Create Task Questions- Snake Game

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)***

My code is written in javascript, using the p5js libraries. I wrote the code in Atom. The purpose of my program is to provide entertainment through a game. My video starts with the main screen of my program. It then shows a user clicking from the main screen on an instructions button that takes them to a new splash screen with instructions. The user then presses a back to menu button that takes them back to the menu. The player chooses first to go to the sea themed snake, they then play the game by eating food and growing longer. The player moves the snake using the arrow keys. When the player dies, the screen changes and the player has the option to go back to the home screen or to replay that level.

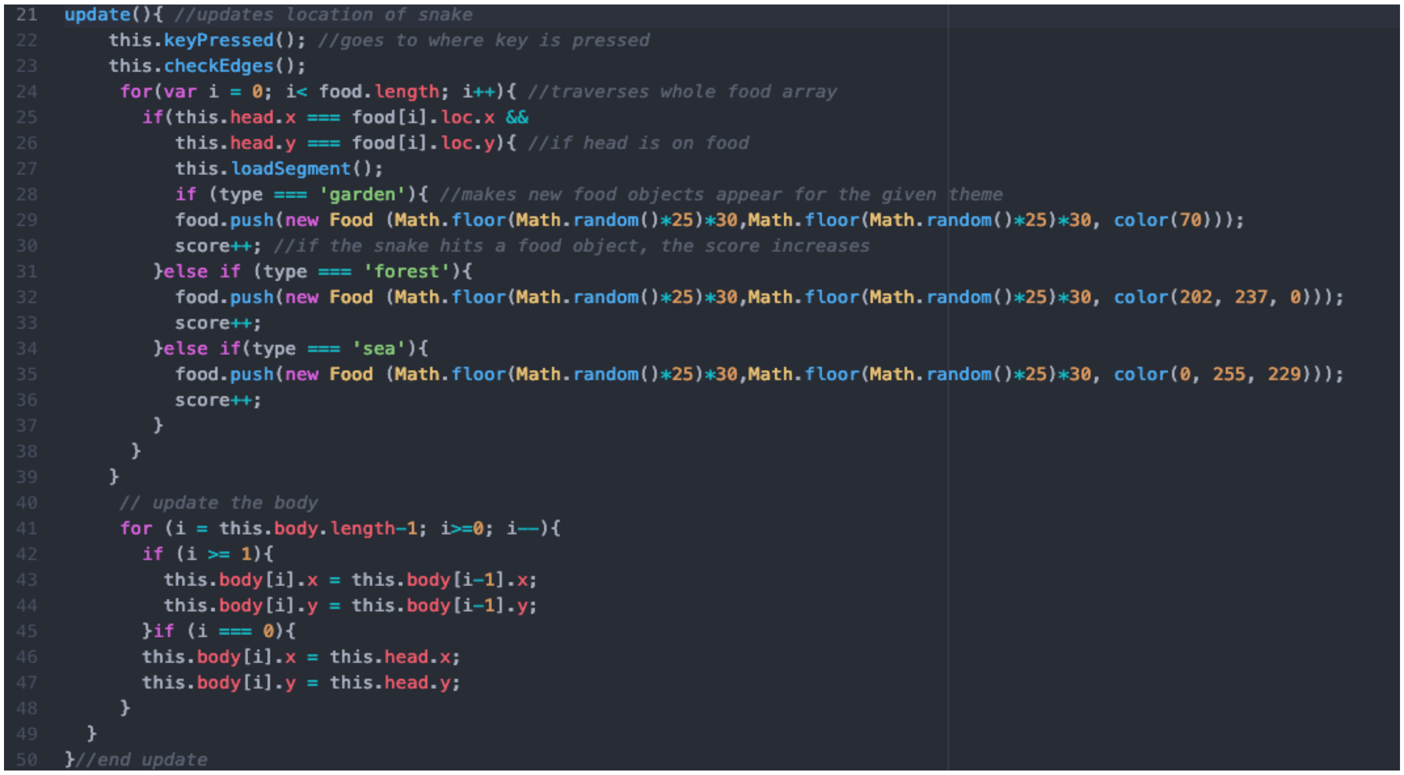
Word Count: 134

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)

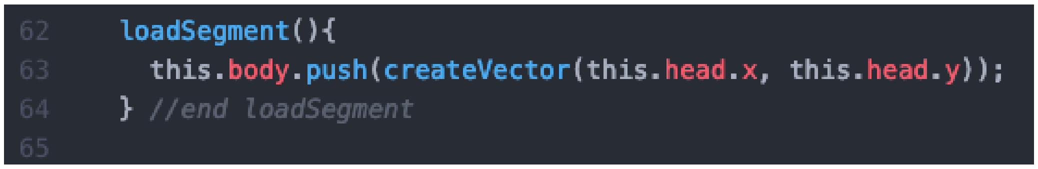
I started with my program from earlier this year that had a start screen, a snake that moved around, and food objects that were eaten. For this project, I added themes and the final splash screen. I started by creating a new gameState for picking the type of snake the user desired. This required me to change the gameStates for all of the buttons since I tried to make the gameStates follow the order of the program. One difficulty was that the buttons would take me to different gameStates simultaneously. For example, the back to the main menu button would flash the snake and food on top of the main menu screen. In order to fix this, I used the debugger in chrome. I put a stopping point where the instructions button was clicked and then stepped through each of the calls that it went through until I found where it went to a gameState that was not desired. Another difficulty I encountered was making sure the text from the words was inside the button. I solved this issue through trial and error and running the program at the desired screen until the text was where I wanted it.

Word Count: 199

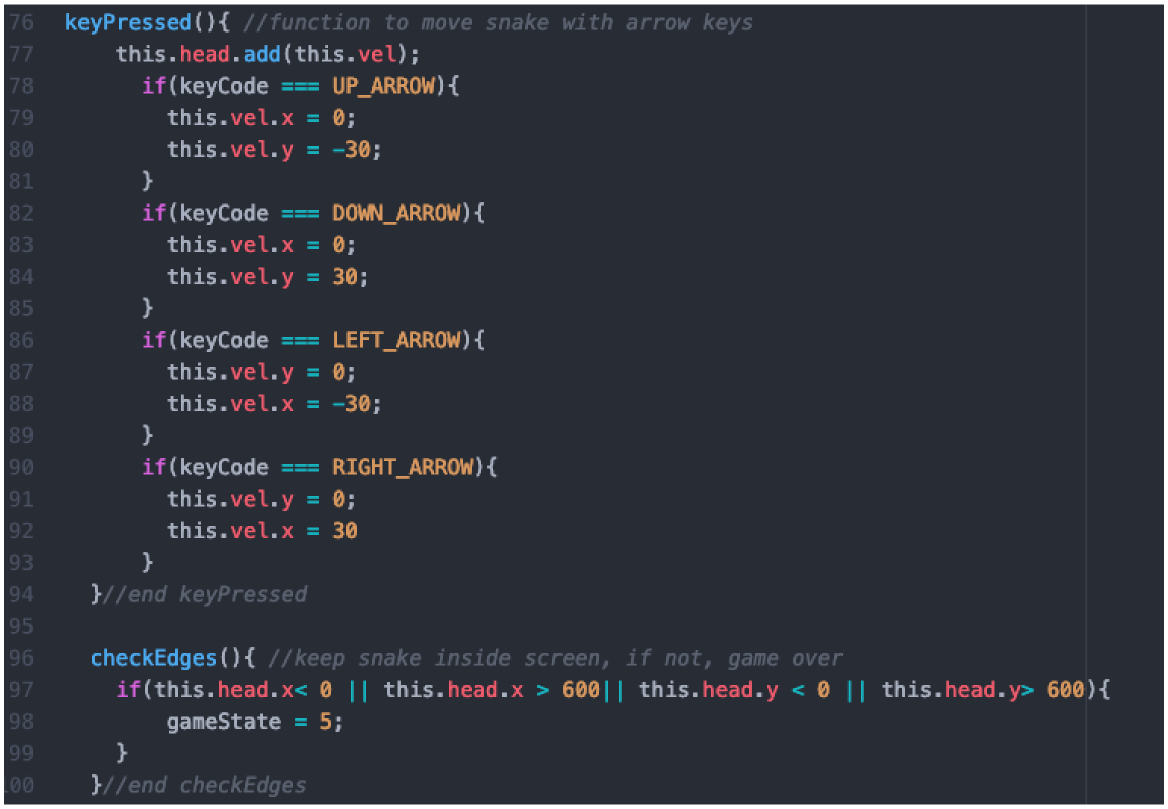
2c. Capture and paste a program code segment that implements an algorithm (marked with an oval in section 3 below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, 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)



```
update(){ //updates location of snake
    this.keyPressed(); //goes to where key is pressed
    this.checkEdges();
     for(var i = 0; i< food.length; i++){ //traverses whole food array
       if(this.head.x === food[i].loc.x &&
          this.head.y === food[i].loc.y){ //if head is on food
          this.loadSegment();
          if (type === 'garden'){ //makes new food objects appear for the given theme
          food.push(new Food (Math.floor(Math.random()*25)*30,Math.floor(Math.random()*25)*30, color(70)));
          score++; //if the snake hits a food object, the score increases
        }else if (type === 'forest'){
          food.push(new Food (Math.floor(Math.random()*25)*30,Math.floor(Math.random()*25)*30, color(202, 237, 0)));
          score++;
        }else if(type === 'sea'){
          food.push(new Food (Math.floor(Math.random()*25)*30,Math.floor(Math.random()*25)*30, color(0, 255, 229)));
          score++;
        }
      }
    }
     // update the body
     for (i = this.body.length-1; i>=0; i--){
       if (i >= 1){
         this.body[i].x = this.body[i-1].x;
         this.body[i].y = this.body[i-1].y;
       }if (i === 0){
       this.body[i].x = this.head.x;
       this.body[i].y = this.head.y;
     }
  }
}//end update
```



```
  loadSegment(){
    this.body.push(createVector(this.head.x, this.head.y));
  } //end loadSegment
```



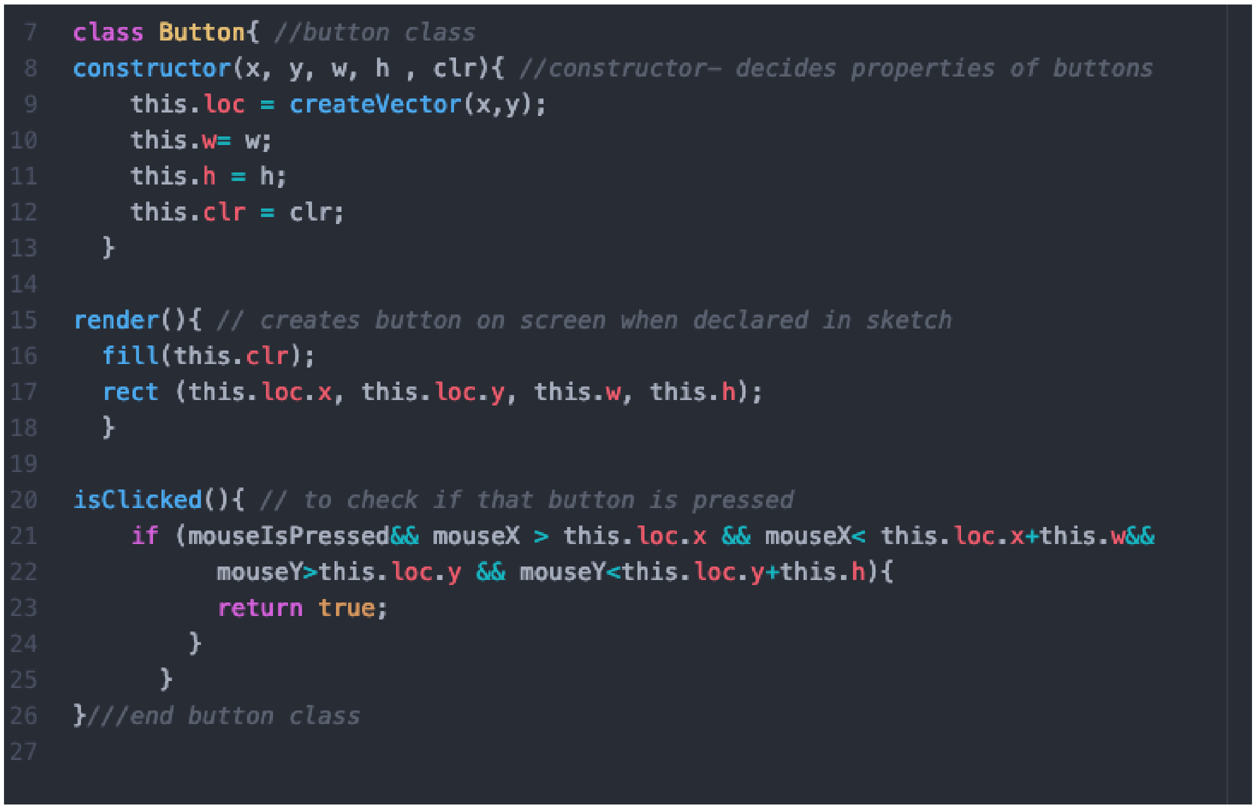
```
keyPressed(){ //function to move snake with arrow keys
    this.head.add(this.vel);
      if(keyCode === UP_ARROW){
        this.vel.x = 0;
        this.vel.y = -30;
      }
      if(keyCode === DOWN_ARROW){
        this.vel.x = 0;
        this.vel.y = 30;
      }
      if(keyCode === LEFT_ARROW){
        this.vel.y = 0;
        this.vel.x = -30;
      }
      if(keyCode === RIGHT_ARROW){
        this.vel.y = 0;
        this.vel.x = 30
      }
  }//end keyPressed

  checkEdges(){ //keep snake inside screen, if not, game over
    if(this.head.x< 0 || this.head.x > 600|| this.head.y < 0 || this.head.y> 600){
        gameState = 5;
    }
  }//end checkEdges
```

The update algorithm creates the snake on the screen. It utilizes the p5js functions math.Random and math.Floor to put the snake at a random place somewhere on the screen. It also works with the keyPressed function, checkEdges function, and the loadSegment function to see if the snake should be moved and where it should be moved. The loadSegment function checks to see whether the snake ate a piece of food, and if it did adds another segment. The checkEdges function tests to see whether the snake has left the canvas using a conditional to see if the snake's position is outside of any of the edges. The keyPressed function tests if the user has pressed the arrows, to see if the snake should move. If the arrows are pushed, I used the p5js function push to move the snake in the direction of the arrow. All of these functions are called in the update algorithm which is called in the sketch class to create a snake every time the draw function is called, which is 30 frames a second.

Word Count: 181

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 below). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. (Must not exceed 200 words)



```
class Button{ //button class
constructor(x, y, w, h , clr){ //constructor- decides properties of buttons
    this.loc = createVector(x,y);
    this.w= w;
    this.h = h;
    this.clr = clr;
  }

render(){ // creates button on screen when declared in sketch
  fill(this.clr);
  rect (this.loc.x, this.loc.y, this.w, this.h);
  }

isClicked(){ // to check if that button is pressed
    if (mouseIsPressed&& mouseX > this.loc.x && mouseX< this.loc.x+this.w&&
          mouseY>this.loc.y && mouseY<this.loc.y+this.h){
          return true;
        }
      }
}///end button class
```

This abstraction is the button class in my program. The button class enabled me to write the template for a button in order to write just one line of code to create a new button instead of writing these 26 lines over and over. If I was just creating one button, I could write 26 lines of code to put a single button on the screen, but since I had seven buttons in my program, I chose to create a button class and then write a single line of code for each button. The abstraction allowed me to give each button a location, height, width, and color in only one line of code. Additionally,

I was able to create an asClicked algorithm which allowed me to see if any of the button were clicked depending on their location and height and width.
Word Count: 142