AttackAlienInvaders Game App-> A cool 2D alien planet exploration steampunk-themed game application with techniques of vanilla JavaScript, HTML5, CSS3 and HTML Canvas. From sprite animation to parallax backgrounds, the game App is completely built from scratch, with no frameworks or libraries, using HTML, CSS and plain vanilla JavaScript. The game contains richful premium art assets for characters, environments and props.

Functionalities/Demo:

Game Premiere Storyline:

[Central Computer]: These alien creatures have very similar physiology to the earth seahorses. Their bodies can easily cut through the thick atmosphere. They can move very fast.

[Explorer]: Seahorses hives are being attacked by something. This one has been damaged. I wonder if you can hack into this creature's central computer to control it for a while.

[Central Computer]: It is surprisingly easy to override its circuits. Seems like these machines are not used by our technology. I'm getting a lot of new data.

[Central Computer]: The atmosphere on this planet is thick enough to allow heavy silicon-based lifeforms to float but smoke is blocking most of the sunlight. Many creatures developed artificial lights and glowing appendages to see through the heavy clouds. The Seahorse sentinel has a basic attack that's powerful against weak enemies, but if it absorbs energy from one of the overcharged creatures, it gets additional firepower for a short period of time and it instantly replenishes its ammo. Ammo also automatically recharges over time. It seems we just need to help it to get through these aggressive swarms in time so it can join its hive.

- There is a stream of data about multiple different species.
- The alien creatures in the hood of their ecosystem have special movements, special abilities, and interact with the environments on other planets.

Developing Languages, Tools, and Techniques Needed:

JavaScript https://www.javascript.com/

Vanilla JS http://vanilla-js.com/

Vscode 1.72 https://code.visualstudio.com/updates/v1_72

HTML5 https://en.wikipedia.org/wiki/HTML5

CSS3 https://en.wikipedia.org/wiki/CSS

Live Server Vscode Extension v5.7.9 https://www.vsixhub.com/vsix/1950/

Google Fonts https://fonts.google.com/specimen/Bangers

Prerequisites & Setups:

HTML & CSS setup:

use `mkdir` to create the local directory for the project.

Set up the basic background for canvas 1 in `index.html`:

```
...
    <title>JavaScript Game</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <canvas id="canvas1"></canvas>
    <script src = "script.js"> </script>
...
```

Set up the fundamental styles for canvas 1 in `style.css`:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
 }
#canvas1 {
    border: 5px solid black;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background: #4d79bc;
```

Basic JavaScript setup:

Create a Load Event in `script.js` to fire when the whole page has been loaded, including all dependent resources as stylesheets and images for graphics using:

```
 window.addEventListener('load', function () {
    //canvas setup
    const canvas = document.getElementById('canvas1');
```

Call the built-in object Drawing Context which contains all methods and properties to draw and animate colors, shapes and other graphics on HTML canvas:

```
  const ctx = canvas.getContext('2d');
```

Now the canvas 1 resizing-responsive:

resizing-responsive canvas1.PNG

Synchronous Developing Notes:

Use Object Oriented JS to wrap variables and functions into objects. So use Encapsulation method to access the data that can be restricted from outside the bundle.

Creating Player and Game objects:

Create new player and new game and pass canvas properties in `script.js`:

```
 class Player {
        constructor(game) {
            this.game = game;
            this.width = 120;
            this.height = 190;
            this.x = 20;
            this.y = 100;
            this.speedY = 0;
        }
        update() {
            this.y += this.speedY;

        }
```

```
        draw(context) {
            context.fillRect(this.x, this.y, this.width, this.height);

        }
    class Game {
        constructor(width, height) {
            this.width = width;
            this.height = height;
            this.player = new Player(this);
        }
        update() {
            this.player.update();

        }
        draw(context) {
            this.player.draw(context);
```

Animation Loop:

Use Request Animation Frame method to tell the browser that we wish to perform an animation and it requests that the browser calls a specified function to update an animation before the next repaint in `script.js`:

```
  const game = new Game(canvas.width, canvas.height);
    //animation loop
    function animate() {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        game.update();
        game.draw(ctx);
        requestAnimationFrame(animate);
    }
    animate();
```

Change `this.speedY` to any numerical speed other than 0, the vertical movement as the animation of the player appears:

player vertical movement as animation appears.PNG

Keyboard Inputs:

Create key input handler to control key up and key down in `script.js`:

```
class InputHandler {
        constructor(game) {
            this.game = game;
            window.addEventListener('keydown', function (e) {
                if (e.key === 'ArrowUp') {
                    this.game.keys.push(e.key);
                }
                console.log(this.game.keys)
```

Console Error: Uncaught TypeError in Console element inspect. DEBUGGING: Use `window.addEventListener('keydown', e => {` in the input handler class.

Now when push up arrow keys, in console these ArrowUp actions all displayed:

arrowup actions displayed in console.PNG

Change key index to make continuous arrowup avoid from appearing massive:

```
  window.addEventListener('keydown', e => {
                if ((e.key === 'ArrowUp') &&
this.game.keys.indexOf(e.key) === - 1) {
                    this.game.keys.push(e.key);
```

Now continuous arrowup appears singularly:

continuous arrow up appears singularly.PNG

Change max speed as 3, now we can update to set: if user press arrow up is going up -maxspeed, if user press arrow down is going down +maxspeed. Else if stay still, stay still:

```
update() {
if (this.game.keys.includes('ArrowUp'))
this.speedY =-this.maxSpeed;
else if (this.game.keys.includes('ArrowDown'))
this.speedY = this.maxSpeed;
else this.speedY = 0;
```

Creating projectiles:

Create Projectile class to handle horizontal projectiles from the player:

```
 class Projectile {
        constructor(game, x, y) {
            this.game = game;
            this.x = x;
            this.y = y;
            this.width = 10;
            this.height = 3;
            this.speed = 3;
            this.markedForDeletion = false;
        }
  update() {
    this.x += this.speed;
    if (this.x > this.game.width * 0.8) this.markedForDeletion = true;
        }
        draw(context) {
            context.fillStyle = 'yellow';
            context.fillRect(this.x, this.y, this.width, this.height);
```

Also use filter method to filter out single projectile from projectiles:

```
this.projectiles = this.projectiles.filter(projectile =>
!projectile.markedForDeletion);
```

Now if player hit Space the horizontal projectiles displayed and shoot from top:

yellow projectiles shoots.PNG

Periodic Events:

Set to manage the ammo time to delta time as a periodic event:

```
        this.maxAmmo = 50;
        this.ammoTimer = 0;
        this.ammoInterval = 500;
    }
    update(deltaTime) {
        this.player.update();
        if (this.ammoTimer > this.ammoInterval) {
            if (this.ammo < this.maxAmmo) this.ammo++;
            this.ammoTimer = 0;
        } else {
            this.ammoTimer += deltaTime;
        }
```

Drawing Game UI:

Now draw the backup ammos as a for loop:

```
class UI {
    constructor(game){
        this.game = game;
        this.fontSize = 25;
        this.fontFamily = 'Helvetica';
        this.color = 'yellow';
    }
    draw(context){
        //ammo
        context.fillStyle = this.color;
        for (let i = 0; i < this.game.ammo; i++){
            context.fillRect(20 + 5 * i , 50, 3, 20);
        }
    }
```

The ammos are now slowly recharging until it reaches the maximum load:

ammos are slowly recharging.PNG

Base Enemy Class:

Create a parent enemy class:

```
class Enemy {
    constructor(game){
        this.game = game;
        this.x = this.game.width;
        this.speedX = Math.random() * -1.5 -0.5;
        this.markedForDeletion  = false;
    }
    update(){}
    draw(){}
```

Create a derived enemy class Angler1 extended from Enemy:

```
  update(deltaTime) {
            this.player.update();
            if (this.ammoTimer > this.ammoInterval) {
                if (this.ammo < this.maxAmmo) this.ammo++;
                this.ammoTimer = 0;
            } else {
                this.ammoTimer += deltaTime;
            }
            this.enemies.forEach(enemy => {
                enemy.update();
            });
            this.enemies = this.enemies.filter(enemy =>
!enemy.markedForDeletion);
            if (this.enemyTimer >  this.enemyInterval &&
!this.gameOver){
                this.addEnemy();
                this.enemyTimer = 0;
            } else {
                this.enemyTimer += deltaTime;
            }
```

Now we are adding enemies:

enemies are coming up.PNG

Minimize enemies to 0.2 size:

minimized enemies.PNG

Collision detection between rectangles:

Create a checkCollision function to check for collision, delete enemy if collided:

```
    checkCollision(rect1, rect2){
            return(   rect1.x < rect2.x + rect2.width &&
                      rect1.x + rect1.width > rect2.x &&
                      rect1.y < rect2.y + rect2.height &&
                      rect1.height + rect1.y > rect2.y)
  enemy.markedForDeletion = true;
```

Also show enemy scores as their lives count:

```
  if (this.checkCollision(projectile, enemy)){
                enemy.lives--;
                projectile.markedForDeletion = true;
                if (enemy.lives <= 0){
                       enemy.markedForDeletion = true;
                       this.score += enemy.score;
                         }
```

Now enemies lives reduced once they hit by ammos projectiles until they are killed:

enemies lives reduced until killed once hit by ammos and scores showed.PNG

Drawing Games Score:

To give the score count more textures, add shadow effect, save and restore it:

```
  draw(context) {
        context.save();
        context.fillStyle = this.color;
        context.shadowOffsetX = 2;
        context.shadowOffsetY = 2;
        context.shadowColor = 'black';
        context.font = this.fontSize + 'px' + this.fontFamily;
        //score
        context.fillText('Score: ' + this.game.score, 20, 40);
        //ammo
        context.restore();
```

more textures added on score count.PNG

Win and Lose Condition:

Set up game over messages if the player win or lose:

If the player won, congrats them

Cheer them with “well done”

If the player lost, encourage them to try again next time:

Set up the font sizes and spacing properly:

```
  if (this.game.gameOver) {
                context.textAlign = 'center';
                let message1;
                let message2;
                if (this.game.score > this.game.winningScore) {
                    message1 = 'You Win!';
                    message2 = 'Well Done!';
                } else {
                    message1 = 'You Lose!';
                    message2 = 'Try Again Next Time!';
                }
                context.font = '50px' + this.fontFamily;
                context.fillText(message1, this.game.width * 0.5,
this.game.height * 0.5 - 40);
                context.font = '25px' + this.fontFamily;
                context.fillText(message2, this.game.width * 0.5,
this.game.height * 0.5 + 40);
            }
            context.restore();
          }
        }
```

The player won and the congrats messages shows:

player won congrats messages.PNG

Counting Game Time:

Set game time limit, if exceed time limit, game over:

```
if (!this.gameOver) this.gameTime += deltaTime;
if (this.gameTime > this.timeLimit) this.gameOver = true;
```

Use toFixed() method to display timer on top:

```
//timer
    const formattedTime = (this.game.gameTime * 0.001).toFixed(1);
    context.fillText('Timer:' + formattedTime, 20, 100);
```

timer displayed on top.PNG

Animated parallax backgrounds:

Add layers and other animated assets in `index.html`.

Define the speed of layers in `script.js`:

```
class Layer {
        constructor(game, image, speedModifier){
            this.game = game;
            this.image = image;
            this.speedModifier = speedModifier;
            this.width = 1768;
            this.height =  500;
            this.x = 0;
            this.y = 0;
        }
        update(){
            if (this.x <= -this.width) this.x = 0;
            else this.x -= this.game.speed * this.speedModifier;
        }
        draw(context){
            context.drawImage(this.image, this.x, this.y);
        }
    }
```

For the Background, get layer 1 move in motion:

```
  class Background {
        constructor(game){
            this.game = game;
            this.image1 = document.getElementById('layer1');
            this.layer1 = new Layer(this.game, this.image1, 1);
            this.layers = [this.layer1];
    update(){}
    draw() {}
```

Also add background in Game class:

```
  this.background = new Background(this);
```

layer 1 move in motion in background.PNG

Changing background layers scenes and accelerating player moving speed:

```
 constructor(game) {
            this.game = game;
            this.image1 = document.getElementById('layer1');
            this.image2 = document.getElementById('layer2');
            this.image3 = document.getElementById('layer3');
            this.image4 = document.getElementById('layer4');
            this.layer1 = new Layer(this.game, this.image1, 5);
            this.layer2 = new Layer(this.game, this.image2, 5);
            this.layer3 = new Layer(this.game, this.image3, 5);
            this.layer4 = new Layer(this.game, this.image4, 5);
            this.layers = [this.layer1, this.layer2, this.layer3,
this.layer4];
        }
```

scenes switching amongst 4 layer backgrounds.PNG

Sprite animation with JavaScript:

Add players assets in `index.html`.

Then draw the players in `script.js`:

```
context.drawImage(this.image, this.x, this.y);
```

characters drawed.PNG

Make the single seahorse vivid:

```
  //sprite animation
            if (this.frameX < this.maxFrame){
                this.frameX ++;
            } else{
                this.frameX = 0;
            }
```

vivid animated single seahorse.PNG

Add animated effects to the enemies anglers as well:

Similarly to seahorse, import enemies assets and define frame width and heights:

```
context.drawImage(this.image, this.frameX * this.width, this.frameY *
this.height , this.width, this.height, this.x, this.y, this.width,
this.height);
```

angler1 drawed.PNG

multiple angler1 on move.PNG

Create angler2 as same as creating angler1:

```
  class Angler2 extends Enemy {
        constructor(game) {
            super(game);
            this.width = 213;
            this.height = 165;
            this.y = Math.random() ...}
```

both anglers added.PNG

Add lucky fish class extended from Enemy class:

```
  class LuckyFish extends Enemy {
        constructor(game) {
            super(game);
            this.width = 99;
            this.height = 95;
            this.y = Math.random() * (this.game.height * 0.9 -
this.height);
            this.image = document.getElementById('lucky');
            this.frameY = Math.floor(Math.random() * 2);
            this.lives = 3;
            this.score = 15;
            this.type = 'lucky'
        }
```

all three kinds of fishes displayed.PNG

Power up the charges to better win the game:

```
  if (this.powerUp){
                if (this.powerUpTimer > this.powerUpLimit){
                    this.powerUpTimer = 0;
                    this.powerUp = false;
                    this.frameY = 0;
                } else {
                    this.powerUpTimer += deltaTime;
                    this.frameY = 1;
                    this.game.ammo += 0.1;
                }
            }
```

So now the seahorse lifespan is powered up and can better win the game:

power up the charges.PNG

If add a bottom projectile:

```
   shootBottom() {
            if (this.game.ammo > 0) {
                this.projectiles.push(new Projectile(this.game, this.x
+ 80, this.y + 175));
            }
        }
```

double ammos top and bottom.PNG

Import Google Font Bangers cursive on:

https://fonts.google.com/specimen/Bangers

into `index.html` and `style.css`.

Replace `this.fontFamily` with Bangers font to make the rendering effective.

win message with bangers font.PNG

lose message with bangers font.PNG

Add other characters like drones and adjust their moving speed varyingly.
Now all characters displayed:
all characters displayed.PNG
Add smokes and fire effects:

```
class SmokeExplosion extends Explosion {
        constructor(game, x, y) {
            super(game, x, y);
            this.image = document.getElementById('smokeExplosion');
        }
    }

    class FireExplosion extends Explosion {
        constructor(game, x, y) {
            super(game, x, y);
            this.image = document.getElementById('fireExplosion');
        }
    }
```

smoke and fire explosion effects added.PNG