WIF3005 Software Maintenance and Evolution
Mohamad Sulaiman bin Mohd Hashim
17096959/3
Alternative Assessment
Question 1
Chosen repository: https://github.com/jakesgordon/javascript-tetris

---

1. Based on the https://github.com/jakesgordon/javascript-tetris source code in the repository chosen, two identified reusable components are:

    - Tetris block definition and generation
    - Canvas resize

1.1. Description of Components

1.1.1. Tetris block definition and generation

In a tetris game, a random block will be generated each instance or the player to place. Additionally, the next instance of block will also be generated but will only be displayed in the “Next Block” section for the player to strategize.

Tetris block definition and generation components are used to define each block shape and color. The code will generate 4 instances of each piece in an array and randomly returns one block when the function is called until the stack is empty. This function fulfills one of the basic gameplay requirements for tetris.

1.1.2. Canvas resize

This Tetris game is a browser based game, therefore when the user resizes the browser or the user using a browser at a different aspect ratio than the program expects, the resize function will adjust the game canvas accordingly. Without this function, there will be a cutoff and the player must scroll to see further as the full game court cannot be displayed within one window. This component fixes that problem.

The way this component solves the problem is by comparing the size of the current canvas with the user canvas (i.e. the size of the browser window) and then it modifies the nx, ny, dx, and dy accordingly. When the draw function refreshes the frame, it will use these new nx, ny, dx, and dy variables to ensure the game court is within the user browser window so the player can see the full court.

1.2. Code Snippet

1.2.1. Tetris block definition and generation

```
//-------------------------------------------------------------------------
// tetris pieces
//
// blocks: each element represents a rotation of the piece (0, 90, 180, 270)
//         each element is a 16 bit integer where the 16 bits represent
//         a 4x4 set of blocks, e.g. j.blocks[0] = 0x44C0
//
//             0100 = 0x4 << 3 = 0x4000
//             0100 = 0x4 << 2 = 0x0400
//             1100 = 0xC << 1 = 0x00C0
//             0000 = 0x0 << 0 = 0x0000
//                               ------
//                               0x44C0
//
//-------------------------------------------------------------------------

var i = { size: 4, blocks: [0x0F00, 0x2222, 0x00F0, 0x4444], color: 'cyan'   };
var j = { size: 3, blocks: [0x44C0, 0x8E00, 0x6440, 0x0E20], color: 'blue'   };
var l = { size: 3, blocks: [0x4460, 0x0E80, 0xC440, 0x2E00], color: 'orange' };
var o = { size: 2, blocks: [0xCC00, 0xCC00, 0xCC00, 0xCC00], color: 'yellow' };
var s = { size: 3, blocks: [0x06C0, 0x8C40, 0x6C00, 0x4620], color: 'green'  };
var t = { size: 3, blocks: [0x0E40, 0x4C40, 0x4E00, 0x4640], color: 'purple' };
var z = { size: 3, blocks: [0x0C60, 0x4C80, 0xC600, 0x2640], color: 'red'    };
//---------------------------------------
// start with 4 instances of each piece and
// pick randomly until the 'bag is empty'
//---------------------------------------
var pieces = [];
function randomPiece() {
  if (pieces.length == 0)
    pieces = [i,i,i,i,j,j,j,j,l,l,l,l,o,o,o,o,s,s,s,s,t,t,t,t,z,z,z,z];
  var type = pieces.splice(random(0, pieces.length-1), 1)[0];
  return { type: type, dir: DIR.UP, x: Math.round(random(0, nx - type.size)), y: 0 };
}
```

1.2.2. Canvas resize

```
var canvas  = get('canvas'),
    ctx     = canvas.getContext('2d'),
    ucanvas = get('upcoming'),
    uctx    = ucanvas.getContext('2d'),
    nx      = 10, // width of tetris court (in blocks)
    ny      = 20, // height of tetris court (in blocks)
    nu      = 5;  // width/height of upcoming preview (in blocks)

function resize(event) {
  canvas.width   = canvas.clientWidth;  // set canvas logical size equal to its
                                        //physical size
  canvas.height  = canvas.clientHeight; // (ditto)
  ucanvas.width  = ucanvas.clientWidth;
  ucanvas.height = ucanvas.clientHeight;
  dx = canvas.width  / nx; // pixel size of a single tetris block
  dy = canvas.height / ny; // (ditto)
  invalidate();
  invalidateNext();
}

var invalid = {};

function invalidate()         { invalid.court  = true; }
function invalidateNext()     { invalid.next   = true; }
function drawCourt() {
  if (invalid.court) {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    if (playing)
      drawPiece(ctx, current.type, current.x, current.y, current.dir);
    var x, y, block;
    for(y = 0 ; y < ny ; y++) {
      for (x = 0 ; x < nx ; x++) {
        if (block = getBlock(x,y))
          drawBlock(ctx, x, y, block.color);
      }
    }
    ctx.strokeRect(0, 0, nx*dx - 1, ny*dy - 1); // court boundary
    invalid.court = false;
  }
}
```

1.3. Sample Implementation in Another Context

Below are the implementation of the isolated components proven as deployable on its own.

1.3.1. Tetris block definition and generation

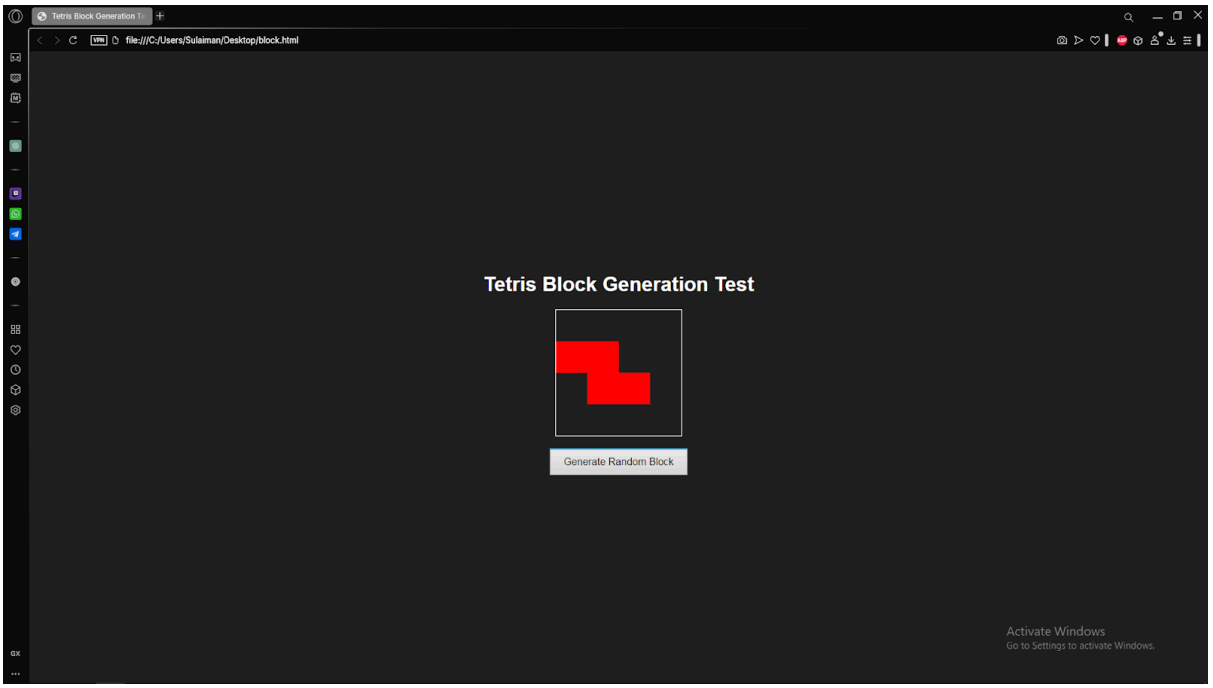

Image 1.3.1.1: Sample implementation of block definition and generation.

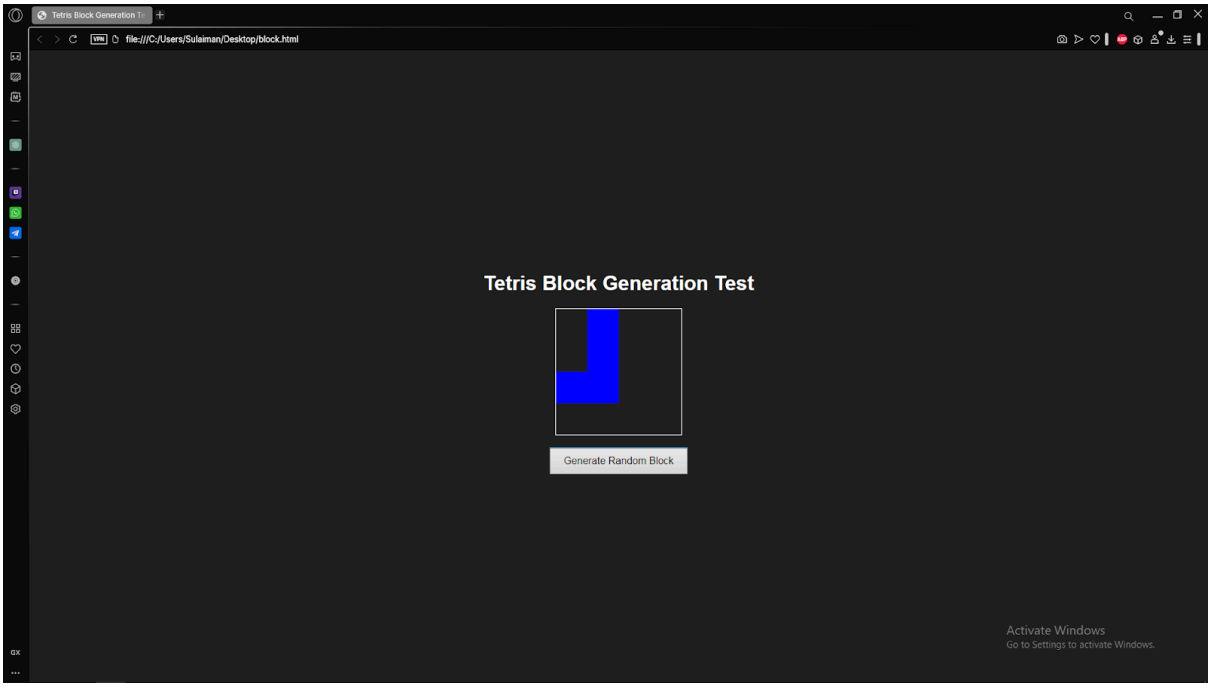

Image 1.3.1.2: Generated a different block randomly.

1.3.2. Canvas resize

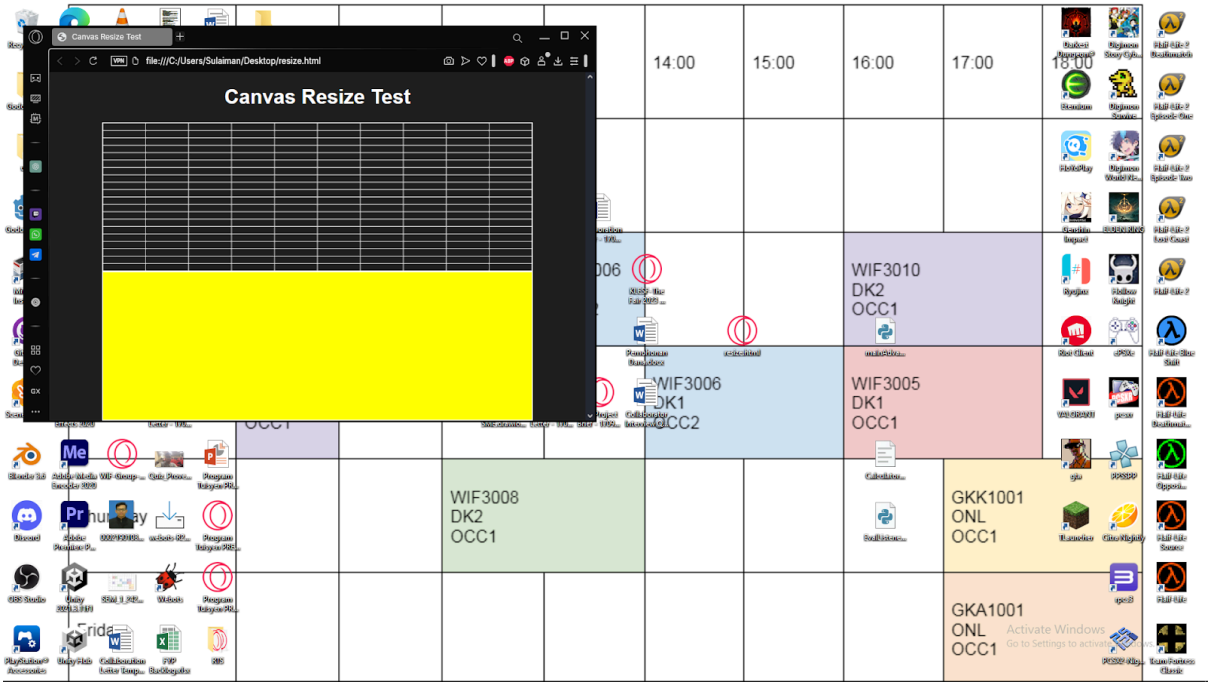

Image 1.3.2.1: Sample implementation of canvas resize.

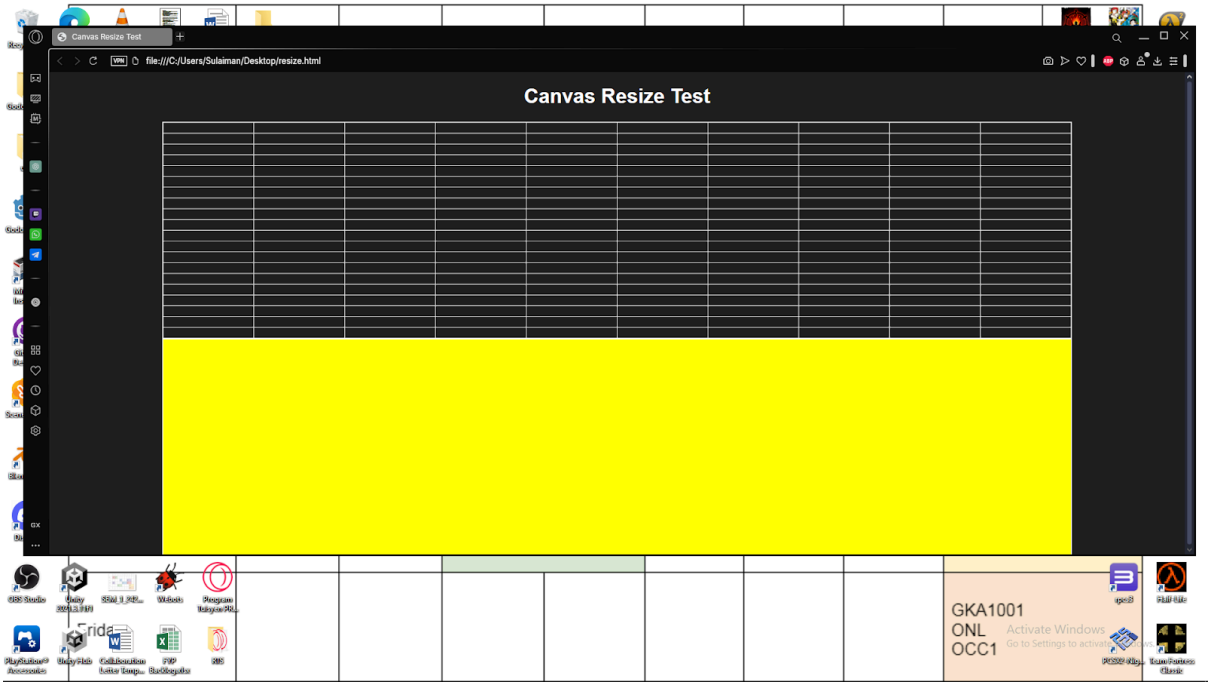

Image 1.3.2.2: Resized to a different window size. Each rows and columns change size accordingly