**PROGRAMME GROUP**: P100 & P101

**UNIT TITLE: Game Development**

**UNIT CODE**: COM: 433E

Lesson 2:

Coverage Area

JavaScript Game Development

- Is JavaScript Good for Game Development?
- Popular Games That Use JavaScript
- What are the Best JavaScript Game Engines?
- How to Code a Game in JavaScript? With an Example

## Is JavaScript Good for Game Development

Now that you know JavaScript can be used to make games, *it brings the question of is it good for this task.*

**Yes!**

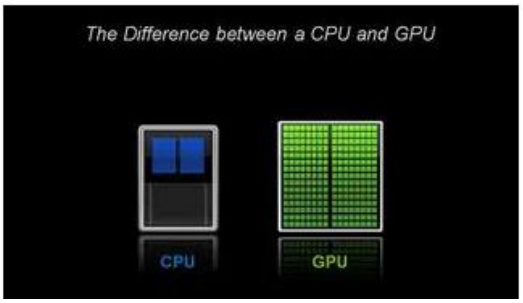
JavaScript is a great language for game development, depending on the type of game you want to create. **JavaScript is best for web-based and mobile games.**

Using platforms and tools can help create both 2D and 3D games that can run directly in your browser.

**On the contrary,**

if you're looking to create the next big **AAA** game,

- like *Call of Duty or FIFA*, using JavaScript, *you may find it challenging*. Even though the language is very versatile.
- **JavaScript is much slower than languages such as C++** and **consumes much more memory**.
- Advanced games require heavy **GPU calculations** and it's a substantial amount of weight to carry that **JavaScript just isn't cut out for.**




**GPUs** have ignited a worldwide AI boom. They've become a key part of *modern supercomputing*. They've been woven into sprawling new hyperscale data centers. **Still prized by gamers,** *they've become accelerators speeding up all sorts of tasks from encryption to networking to AI.*

- While **GPUs** (graphics processing unit) are now about a lot more than the PCs in which they first appeared, they remain anchored in a much older idea called parallel computing. And that's what makes GPUs so powerful.

- **CPUs,** to be sure, remain essential. Fast and versatile, CPUs race through a series of tasks requiring lots of interactivity. Calling up information from a hard drive in response to user's keystrokes, for example.

**What is parallel computing with example?**

- Image result for parallel computing is Shared memory parallel computers *use multiple processors to access the same memory resources.*
  - **Examples** of shared memory parallel architecture are *modern laptops, desktops, and smartphones.* Distributed memory *parallel computers use multiple processors, each with their own memory, connected over a network.*

By contrast, GPUs break complex problems into thousands or millions of separate tasks and work them out at once.

That makes them ideal for graphics, where textures, lighting and the rendering of shapes have to be done at once to keep images flying across the screen.

CPU vs GPU

| CPU | GPU |
|---|---|
| Central Processing Unit | Graphics Processing Unit |
| Several cores | Many cores |
| Low latency | High throughput |
| Good for serial processing | Good for parallel processing |
| Can do a handful of operations at once | Can do thousands of operations at once |

**GPUs: Key to AI, Computer Vision, Supercomputing and More**

- **GPUs** *perform much more work for every unit of energy than CPUs*. That makes them key to supercomputers that would otherwise push past the limits of today's electrical grids.

## Popular games in JavaScript

- **Tower Building** is a great way to get started with JavaScript games. The game allows players to stack blocks to create a very tall tower. This is a fantastic game to look at because it not only includes a QR code for you to play on your phone, but you can also browse, fork, and clone the GitHub repository to see how the game was created.

- **Polycraft** is a 3D game that is playable in your browser. Polycraft is full of adventure, exploration, base-building, gathering, crafting, and even fighting. It's an excellent example of how you can move past 2D games with Javascript.

- **Words With Friends 2** is a mobile app game that uses React Native, a framework that utilizes JavaScript to create mobile applications. Zynga chose to use React Native for its ability to create a game that can be played on multiple platforms using JavaScript with one code-base.

## What are the Best JavaScript Game Engines?

When developing games with JavaScript it's very common to use a game engine or rendering library

**Game engines** *are software that allow you to create extra components for games such as sound, animations, graphics, and physics*

### PixiJS

PixiJS is an open-sourced engine that prides itself on speed and beautiful API. The 2D renderer also has cross-platform support so you can make your game for multiple applications. Being open-source also allows a highly supportive community to take part in providing consistent improvements to the engine.

### BabylonJS

BabylonJS is a rendering library that has very powerful tools that allow you to create anything from simple animations to 3D games. Like PixiJS, BabylonJS is also open-sourced and has a large community of developers to help it grow.

### Phaser

Phaser offers support for desktop and mobile HTML5 games. Its focus is on 2D game development that can be compiled to multiple platforms. A benefit of using Phaser is the ability to use additional plugins as needed. This allows you to keep your tools small in size so you don't have too many unnecessary components.

### Three.js

Another popular library for rendering 3D graphics in a web browser is Three.js. It's fairly easy to learn and is highly popular, which means there are an endless amount of examples available. Its default renderer is WebGL, but it also provides support for SVG, Canvas 2D, and CSS3D renderers.

## HTML Canvas Game

### Learn how to make games, using nothing but HTML and JavaScript:

### To make a game, start by creating a gaming area, and make it ready for drawing:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style>
canvas {                    //canvas for rendering 2D shapes and bitmaps images.
   border:1px solid #d3d3d3;
   background-color: #f1f1f1;
}
</style>
</head>
<body onload="startGame()">
<script>

var myGamePiece; // game components (rules in games)
var myObstacles = [];   // game components in every frame
var myScore;

function startGame() {    // creating a  function start
   myGamePiece = new component (30, 30, "red", 10, 120); // adding constructors
   myGamePiece.gravity = 0.05;
   myScore = new component ("30px", "Consolas", "black", 280, 40, "text");
   myGameArea.start();
}
```

```
var myGameArea = {      // game parameters
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas, document.body.childNodes[0]);
        this.frameNo = 0;
        this.interval = setInterval(updateGameArea, 20);
        },
    clear: function () { // method removes all elements from the list. no parameters or return value.
        this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
    }
}

function component (width, height, color, x, y, type) {
    this.type = type;   // this key word referencing the current object/code
    this.score = 0;
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.gravity = 0;
    this.gravitySpeed = 0;
```

```
this.update = function() {
    ctx = myGameArea.context; // ctx means contex in java script used in canvas
    if (this.type == "text") {
        ctx.font = this.width + " " + this.height;
        ctx.fillStyle = color;
        ctx.fillText(this.text, this.x, this.y);
    } else {
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
}
this.newPos = function() {
    this.gravitySpeed += this.gravity;
    this.x += this.speedX;
    this.y += this.speedY + this.gravitySpeed;
    this.hitBottom();
}
this.hitBottom = function() {
    var rockbottom = myGameArea.canvas.height - this.height;
    if (this.y > rockbottom) {
        this.y = rockbottom;
        this.gravitySpeed = 0;
    }
}
this.crashWith = function(otherobj) {
    var myleft = this.x;
    var myright = this.x + (this.width);
```

```
        var mytop = this.y;
        var mybottom = this.y + (this.height);
        var otherleft = otherobj.x;
        var otherright = otherobj.x + (otherobj.width);
        var othertop = otherobj.y;
        var otherbottom = otherobj.y + (otherobj.height);
        var crash = true;
        if ((mybottom < othertop) || (mytop > otherbottom) || (myright < otherleft) ||
(myleft > otherright)) {
            crash = false;
        }
        return crash;
    }
}

function updateGameArea() {
    var x, height, gap, minHeight, maxHeight, minGap, maxGap;
    for (i = 0; i < myObstacles.length; i += 1) {
        if (myGamePiece.crashWith(myObstacles[i])) {
            return;
        }
    }
    myGameArea.clear();
    myGameArea.frameNo += 1;
    if (myGameArea.frameNo == 1 || everyinterval(150)) {
        x = myGameArea.canvas.width;
        minHeight = 20;
```

```
        maxHeight = 200;
        height = Math.floor(Math.random()*(maxHeight-minHeight+1)+minHeight);
        minGap = 50;
        maxGap = 200;
        gap = Math.floor(Math.random()*(maxGap-minGap+1)+minGap);
        myObstacles.push(new component(10, height, "green", x, 0));
        myObstacles.push(new component(10, x - height - gap, "green", x, height + gap));
    }
    for (i = 0; i < myObstacles.length; i += 1) {
        myObstacles[i].x += -1;
        myObstacles[i].update();
    }
    myScore.text="SCORE: " + myGameArea.frameNo;
    myScore.update();
    myGamePiece.newPos();
    myGamePiece.update();
}

function everyinterval(n) {
    if ((myGameArea.frameNo / n) % 1 == 0) {return true;}
    return false;
}

function accelerate(n) {
    myGamePiece.gravity = n;
}
</script>
```

<br>

<button onmousedown="accelerate(-0.2)"
onmouseup="accelerate(0.05)">ACCELERATE</button>

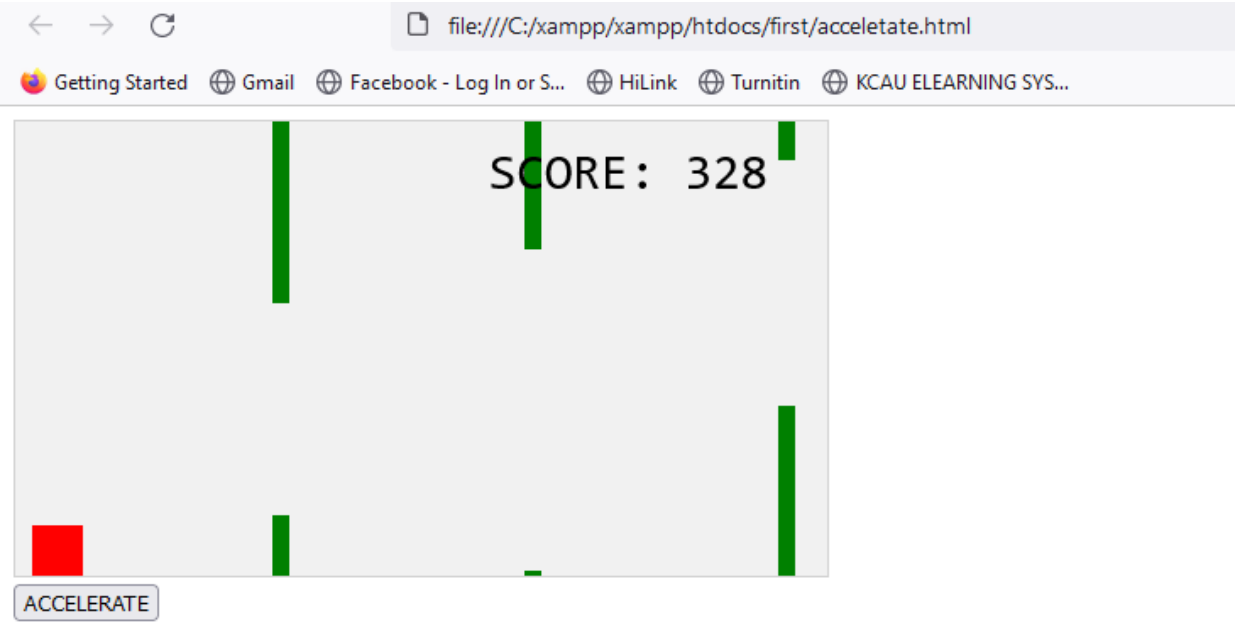
Use the ACCELERATE button to stay in the air

How long can you stay alive?

</body>

</html>

## Output……..




## Code Analysis

- The viewport is **the user's visible area of a web page**.
- <canvas> is an HTML element which can be used to **draw graphics via scripting** (usually JavaScript). This can, for instance, be used to draw graphs, combine photos, or create simple (and not so simple) animations