Spike: Task 30
Title: Web Game – Plane War

Author: Wai Hong Chong, 101783959

Goals / deliverables:

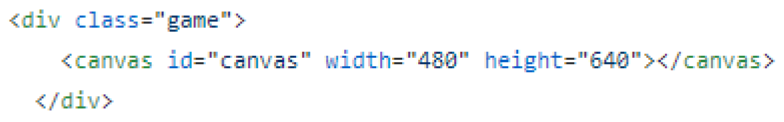
- Code
  - With constructor design pattern and prototype design pattern (for those expandable object)
  - Mouse input (event handle)
  - Draw and move image
  - Collison
- Able to create an web application that can use the mouse to move the image, and familiar with the Collison test techniques.

Technologies, Tools, and Resources used:

- https://addyosmani.com/resources/essentialjsdesignpatterns/book/
- https://stackoverflow.com/
- https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
- https://www.w3schools.com/js/js_object_constructors.asp
- Visual Studio Code

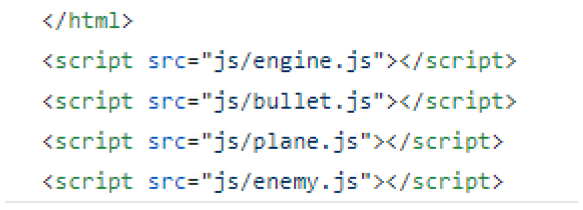
Tasks undertaken:

- Create an element that will be used to rendered the entire game, which is canvas in the HTML file.



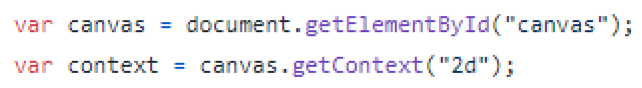
```
<div class="game">
    <canvas id="canvas" width="480" height="640"></canvas>
  </div>
```

- Then, you can choose either develop the JavaScript under the HTML file or a new JS file. in my case, I choose to implement in new JS files, coz it will make the code look cleaner.
  Note: you can create one or many JS file to contain the codes, it's up to you, because it not like C++, that we need to include the header file before we use it.



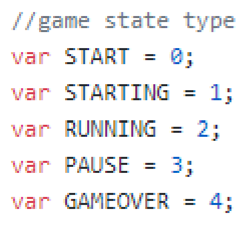
```
</html>
<script src="js/engine.js"></script>
<script src="js/bullet.js"></script>
<script src="js/plane.js"></script>
<script src="js/enemy.js"></script>
```

- To able render graphics on the canvas element, we need to create a canvas variable (JavaScript) to store the canvas element (HTML) and create context variable to store the 2D rendering context.

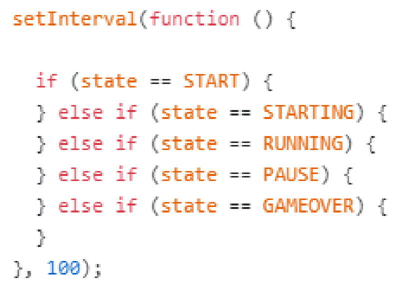


```
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");
```

- Implement a simple game state controller.



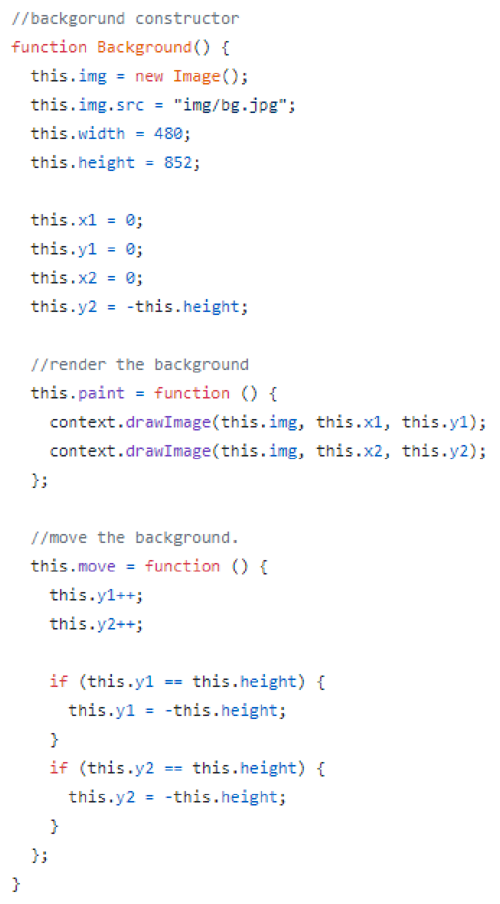
```
//game state type
var START = 0;
var STARTING = 1;
var RUNNING = 2;
var PAUSE = 3;
var GAMEOVER = 4;
```



```
setInterval(function () {

  if (state == START) {
  } else if (state == STARTING) {
  } else if (state == RUNNING) {
  } else if (state == PAUSE) {
  } else if (state == GAMEOVER) {
  }
}, 100);
```

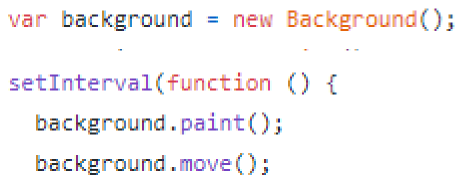
The setInterval is a JavaScript built-in function, that will automatically run the code every 100 milliseconds (you can set it to any time long you want). And it will only stop until cleatInterval() is called. In this case, we will just simply be using the setInterval to represent the game main-loop, so that it will keep running if-else statement like the game-state management. The left-side image is the state that we have in this game, and the right-side image is how to code the game-state manager.

- Setup the background feature, so that the background will looks like moving up.



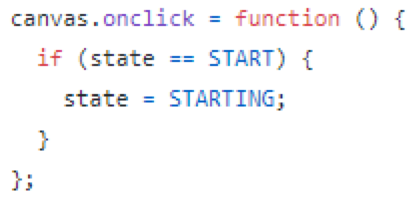
```
//backgorund constructor
function Background() {
  this.img = new Image();
  this.img.src = "img/bg.jpg";
  this.width = 480;
  this.height = 852;

  this.x1 = 0;
  this.y1 = 0;
  this.x2 = 0;
  this.y2 = -this.height;

  //render the background
  this.paint = function () {
    context.drawImage(this.img, this.x1, this.y1);
    context.drawImage(this.img, this.x2, this.y2);
  };

  //move the background.
  this.move = function () {
    this.y1++;
    this.y2++;

    if (this.y1 == this.height) {
      this.y1 = -this.height;
    }
    if (this.y2 == this.height) {
      this.y2 = -this.height;
    }
  };
}
```

As you can see that we implement the background by constructor. The background will not take any parameter to construct or the function., coz we will have only one background in this game, and the other variables like width and height are declaring the image width and height in pixels. The other variables like x1, x2, y1,and y2 are for animation purpose, you can see that we render the background twice to make it looks more like moving up. And the step function is to keep the update the y position.



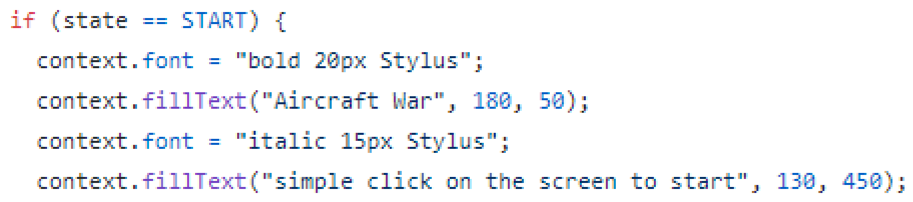
```
var background = new Background();

setInterval(function () {
  background.paint();
  background.move();
```

Then, we need to declare a new background object, so that we can keep the background moving when we get in the web page. To ensure the background will moving at all time without limit by the game state, we just need to run the render and move function before we do the game state management function.



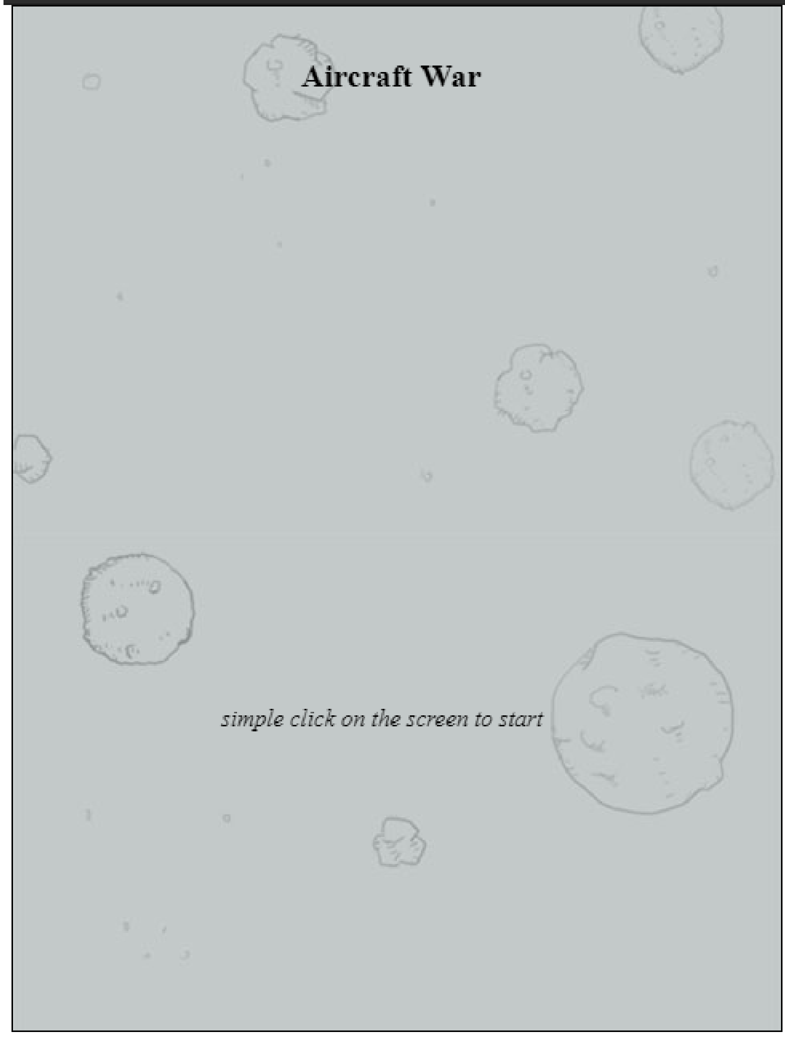
```
canvas.onclick = function () {
  if (state == START) {
    state = STARTING;
  }
};
```

The picture above will change the game state to starting once the user click on the canvas and also in “start” state.

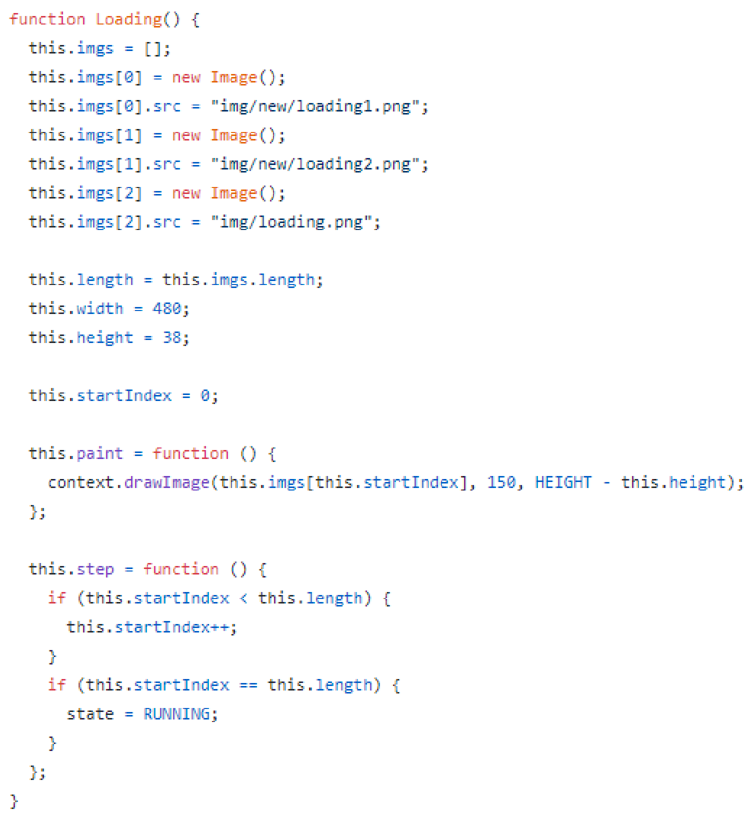


```
if (state == START) {
  context.font = "bold 20px Stylus";
  context.fillText("Aircraft War", 180, 50);
  context.font = "italic 15px Stylus";
  context.fillText("simple click on the screen to start", 130, 450);
```

In the start state, you can just simply write some text like the game name or hint on how to start the game. Then you will get a similar design like the following picture:

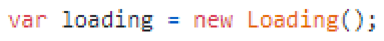


- Setup the loading animation.



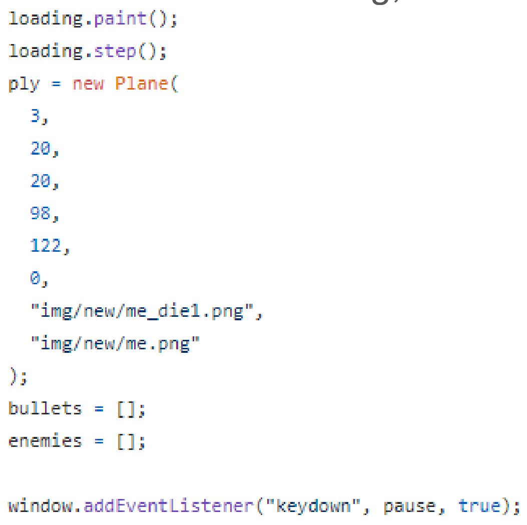
```
function Loading() {
  this.imgs = [];
  this.imgs[0] = new Image();
  this.imgs[0].src = "img/new/loading1.png";
  this.imgs[1] = new Image();
  this.imgs[1].src = "img/new/loading2.png";
  this.imgs[2] = new Image();
  this.imgs[2].src = "img/loading.png";

  this.length = this.imgs.length;
  this.width = 480;
  this.height = 38;

  this.startIndex = 0;

  this.paint = function () {
    context.drawImage(this.imgs[this.startIndex], 150, HEIGHT - this.height);
  };

  this.step = function () {
    if (this.startIndex < this.length) {
      this.startIndex++;
    }
    if (this.startIndex == this.length) {
      state = RUNNING;
    }
  };
}
```

The picture above is the code to render the loading process as an animation. It quite like the background feature, but the different is just we have three images in this object, so to representing the animation, we need to render different image by changing the startIndex. And the render function will render difference image, but with same x position and y position. Also, the step function here, is to update the startIndex, to make the animation more real, and once the startIndex meets the limit, then it will change the game state.



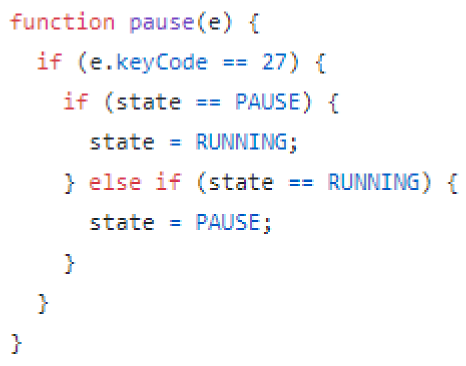
```
var loading = new Loading();
```

to able to use the loading, we need to create an loading as an object.



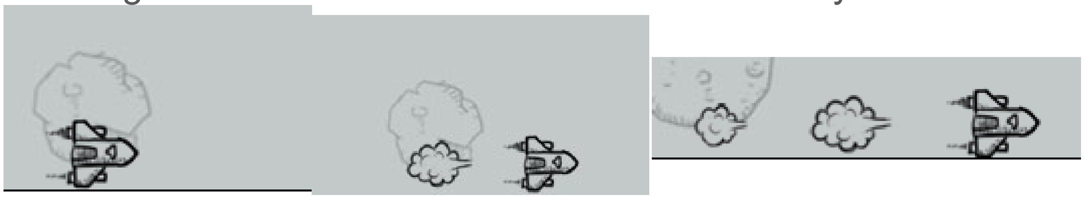
```
loading.paint();
loading.step();
ply = new Plane(
  3,
  20,
  20,
  98,
  122,
  0,
  "img/new/me_die1.png",
  "img/new/me.png"
);
bullets = [];
enemies = [];

window.addEventListener("keydown", pause, true);
```

Then in the “starting” state, it will do the loading animation, and declare a player, array of bullets and array of enemies. Also, it will add a enlister to pause the game.



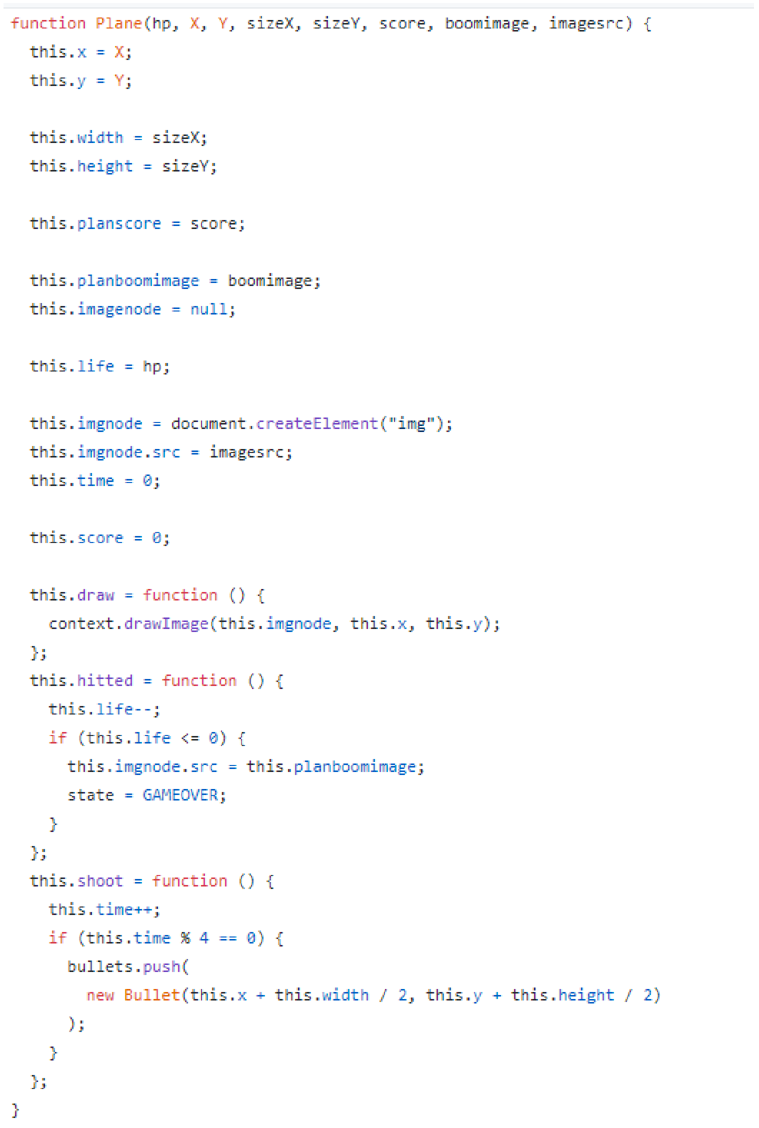
```
function pause(e) {
  if (e.keyCode == 27) {
    if (state == PAUSE) {
      state = RUNNING;
    } else if (state == RUNNING) {
      state = PAUSE;
    }
  }
}
```

When the user pressed on “ESC”, it will change the state between Running and Pause. Note the “27” is the “ESC” keycode.



Note: that the state will update to “RUNNING” once the loading image is done.

- We will take about the “RUNNING” state later, coz we need to do more implementation before that, like the player, enemy and bullet object.
- Implement the plane object function.



```
function Plane(hp, X, Y, sizeX, sizeY, score, boomimage, imagesrc) {
  this.x = X;
  this.y = Y;

  this.width = sizeX;
  this.height = sizeY;

  this.planscore = score;

  this.planboomimage = boomimage;
  this.imagenode = null;

  this.life = hp;

  this.imgnode = document.createElement("img");
  this.imgnode.src = imagesrc;
  this.time = 0;

  this.score = 0;

  this.draw = function () {
    context.drawImage(this.imgnode, this.x, this.y);
  };
  this.hitted = function () {
    this.life--;
    if (this.life <= 0) {
      this.imgnode.src = this.planboomimage;
      state = GAMEOVER;
    }
  };
  this.shoot = function () {
    this.time++;
    if (this.time % 4 == 0) {
      bullets.push(
        new Bullet(this.x + this.width / 2, this.y + this.height / 2)
      );
    }
  };
}
```

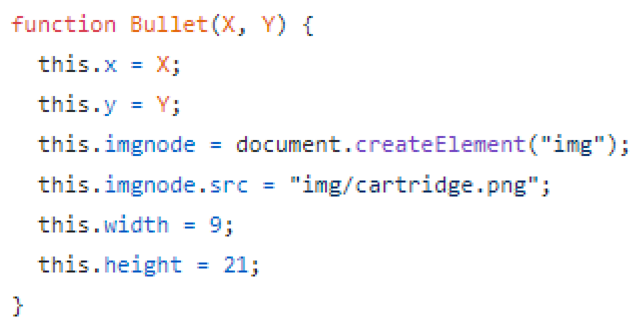
In “Plane” function is also similar to the “Background”, but it contain more variable, like the plane position (x and y), the plane size (width, height), plane’s image (normal and crash ‘s image), plane, life, score. Then, the plane will also have the draw function, which is to render the image on the screen.
The hitted function is to reduce the plane’s life and change the image source to crash image and update the game state to game over.
The shoot function will automatically fire a bullet every 4 second.

- Implement the enemy object function.

```
function enemy(hp, X, Y, sizeX, sizeY, Score, boomimage, imagesrc) {
  this.x = X;
  this.y = Y;

  this.width = sizeX;
  this.height = sizeY;

  this.crashImg = boomimage;
  this.imagenode = null;

  this.crash = false;
  this.life = hp;

  this.score = Score;

  this.imgnode = document.createElement("img");
  this.imgnode.src = imagesrc;

  this.draw = function () {
    context.drawImage(this.imgnode, this.x, this.y);
  };
  this.move = function () {
    this.y += 2;
  };
  this.checkHit = function (object) {
    return (
      object.y + object.height > this.y &&
      object.x + object.width > this.x &&
      object.y < this.y + this.height &&
      object.x < this.x + this.width
    );
  };
  this.hitted = function () {
    this.life--;
    if (this.life == 0) {
      this.imgnode.src = this.crashImg;
      this.crash = true;
    }
  };
}
```

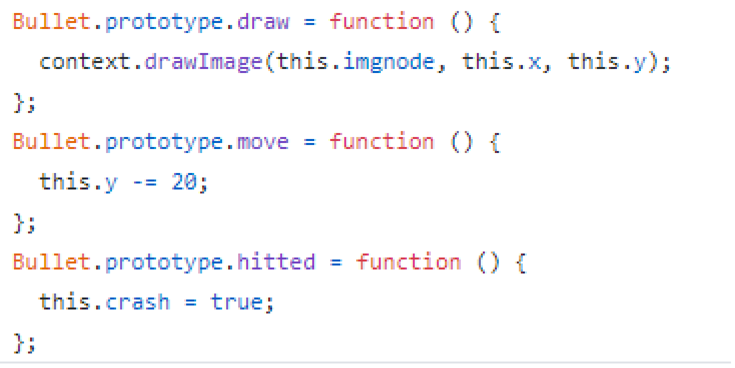
The “Enemy” function is similar to the Plane, they all having the same parameter, but the “enemy” have some different of method like the move and checkHit.
The move method is to set the enemy move by same velocity.
The checkHit method is to return if any object happen collision with the enemy object.
The rest method is same as the Plane function.

- Implement Bullet object function.



```
function Bullet(X, Y) {
  this.x = X;
  this.y = Y;
  this.imgnode = document.createElement("img");
  this.imgnode.src = "img/cartridge.png";
  this.width = 9;
  this.height = 21;
}
```

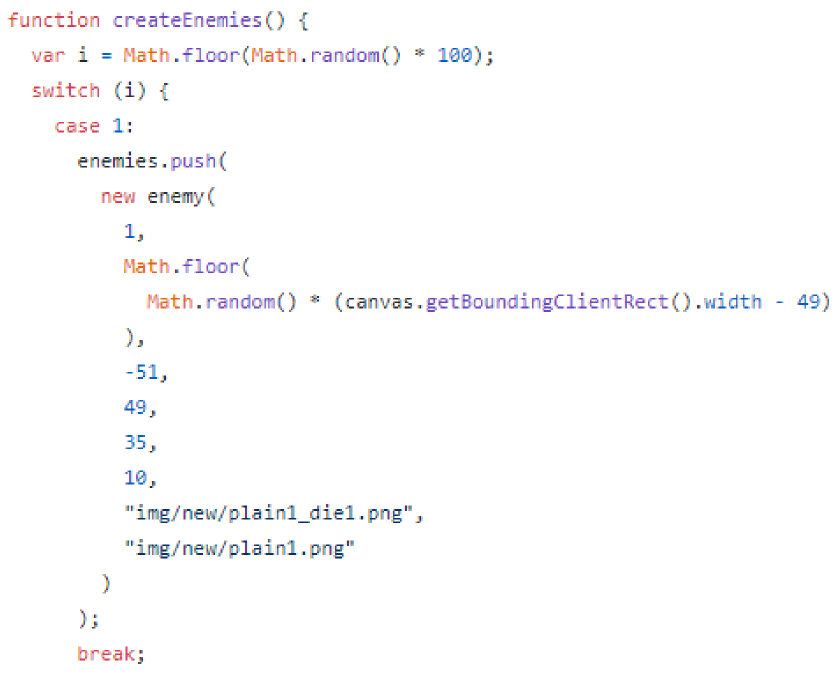
The picture above is the constructor for Bullet object, it will only get position x and y from the Plane function, which is the middle of the player plane.



```
Bullet.prototype.draw = function () {
  context.drawImage(this.imgnode, this.x, this.y);
};
Bullet.prototype.move = function () {
  this.y -= 20;
};
Bullet.prototype.hitted = function () {
  this.crash = true;
};
```

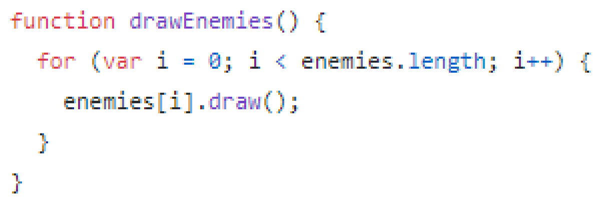
The picture above is about the three functions that the bullet will have, which is just to render the bullet image, and move the bullet vertically with a velocity and also set the image to crash when hitted something.

- Implement functions that will automatically create, move, render, check Hit and remove Enemies.

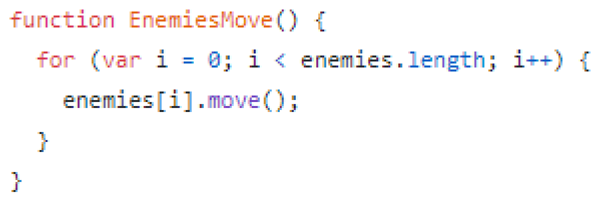


```
function createEnemies() {
  var i = Math.floor(Math.random() * 100);
  switch (i) {
    case 1:
      enemies.push(
        new enemy(
          1,
          Math.floor(
            Math.random() * (canvas.getBoundingClientRect().width - 49)
          ),
          -51,
          49,
          35,
          10,
          "img/new/plain1_die1.png",
          "img/new/plain1.png"
        )
      );
      break;
```

createEnemies method will auto push a new enemy into the enemies' array, once it meets the switch condition. In my case, I got three type of enemy, but I only show one of it here, because the other is same almost same the switch case, the different is the image source, width, height and the life.

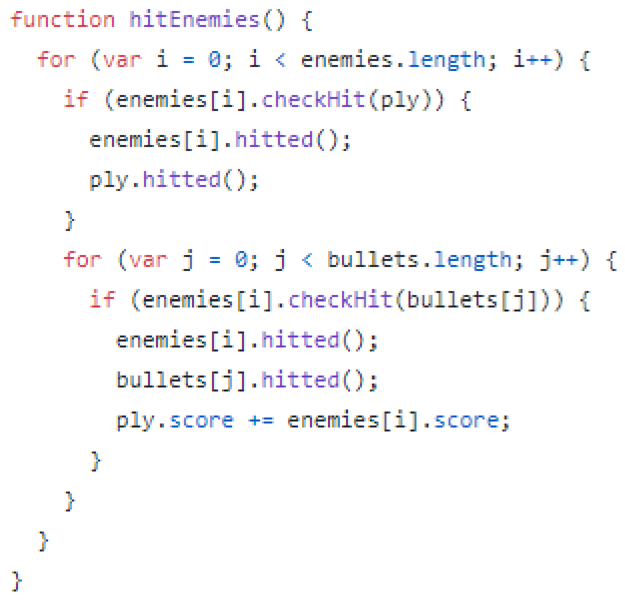


```
function drawEnemies() {
  for (var i = 0; i < enemies.length; i++) {
    enemies[i].draw();
  }
}
```



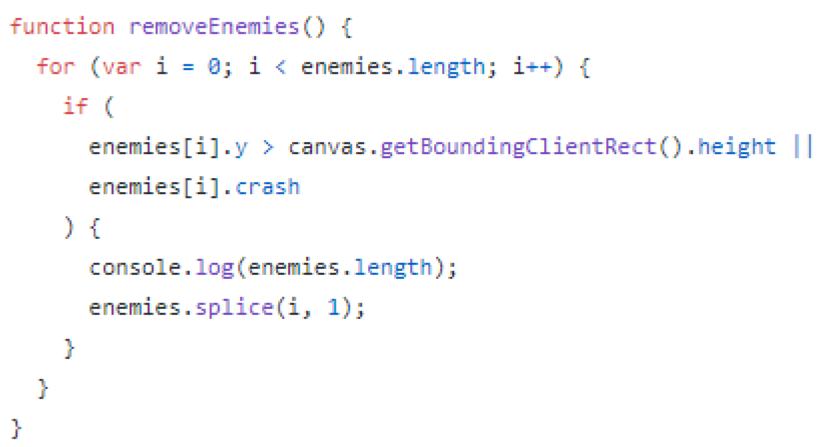
```
function EnemiesMove() {
  for (var i = 0; i < enemies.length; i++) {
    enemies[i].move();
  }
}
```

the drawEnemies and EnemiesMove is just to call render functiuon and move function for each enemies.



```
function hitEnemies() {
  for (var i = 0; i < enemies.length; i++) {
    if (enemies[i].checkHit(ply)) {
      enemies[i].hitted();
      ply.hitted();
    }
    for (var j = 0; j < bullets.length; j++) {
      if (enemies[i].checkHit(bullets[j])) {
        enemies[i].hitted();
        bullets[j].hitted();
        ply.score += enemies[i].score;
      }
    }
  }
}
```

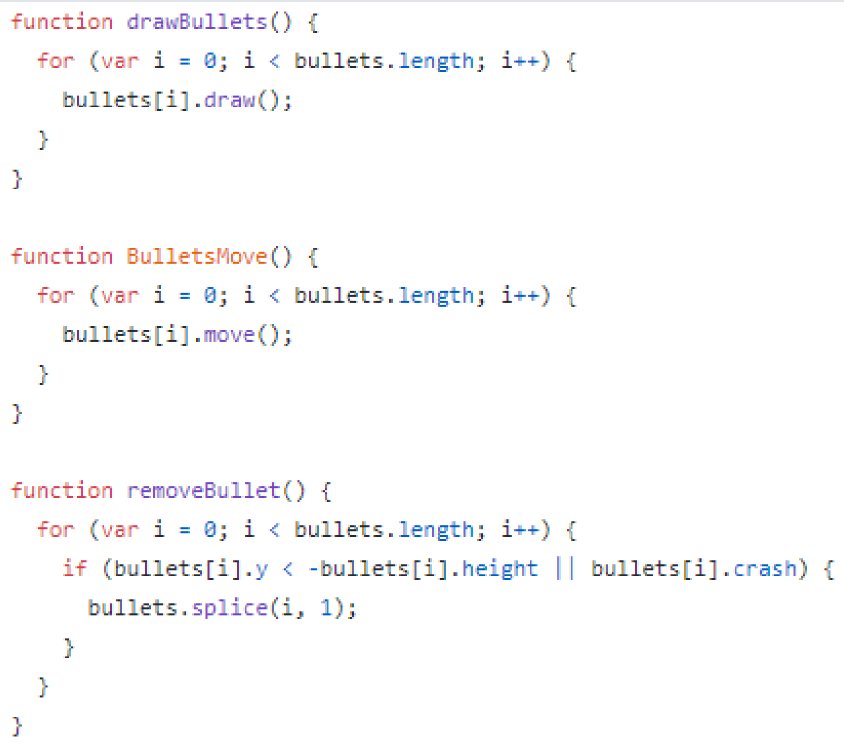
The hitEnemies will check if the enemies object hitted player's plane or the bullet hitted the enemy's plane.



```
function removeEnemies() {
  for (var i = 0; i < enemies.length; i++) {
    if (
      enemies[i].y > canvas.getBoundingClientRect().height ||
      enemies[i].crash
    ) {
      console.log(enemies.length);
      enemies.splice(i, 1);
    }
  }
}
```

The removeEnemies function will check all the enemy if the enemy is going out the screen or if the enemy is crash. If then, it will delete the enemy from the array.

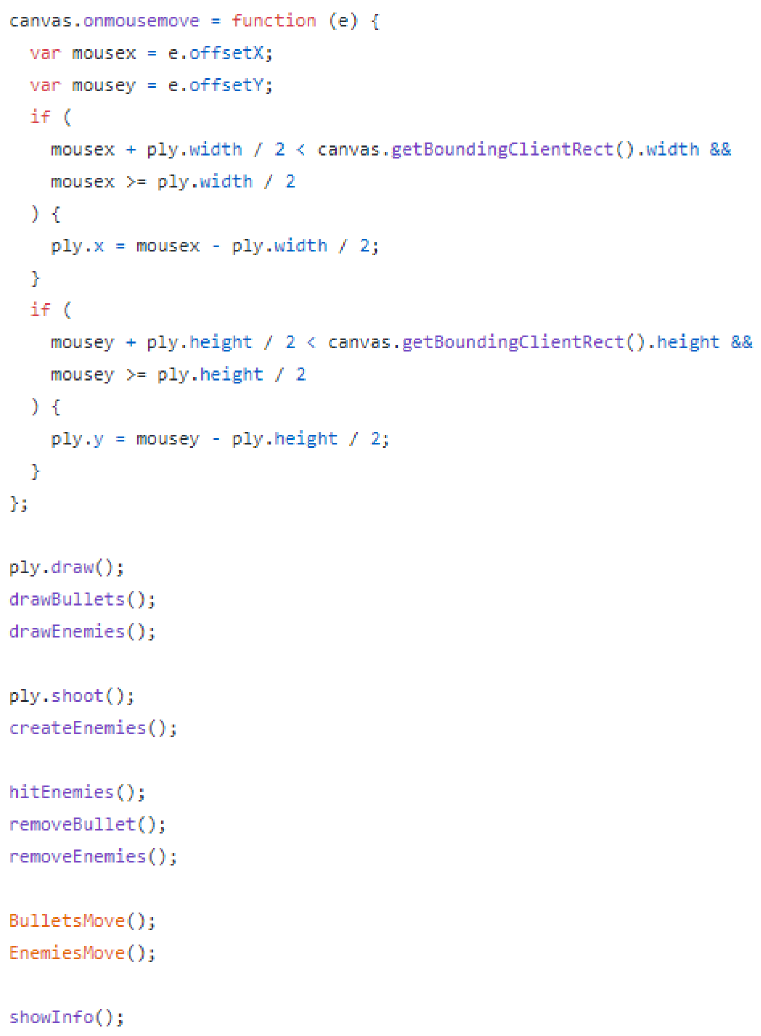
- Implement the function to draw, move and remove the Bullet's array.



```
function drawBullets() {
  for (var i = 0; i < bullets.length; i++) {
    bullets[i].draw();
  }
}

function BulletsMove() {
  for (var i = 0; i < bullets.length; i++) {
    bullets[i].move();
  }
}

function removeBullet() {
  for (var i = 0; i < bullets.length; i++) {
    if (bullets[i].y < -bullets[i].height || bullets[i].crash) {
      bullets.splice(i, 1);
    }
  }
}
```

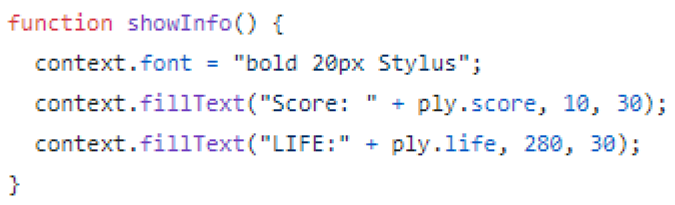
These three functions very similar to the enemies related function, you can refer to it for the logic.

- The second last step is the "RUNNING" state.



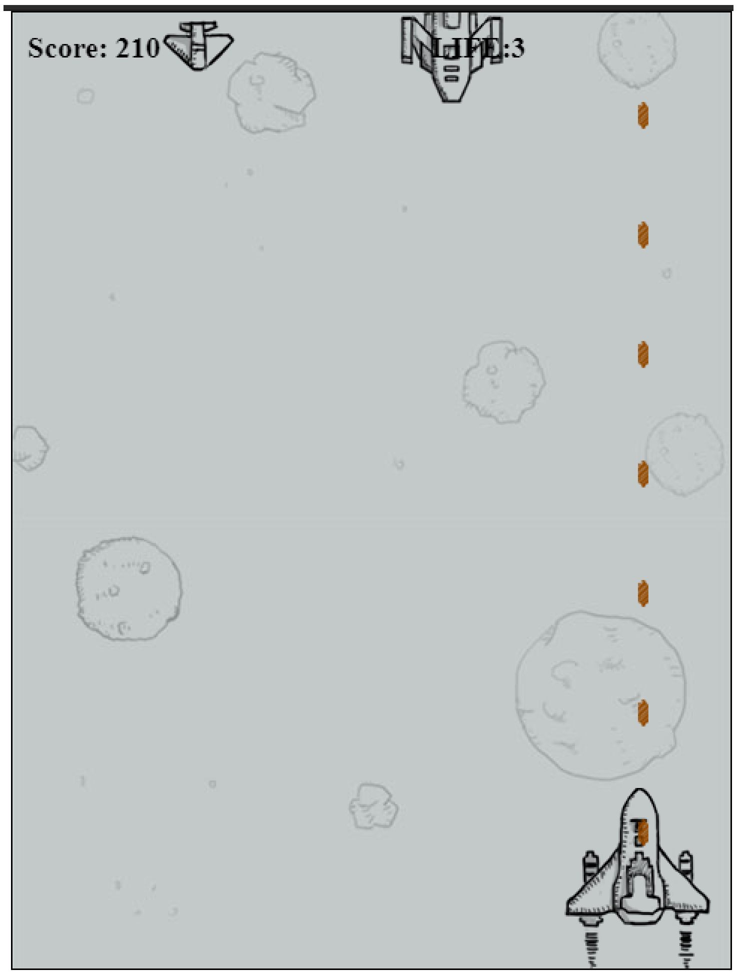
```
canvas.onmousemove = function (e) {
  var mousex = e.offsetX;
  var mousey = e.offsetY;
  if (
    mousex + ply.width / 2 < canvas.getBoundingClientRect().width &&
    mousex >= ply.width / 2
  ) {
    ply.x = mousex - ply.width / 2;
  }
  if (
    mousey + ply.height / 2 < canvas.getBoundingClientRect().height &&
    mousey >= ply.height / 2
  ) {
    ply.y = mousey - ply.height / 2;
  }
};

ply.draw();
drawBullets();
drawEnemies();

ply.shoot();
createEnemies();

hitEnemies();
removeBullet();
removeEnemies();

BulletsMove();
EnemiesMove();

showInfo();
```

The above picture is the game logic for “RUNNING” state.
The onmouse mouse is to connect the player plane with the mouse position.
Then, it will draw all the objects (player, enemies, and bullet), and follow by create new bullet and enemies. Then, check if the enemies been hitted, if then it will remove the bullet, and enemies. Then it will move the Bullets and Enemies position. The last step is to show the score and player life, so we have it in showInfo function. The following is the show info function.

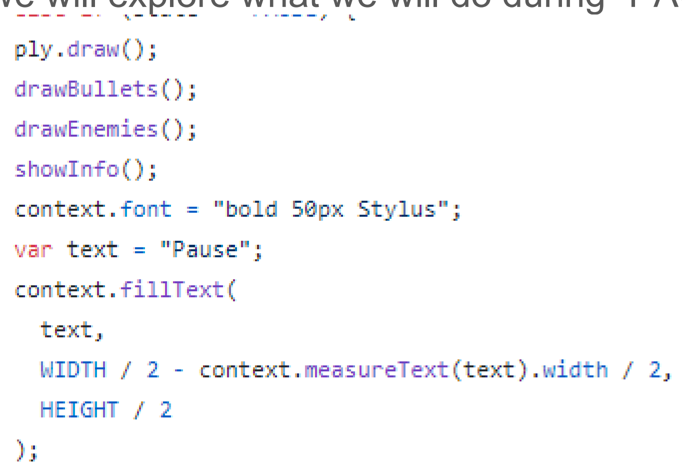


```
function showInfo() {
  context.font = "bold 20px Stylus";
  context.fillText("Score: " + ply.score, 10, 30);
  context.fillText("LIFE:" + ply.life, 280, 30);
}
```

The below image is an example:

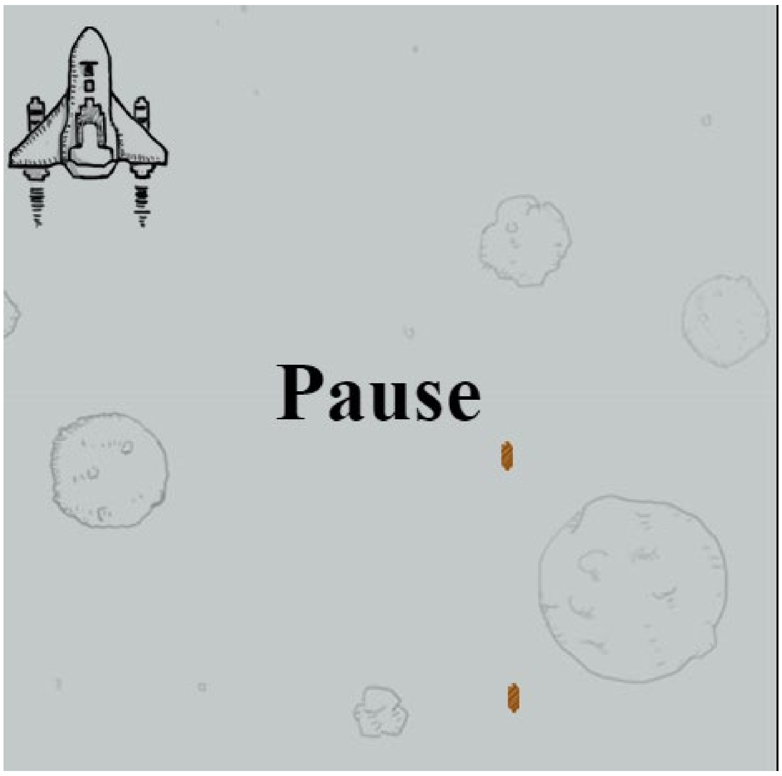


- We discussed about how to change the state to pause before, and here we will explore what we will do during “PAUSE” state.

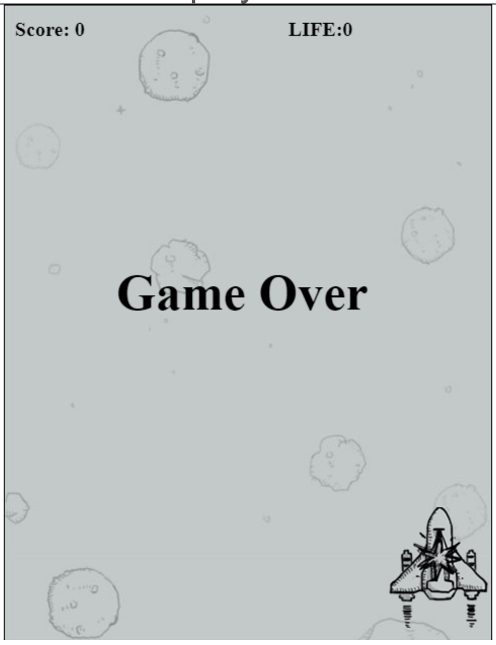


```
ply.draw();
drawBullets();
drawEnemies();
showInfo();
context.font = "bold 50px Stylus";
var text = "Pause";
context.fillText(
  text,
  WIDTH / 2 - context.measureText(text).width / 2,
  HEIGHT / 2
);
```

The above picture is what I doing during “PAUSE” state, which is render the player, bullets, enemies and display the score, life and some text.



- The last step is the "GAME OVER" state, which is just to display all the objects and display a "GAME OVER" text.



What we found out:

We found out that the JavaScript is more scalable than the C++ because the JavaScript is an object-based language based on prototypes, rather than being class-based. Base on these differences, the way you use JavaScript to create hierarchies of objects and to inherit the properties and their values are quite different. even the other language also based on OOP, but I think the JavaScript is more flexible.

Recommendations:

- We can make the “Plane” to be the base object, and just simply extend the Plane with some addition function and variable by using Decorator Pattern, so that the Player and Enemy will have the same base Component.