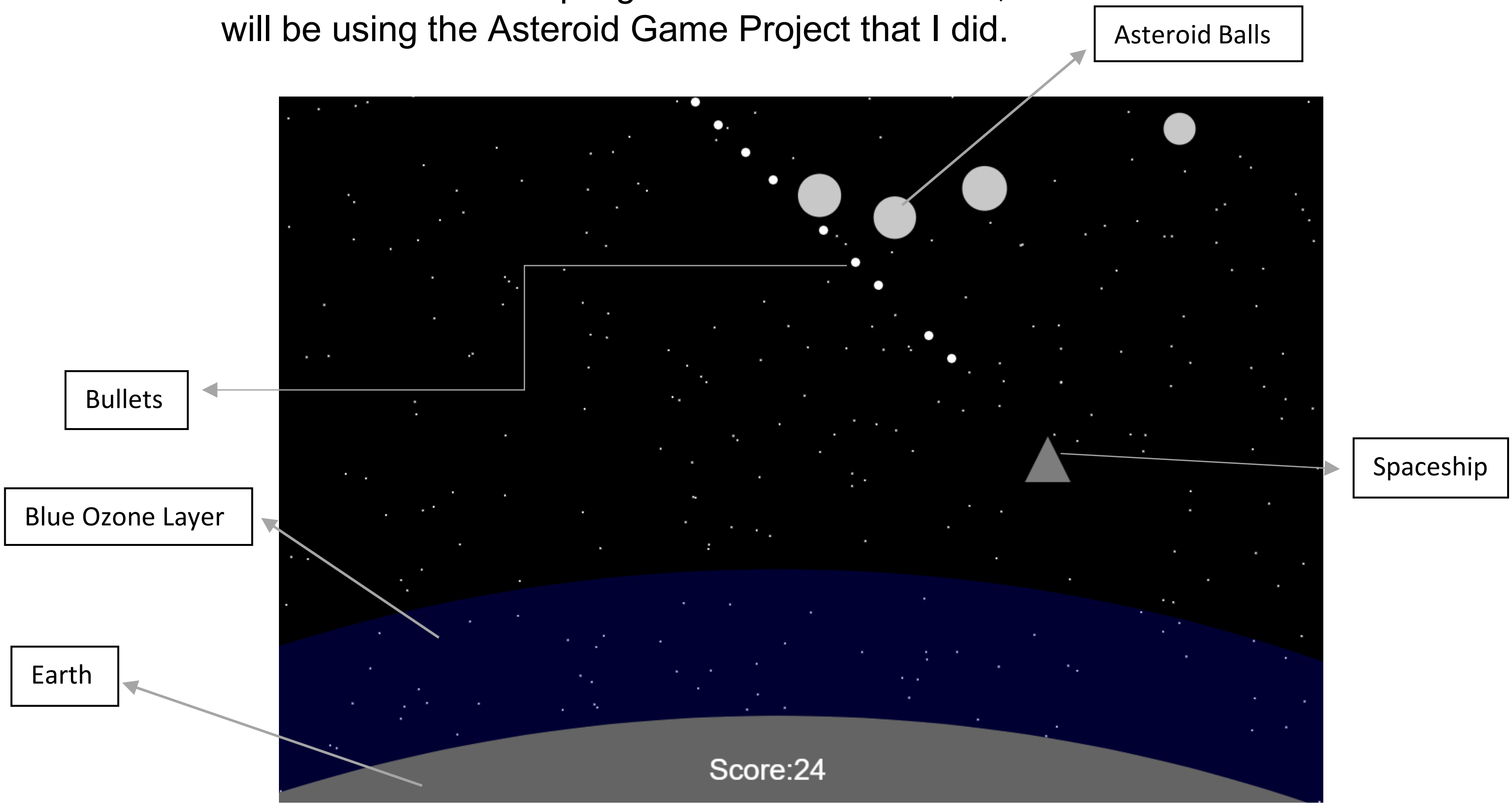
Part 1 – Module Coupling and Cohesion

1. Program Description

   I used the Graphics Programming Project (GPP) that I had done in this semester. GPP is run with JavaScript using Brackets.IO, a text editor used for web development. The program allows the creation of drawings and 3D models to form a variety of structures and animated objects. It also involves the of a physics engine for animating elements on the screen, using vectors and forces.

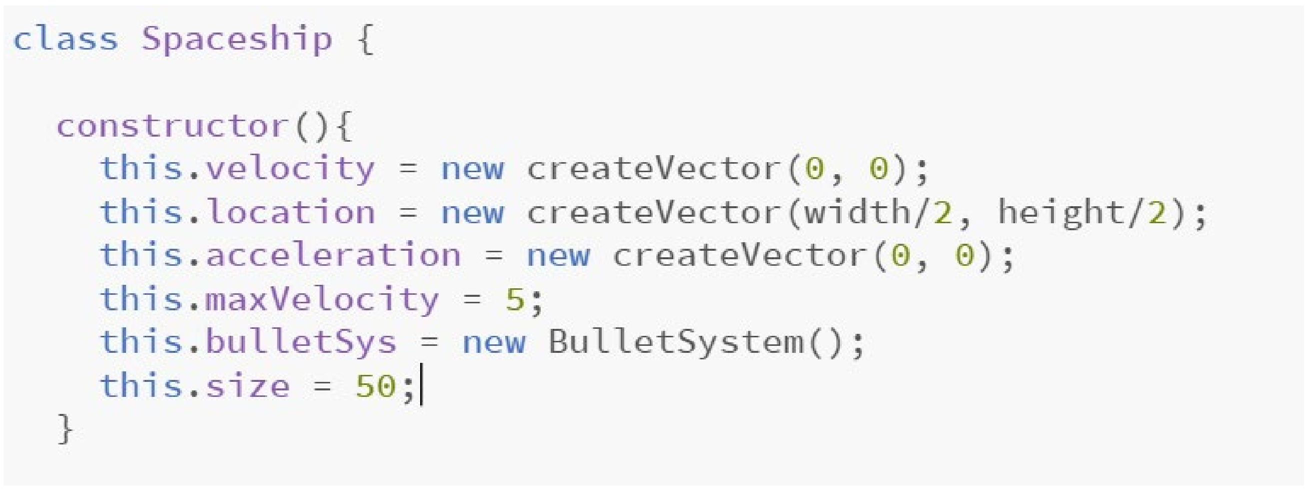
   For both Module Coupling and Module Cohesion, I will be using the Asteroid Game Project that I did.



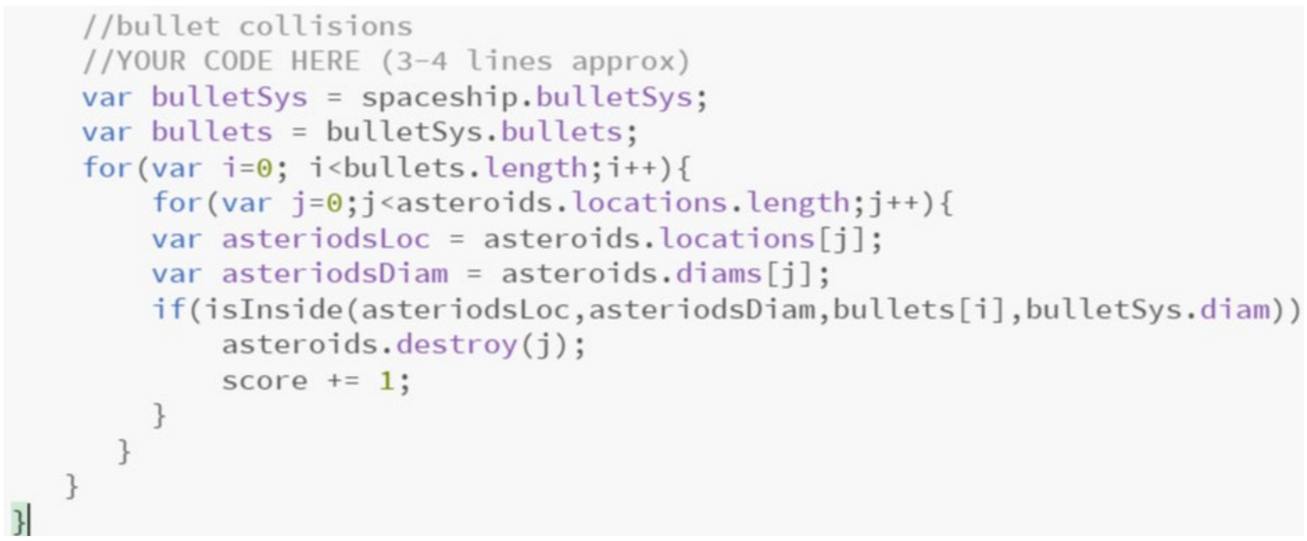
2. Module Coupling

   Module Coupling refers to the manner and degree of interdependence between modules, the strength of the relationship between modules and/or how closely related two routines or modules are. It also measures how closely connected two routines or modules are.

1) Data Coupling



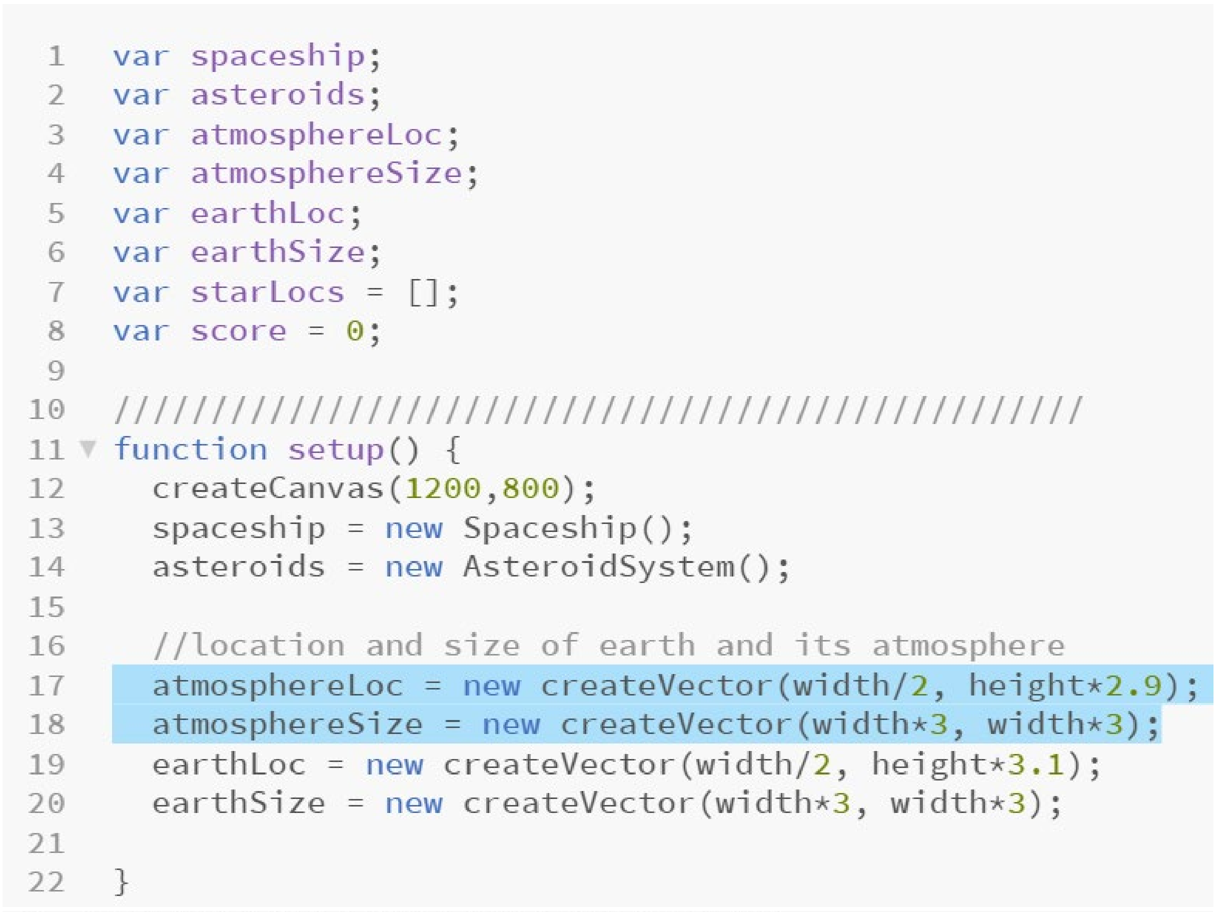
```
class Spaceship {

  constructor(){
    this.velocity = new createVector(0, 0);
    this.location = new createVector(width/2, height/2);
    this.acceleration = new createVector(0, 0);
    this.maxVelocity = 5;
    this.bulletSys = new BulletSystem();
    this.size = 50;
  }
```



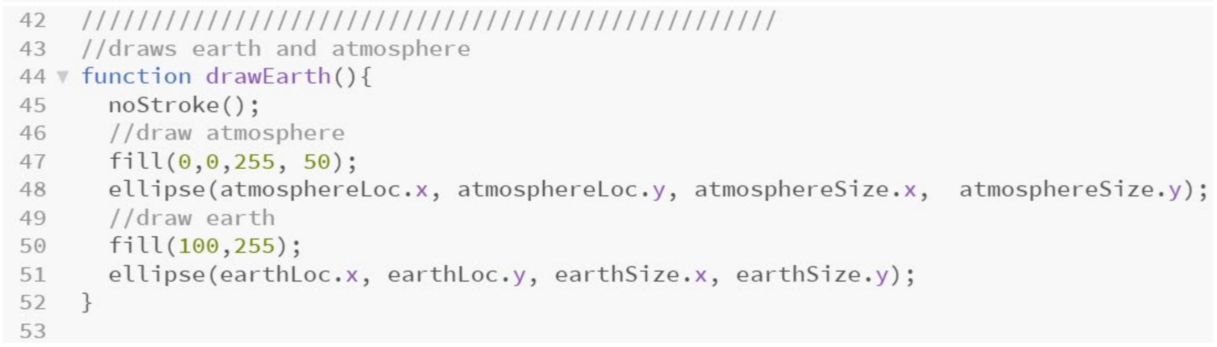
```
    //bullet collisions
    //YOUR CODE HERE (3-4 lines approx)
    var bulletSys = spaceship.bulletSys;
    var bullets = bulletSys.bullets;
    for(var i=0; i<bullets.length;i++){
        for(var j=0;j<asteroids.locations.length;j++){
        var asteriodsLoc = asteroids.locations[j];
        var asteriodsDiam = asteroids.diams[j];
        if(isInside(asteriodsLoc,asteriodsDiam,bullets[i],bulletSys.diam))
            asteroids.destroy(j);
            score += 1;
        }
      }
  }
}
```

The function “this.bulletSys” is created under the constructor functions in the spaceship.js file to check if the bullet is in contact with the asteroid and sends the data for bullet collisions occur so as to attack the incoming asteroids. This is a form of Data Coupling as the output of one module, “this.bulletSys” ,serves as an input for the “var bulletSys” and “bullets” functions. These 3 functions communicate back and forth to create the game where users attack the asteroids efficiently from the spaceship.

2) Common Environment Coupling



```
var spaceship;
var asteroids;
var atmosphereLoc;
var atmosphereSize;
var earthLoc;
var earthSize;
var starLocs = [];
var score = 0;

//////////////////////////////////////////////////
function setup() {
  createCanvas(1200,800);
  spaceship = new Spaceship();
  asteroids = new AsteroidSystem();

  //location and size of earth and its atmosphere
  atmosphereLoc = new createVector(width/2, height*2.9);
  atmosphereSize = new createVector(width*3, width*3);
  earthLoc = new createVector(width/2, height*3.1);
  earthSize = new createVector(width*3, width*3);

}
```



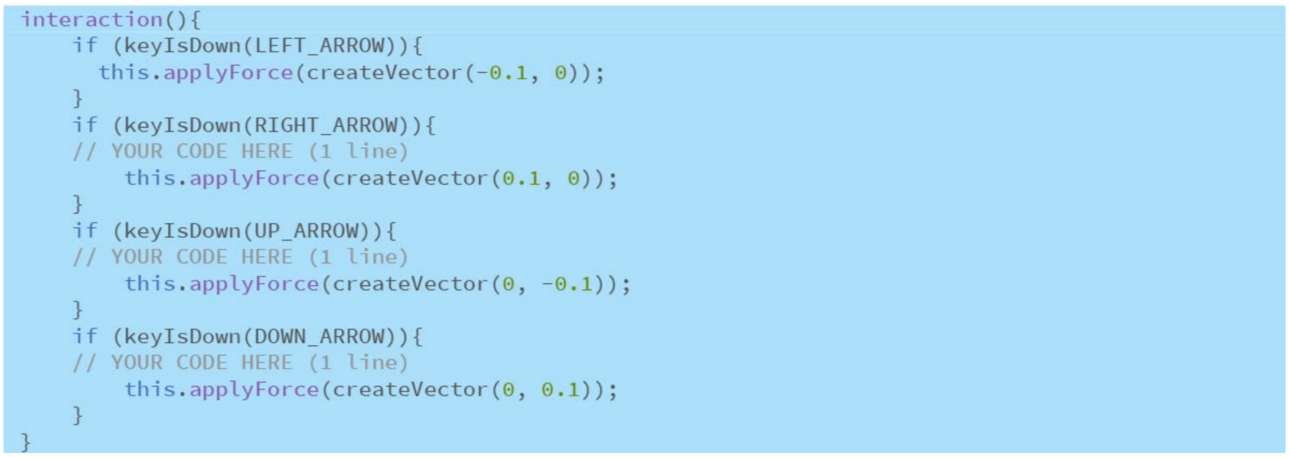
```
//////////////////////////////////////////////////
//draws earth and atmosphere
function drawEarth(){
  noStroke();
  //draw atmosphere
  fill(0,0,255, 50);
  ellipse(atmosphereLoc.x, atmosphereLoc.y, atmosphereSize.x,  atmosphereSize.y);
  //draw earth
  fill(100,255);
  ellipse(earthLoc.x, earthLoc.y, earthSize.x, earthSize.y);
}

```

An illustration of shared environment coupling might be this. A common piece of information is used in the function setup (atmosphereLoc). The atmosphereLoc variable is also used under the “drawEarth” function to showcase that the Earth and atmosphere are drawn in such a way that there is a gap between the Earth and atmosphere also known as Blue Ozone Layer .Hence, whenever the data is changed, both modules will be affected identically, eliminating any potential error-causing effects from using separate data.

3. Module Cohesion

Module Cohesion is the manner and degree to which the tasks performed by a single software module are related to one another within a single module. It is the measure of the strength association of the elements in a module.

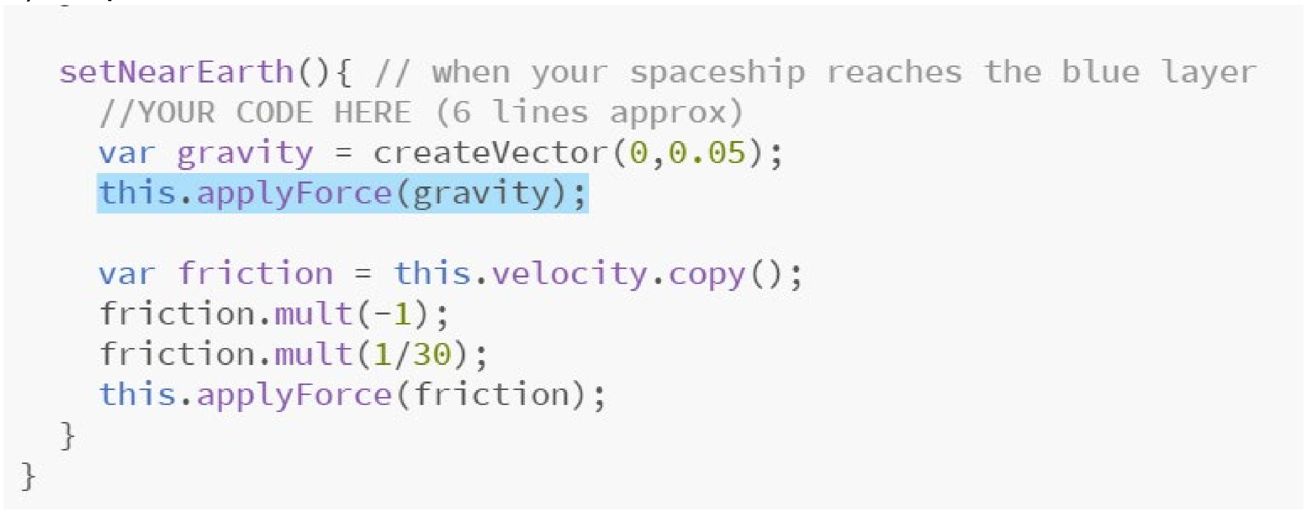
1) Temporal Cohesion:



```
interaction(){
    if (keyIsDown(LEFT_ARROW)){
      this.applyForce(createVector(-0.1, 0));
    }
    if (keyIsDown(RIGHT_ARROW)){
    // YOUR CODE HERE (1 line)
        this.applyForce(createVector(0.1, 0));
    }
    if (keyIsDown(UP_ARROW)){
    // YOUR CODE HERE (1 line)
        this.applyForce(createVector(0, -0.1));
    }
    if (keyIsDown(DOWN_ARROW)){
    // YOUR CODE HERE (1 line)
        this.applyForce(createVector(0, 0.1));
    }
}
```

A case of temporal cohesion is seen above. The movement of the spaceship is controlled by the function interaction () {}, which is made up of a series of if statements. For instance, the spaceship will move towards the right when the user presses the right arrow at a specific point in the game. This is a case of temporal cohesion because this module is only ever executed when the user pushes the right arrow.

2) Sequential Cohesion:



```
  setNearEarth(){ // when your spaceship reaches the blue layer
    //YOUR CODE HERE (6 lines approx)
    var gravity = createVector(0,0.05);
    this.applyForce(gravity);

    var friction = this.velocity.copy();
    friction.mult(-1);
    friction.mult(1/30);
    this.applyForce(friction);
  }
}
```

In the “class Spaceship” function, the “applyForce” function serves as an output (measure how fast the spaceship moves) – the spaceship moves even more quickly when it reaches the blue Ozone Layer and slower when it is outside of the layer. At the same time, it serves as an input in the “SetNearEarth” function where the “this.applyForce” uses gravity and function to adjust the speed when the spaceship comes near to the Earth.