 2));
            }
            combo++;
            t.localPosition = new Vector3 (lastTilePosition.x, scoreCount, lastTilePosition.z);
        }
    }

    secondaryPosition = (isMovingOnX)
        ? t.localPosition.x
        : t.localPosition.z;
    isMovingOnX = !isMovingOnX;

    return true;
}
```

In this part of the script, the code to update the game in terms of what's happening was done, such as when to end the game, update the score, and move the stack with every tile placed.

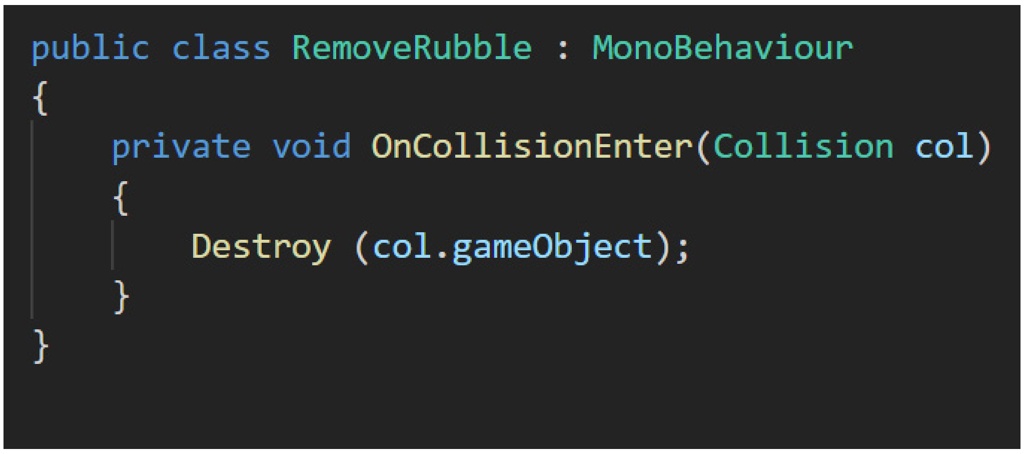


```
// With every mouse click a tile is placed.
0 references
private void Update()
{
    if (gameOver)
        return;

    if(Input.GetMouseButtonDown (0))
    {
        if(PlaceTile())
        {
            SpawnTile ();
            scoreCount++;
            scoreText.text = scoreCount.ToString ();
        }
        else
        {
            EndGame ();
        }
    }

    MoveTile ();

        //Move the stack
        transform.position = Vector3.Lerp(transform.position,desiredPosition,STACK_MOVING_SPEED * Time.deltaTime);
}
```

Once the game was in a playable condition, I added the rubble effect for the discarded part of the tile. The discarded part of the tile will detach and fall. To reduce the intensity of the game on the device, once the the tile rubble falls out of the screen, it will be destroyed and removed from the game making the game run smoother.



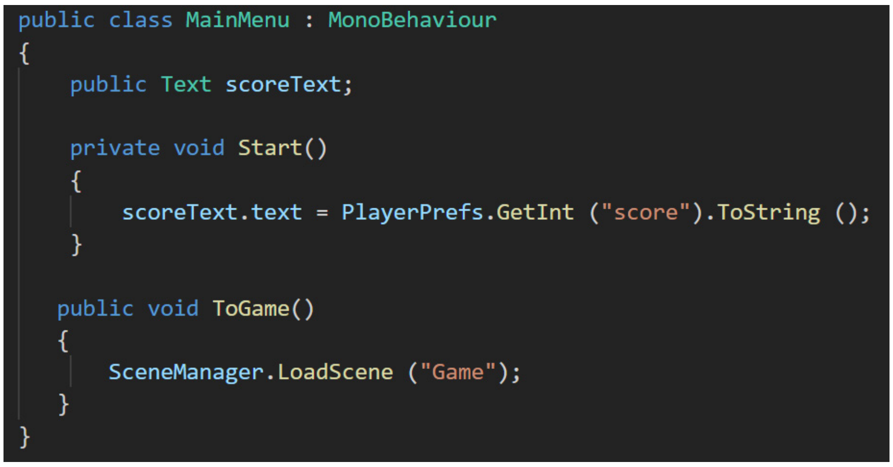
```
// The cut tile piece will fall down.
2 references
private void CreateRubble(Vector3 pos, Vector3 scale)
{
    GameObject go = GameObject.CreatePrimitive (PrimitiveType.Cube);
    go.transform.localPosition = pos;
    go.transform.localScale = scale;
    go.AddComponent<Rigidbody> ();

    go.GetComponent<MeshRenderer> ().material = stackMat;
    ColorMesh(go.GetComponent<MeshFilter> ().mesh);
}
```



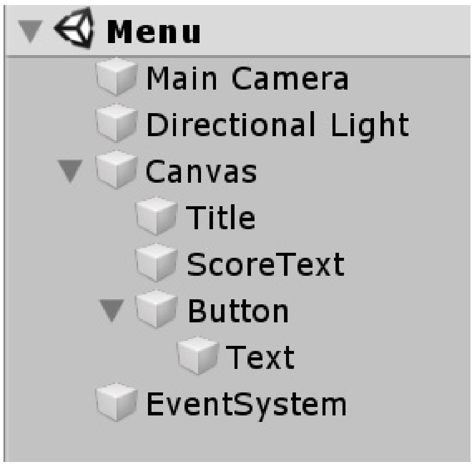
```
public class RemoveRubble : MonoBehaviour
{
    private void OnCollisionEnter(Collision col)
    {
        Destroy (col.gameObject);
    }
}
```

The final part of the game was creating the menu. There are two menus in the game which are, the main menu, and the game over menu.

Main menu script



```
public class MainMenu : MonoBehaviour
{
    public Text scoreText;

    private void Start()
    {
        scoreText.text = PlayerPrefs.GetInt ("score").ToString ();
    }

   public void ToGame()
   {
       SceneManager.LoadScene ("Game");
   }
}
```



```
▼ Menu
      Main Camera
      Directional Light
   ▼ Canvas
        Title
        ScoreText
     ▼ Button
          Text
      EventSystem
```