HTML5 Game

By Sarah Hosam & Hadeer Shehab

Game Logic:

The game consists of a jet which fires bullets, this being the player we control, and the other aspect is the enemies who also fire bullets that should be avoided by the player in order to stay alive.

The player jet is controlled by the arrows on the keyboard, and bullets are fired using the space button. Whenever a player bullet collides with an enemy, the enemy dies, and the level goes on until all the enemies are dead, we then move to level two. However, if the player is touched by one of the enemy bullets, its game over and the player has an option to restart the game.

Techniques used throughout the game:

First of all, we used 3 different canvases, to avoid lagging where we won't have to refresh and draw everything each frame. One canvas would be for the background, moving back and forth each frame, another for the jet and the last one for the bullets.

An object pool is used, to reuse old objects so we don't have to keep forming and deleting objects all the time. This pool is made for the bullets.

```
function Pool(maxSize) {
    var size = maxSize;
    var pool = [];

    this.getPool = function() {
        var obj = [];
        for (var i = 0; i < size; i++) {
            if (pool[i].alive) {
                obj.push(pool[i]);
            }
        }
        return obj;
    }
    this.init = function(object) {
        if (object == "bullet") {
            for (var i = 0; i < size; i++) {
                // Initalize the object
                var bullet = new Bullet("bullet");
                bullet.init(0,0, imageRepository.bullet.width,
                            imageRepository.bullet.height);
                bullet.collidableWith = "enemy";
                bullet.type = "bullet";
                pool[i] = bullet;
            }
        }
        else if (object == "enemy") {
            for (var i = 0; i < size; i++) {
                var enemy = new Enemy();
                enemy.init(0,0, imageRepository.enemy.width,
```

A technique called dirty rectangle is used with the bullets, this ensures that when bullets are gone out of the screen, they are ready to be re-used, this techniques makes it a lot faster performance-wise.

Keyboard input has been handled by a technique developed earlier, and we hold no copyright for that, this works by creating a JSON object that holds the key codes for each key and how it maps to our game.

```
KEY_CODES = {
  32: 'space',
  37: 'left',
  38: 'up',
  39: 'right',
  40: 'down',
}
KEY_STATUS = {};
for (code in KEY_CODES) {
  KEY_STATUS[KEY_CODES[code]] = false;
}
document.onkeydown = function(e) {

  var keyCode = (e.keyCode) ? e.keyCode : e.charCode;
  if (KEY_CODES[keyCode]) {
    e.preventDefault();
    KEY_STATUS[KEY_CODES[keyCode]] = true;
  }
}
document.onkeyup = function(e) {
  var keyCode = (e.keyCode) ? e.keyCode : e.charCode;
  if (KEY_CODES[keyCode]) {
    e.preventDefault();
    KEY_STATUS[KEY_CODES[keyCode]] = false;
  }
}
```

The enemies in this game move all together, this is done by giving each enemy 3 boundaries, left, right and bottom. When an enemy has been fired, it is removed from the enemy pool.

Handling collisions (between player and enemy bullets, or player bullets and enemy) has been detected by a technique called spatial portioning. Having to compare the positions of all the objects on the screen with each other would take a huge amount of time so instead, we divide up the canvas into different sections, and only compare everything that exists in that one section. This speeds up the process a whole lot, using a data structure called a quad tree.

```
function QuadTree(boundBox, lvl) {
  var maxObjects = 10;
  this.bounds = boundBox || {
    x: 0,
    y: 0,
    width: 0,
    height: 0
  };
  var objects = [];
  this.nodes = [];
  var level = lvl || 0;
  var maxLevels = 5;
  this.clear = function() {
    objects = [];

    for (var i = 0; i < this.nodes.length; i++) {
      this.nodes[i].clear();
    }

    this.nodes = [];
  };
  this.getAllObjects = function(returnedObjects) {
    for (var i = 0; i < this.nodes.length; i++) {
      this.nodes[i].getAllObjects(returnedObjects);
    }

    for (var i = 0, len = objects.length; i < len; i++) {
      returnedObjects.push(objects[i]);
    }
```

Now a score is kept, where every enemy killed adds 100 points to the player's score

```
      return false;
    }
    else {
      game.playerScore += 100;
      return true;
    }
  };
```

A game over screen appears when one of the enemy bullets collides with the player

```
// Game Over
this.gameOver = function() {

  document.getElementById('game-over').style.display = "block";
};
}
```

And if all enemies are dead, a new level is initiated

```
function nextlevel()
{
  document.getElementById('level2').style.display = "none";
  game.spawnWave();
}
```

Background sound has been added

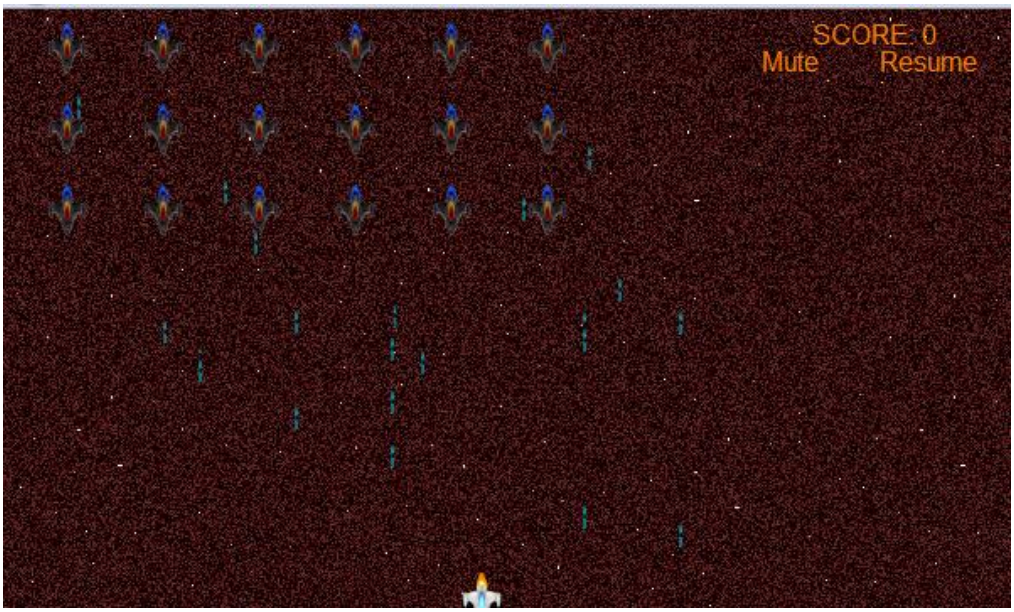
```
window.onload = PlayHTML5Audio;
var backgroundMusic = new Audio("Knight raider.wav");
function PlayHTML5Audio()
{
  backgroundMusic.play();
  backgroundMusic.loop = true;
}
```

Along with mute and pause buttons

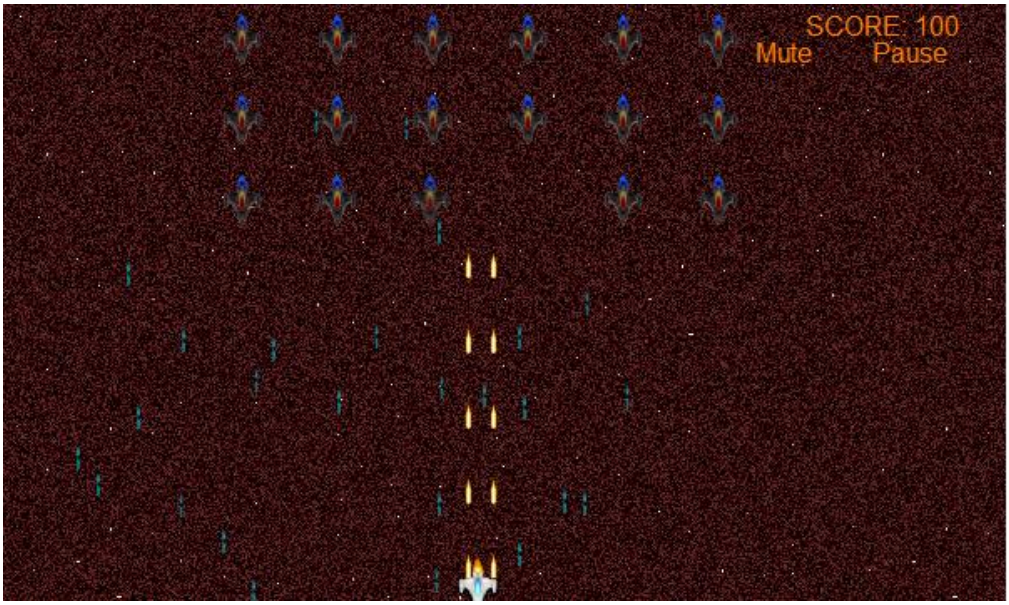
```
var pause = false;
var mute = false;
function triggerMute()
  {
    if (mute == true)
    {
      mute = false;
  backgroundMusic.muted = false;
  //"<p><span onclick="triggerMute()">Mute</span></p>"
  //document.getElementById('muteButton').firstChild.firstChild.firstChild.textContent
      //document.getElementById('muteButton').innerHTML = "Mute";
  document.getElementById('muteButton').firstChild.firstChild.firstChild.textContent = "Mute";
    }
    else
    {
      mute = true;
  backgroundMusic.muted = true;
      //document.getElementById('muteButton').innerHTML = "Un-Mute";
  document.getElementById('muteButton').firstChild.firstChild.firstChild.textContent = "Un-Mute";
    }
  }

  function triggerPause()
  {
    if (pause == true)
```

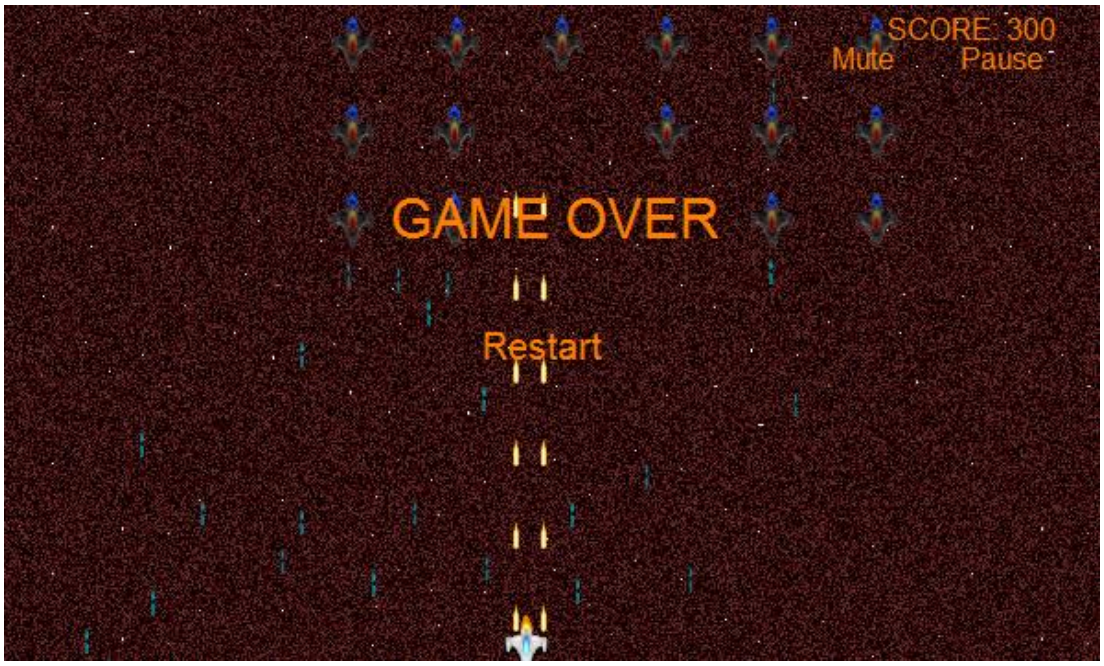
The game:



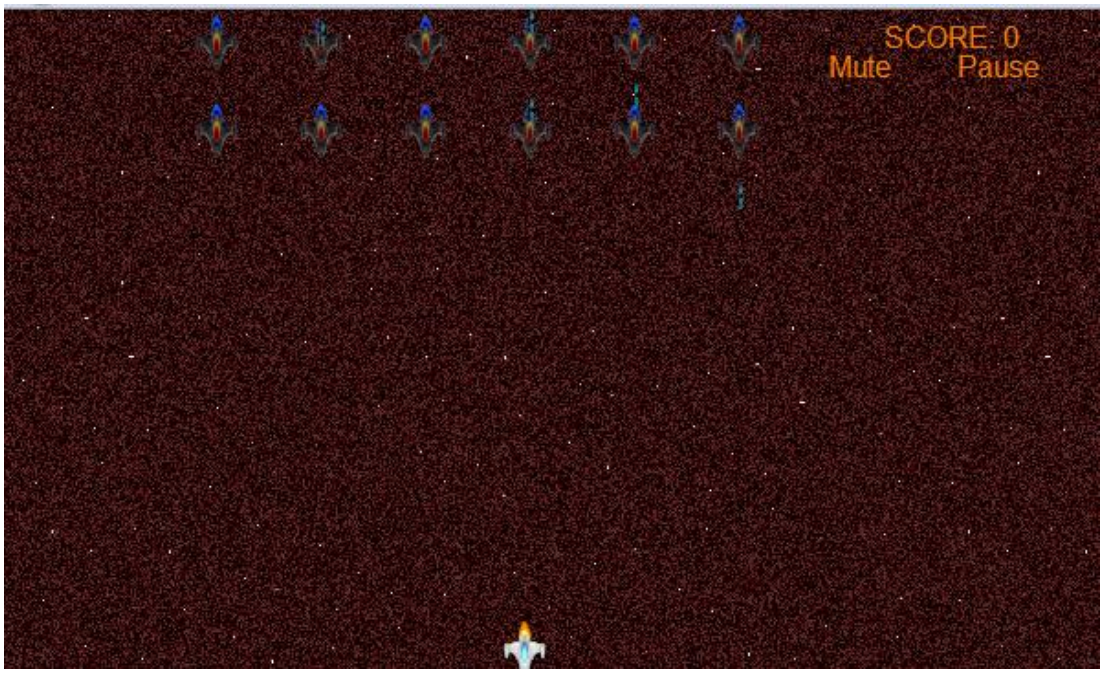
Firing bullets:



Game Over, hit by a blue bullet:

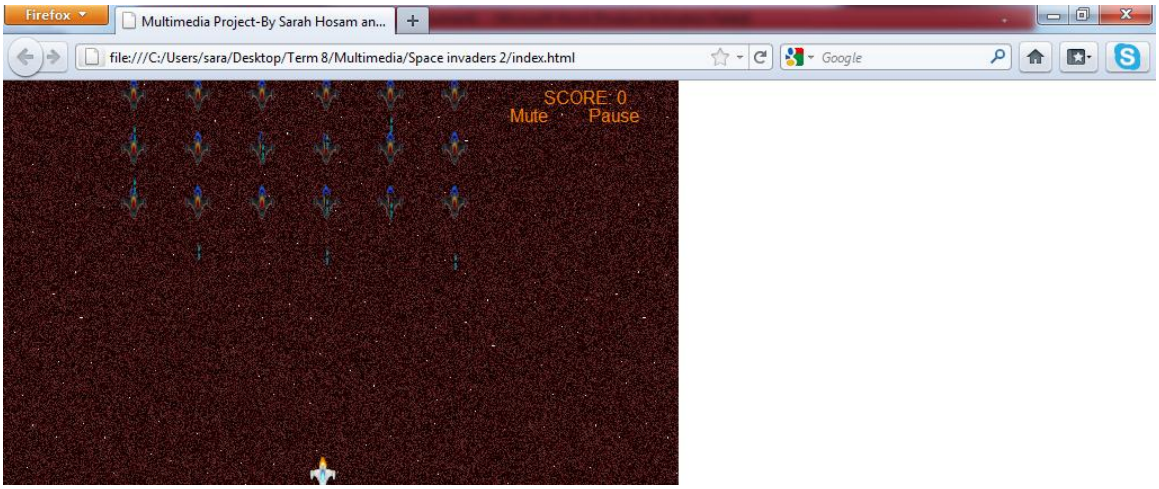


Restarts and resets the score:



When all are dead, level 2 :

Works on firefox:



Works on chrome: