Create PT - Written Response Template

Assessment Overview and Performance Task Directions for Students

Video Submit one video in .mp4, .wmv, .avi, or .mov format that demonstrates the running of at least one significant feature of your program. Your video must not exceed 1 minute in length and must not exceed 30MB in size

Prompt 2a. Provide a written response or audio narration in your video that:
- identifies the programming language;
- identifies the purpose of your program; and
- explains what the video illustrates.

(Must not exceed 150 words)

The programming languages that we have used include: HTML, CSS, javascript(ES6 and ES5 syntax), and some shell scripting. The purpose of the program is to provide a game to people where concepts are taken from Minecraft and the battle royale genre of games to create a two-player, lightweight browser-based game that is not blocked by MCPS.
The video illustrates the fundamental concepts that go into the game mechanics, such as vector calculation for determining the paths of bullets and javascript OOP where the walls and bullets and player are examples of instances of a predefined class. It also shows the server working to serve the HTML and javascript data with a node framework called express. Here, you can see the collision check implemented between the player and the wall, there is also collision between the player and the bullets, and player and items.

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and / or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. *(Must not exceed 200 words)*

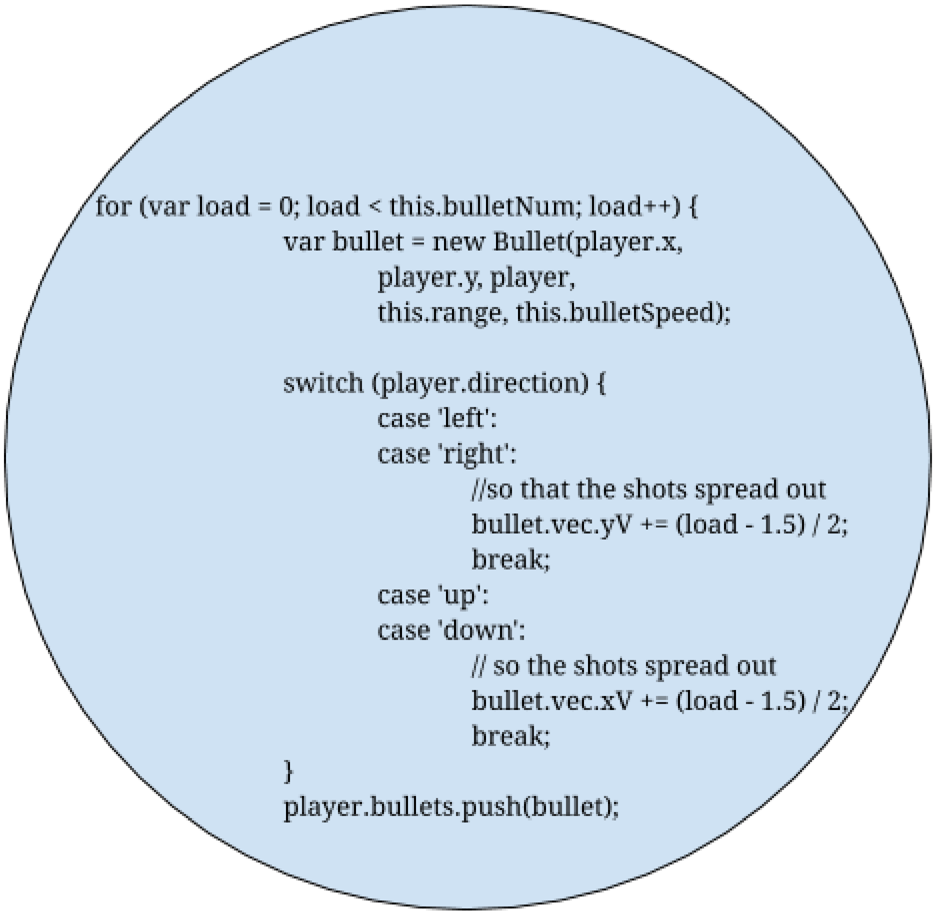
The difficulties that we have faced with using javascript is the lack of type checking and restrictions, which make it harder to debug. whilst at the same time, javascript doesn't have a good IDE built-in debugger, making it harder to find errors, Even though I was able to find a chrome debugger as one of the vscode extensions, it required a lot of setup and environment configuration. The way I made the situation better was by using typescript; a language built on top javascript and transcompiles to Javascript code. The mistakes were resolved by logging variables to the console, and one thing particular in javascript that helped was the fact that logging a class instance into the console gives an extendable object where we can see the attributes, whereas, in other programming languages, it just gives the pointer address to the object. The development process was individual, I made the framework and specific implementations of the program, whilst also debugging the code and writing comments to help other programmers understand it better.

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3**) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Must not exceed 200 words)*

Code Segment

Gun.js

```
const Shotgun = new Gun('shotgun', 100, 250, 3, 4, 17, 11);
Shotgun.shoot = function (player) {
        const rs = 30;
```



```
for (var load = 0; load < this.bulletNum; load++) {
            var bullet = new Bullet(player.x,
                  player.y, player,
                  this.range, this.bulletSpeed);

            switch (player.direction) {
                  case 'left':
                  case 'right':
                        //so that the shots spread out
                        bullet.vec.yV += (load - 1.5) / 2;
                        break;
                  case 'up':
                  case 'down':
                        // so the shots spread out
                        bullet.vec.xV += (load - 1.5) / 2;
                        break;
            }
            player.bullets.push(bullet);
```

```
            this.range, this.bulletSpeed);


      }
}
```

Written Response

The algorithm in the gun class is the switch statement and the cases, it checks for if the player is moving left/right or up/down which require changes to the x and y coordinate respectively, here, I use the literation through the number of bullets(load) to calculate the position of the bullets, the increase to the x velocity vector and y velocity vector gives the shotgun's bullets the ability to spread out. By subtracting 1.5 in: (load - 1.5) I was able to create an offset to the fact that the bullets are drawn slightly up&left because of problems with the HTML canvas and the way in which javascript draw functions sometimes starts in the corner of the shape instead of at the center. The functionality of the shotgun is later used in the shooting mechanics of the player, and they are combined to make the game more interesting and well balanced, making it achieve the intended purpose of entertaining the user. Since the game needs a decent short ranged weapon. I did this by constantly debugging the code, so I could focus on the mathematical and logical concepts instead of worrying about errors.

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a **rectangle** in **section 3**). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. *(Must not exceed 200 words)*

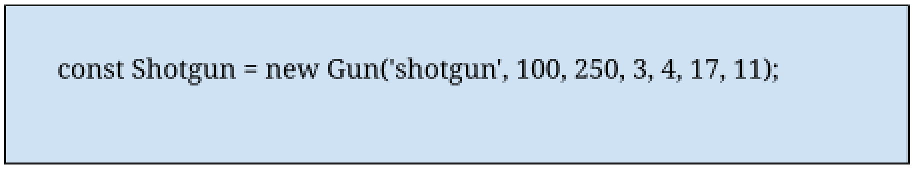
Code Segment

```
class Vector {
   constructor(xV, yV) {
      this.xV = xV;
      this.yV = yV;
      this.vec = [this.xV, this.yV];
      this.mg = Math.sqrt(this.xV * this.xV + this.yV * this.yV);
      this.unitVec = [this.xV / this.mg, this.yV / this.mg];
   }
}
class Bullet {
.....

                    this.vec;
                    if (this.direction === 'left') {
                              this.vec = new Vector(-this.moveMentSpeed, 0);
                    } else if (this.direction === 'right') {
                              this.vec = new Vector(this.moveMentSpeed, 0);
                    } else if (this.direction === 'up') {
                              this.vec = new Vector(0, -this.moveMentSpeed);
                    } else if (this.direction === 'down') {
                              this.vec = new Vector(0, this.moveMentSpeed);
```

```
                }
        }
        setPos(player) {
                this.x += this.vec.xV;
                this.y += this.vec.yV;
                //if it has reachs the end
                const diff = this.maxDistance - distance(this.x, this.y, this.initX, this.initY);
                if (diff <= 5) {
                        player.bullets.splice(player.bullets.indexOf(this), 1);
                }
        }
.....
}

const Pistol = new Gun('pistol', 20, 400, 3, 1, 13, 8);
Pistol.shoot = function (player) {
        player.bullets.push(new Bullet(player.x, player.y,
                player, this.range, this.bulletSpeed));
}
```



```
const Shotgun = new Gun('shotgun', 100, 250, 3, 4, 17, 11);
```

```
Shotgun.shoot = function (player) {
        const rs = 30;
        for (var load = 0; load < this.bulletNum; load++) {
                var bullet = new Bullet(player.x,
                        player.y, player,
                        this.range, this.bulletSpeed);

                switch (player.direction) {
                        case 'left':
                        case 'right':
                                //so that the shots spread out
                                bullet.vec.yV += (load - 1.5) / 2;
                                break;
                        case 'up':
                        case 'down':
                                // so the shots spread out
                                bullet.vec.xV += (load - 1.5) / 2;
                                break;
                }
                player.bullets.push(bullet);
        }
}
```

```
Class Player {
…

          shoot() {
                    if (this.canShoot) {
                         this.gun.shoot(this);
                              this.canShoot = false;
                    }
          }
…
}
```

Written Response

The class Vector helped me manage the complexities of the program since the path of the bullets would be hard to calculate otherwise and within the constructor of the vector class, all the attributes, including the magnitude, unit vector of the input directions are calculated so that when the positions of the bullets are getting calculated in the setPos function in the bullet class, it only has to calculate the <originalX, originalY> + t<vectorX, vectorY>. Without having to know what is happening within the calculation of the vectors. And the bullet instances are later used in the gun instances in the shoot function while I, the programmer don't have to remember the details of how the paths of the bullets are being calculated. The guns are then later a part of the attributes of the players' instances, and they can simply just call the shoot function on a particular gun without having to think twice about the vector calculation that goes on inside of it. The parameters in the initializer of the gun class helps the programmer to manage the different types of guns or create new ones simply by putting in different arguments.

Export or save this document as a PDF and turn in to the AP Digital Portfolio along with your **Video** and **Program Code** (separate files).