Create – Applications From Ideas
Written Response Submission Template

Please see Assessment Overview and Performance Task Directions for Student for the task directions and recommended word counts.

Program Purpose and Development

2a)

The purpose of this snake game is to entertain the user and was made in Javascript. The program works by first showing the instructions on the screen. The instructions are first shown to the player on startup. When the user presses the spacebar, the game begins, and the player's snake will start in a random position going in a random direction. The player uses the WASD or Arrow keys to move the snake in a direction. The objective of the game is to eat as much fruit as you can to increase your score before crashing into yourself or crashing into the wall. The player can also activate the snake AI by typing 'bot' during the instructions screen and then pressing the spacebar.

2b)

Project was done independently. First, I thought of an idea of how to make the snake move and where the snake can move to. Then, I coded it and added the game mechanic of having the fruit spawn in a random location and increase the snake's length once the snake ate it. Next, I made a gameover function to display the gameover screen when the snake crashes and an option to play again. Next, I added an instructions screen. Lastly, I created an AI for the snake for fun and to test the game.

A difficulty I had was figuring out how to have the trailing blocks of the snake follow the path of the leading block. This was resolved by creating an array that would contain the coordinates of where the snake has been and creating a for loop to set the position for each block.

Another difficulty I had was that the fruit would sometimes spawn on top of the snake after it was eaten. This was fixed by having a while loop check if the coordinates of the fruit were already occupied by a snake block; if it was, then it'd generate another coordinate pair.

2c)

```
var highScore = 0;

function snakeObj() {
    if (ai.access)
        aiBot();

    if (snake.x > 300 || snake.x < 10 || snake.y > 430 || snake.y < 20)
        deathAction();

    if (!snake.didCrash) {

        if (snake.moves.length == snake.IDandLength + 1)
            snake.moves.pop();
        snake.moves.splice(0, 0, snake.x + " " + snake.y);

        if (snakeCol(fruit.x, fruit.y, fruit.w, fruit.h))
            addNewBlock();

        setPosition("snake", snake.x, snake.y, snake.w, snake.h);

        for (var a = 1; a < snake.IDandLength; a++) {
            var coords = snake.moves[a].split(" ");

            var x = parseInt(coords[0]);
            var y = parseInt(coords[1]);

            setPosition("" + a, x, y, 10, 10);

            if (snakeCol(x, y, snake.w, snake.h))
                deathAction();
        }
    }
    snake.x += snake.velX;
    snake.y += snake.velY;
}
```

```
function addNewBlock() {
    var coords = snake.moves[snake.IDandLength].split(" ");
    var x = parseInt(coords[0]);
    var y = parseInt(coords[1]);

    image("" + snake.IDandLength, "greenBox.png");
    setPosition("" + snake.IDandLength, x, y, 10, 10);

    snake.IDandLength++;

    fruit.x = points.x[randomNumber(0, points.x.length - 1)];
    fruit.y = points.y[randomNumber(0, points.y.length - 1)];

    while (xyIsSame(fruit.x, fruit.y)) {
        fruit.x = points.x[randomNumber(0, points.x.length - 1)];
        fruit.y = points.y[randomNumber(0, points.y.length - 1)];
    }

    setPosition("fruit", fruit.x, fruit.y, fruit.w, fruit.h);

    setText("scorelabel", "Score: " + snake.IDandLength);
}
```

```
function deathAction() {
    stopTimedLoop();
    snake.didCrash = true;

    if (doGameOver == 0) {
        doGameOver++;
        gameOver();
    }
}
```

snakeObj is called every fifty milliseconds to move the snake and check for collision. First, if **access** is true, **aiBot** will give the snake a direction to go in. Next, if the snake's coordinates are outside the play area, it'll call **deathAction**. If the snake didn't crash, it'll ensure the array, **moves**, contains **IDandLength**+1 coordinates by removing the last coordinate and adding the new coordinate to the front of **moves** since the next block of the snake will be at the last coordinate in **moves**. **addNewBlock** is called if the snake collides with the fruit. Next, it repositions the snake with its current coordinate and its blocks by looping through **moves** to set its position with its coordinate. Also, if the snake collided with that block, it'll call **deathAction**. Afterwards, the snake's x and y coordinate change by adding its corresponding **vel** variable.

addNewBlock's algorithm works by using the last coordinate in **moves** as its spawning position, and increasing **IDandLength** by one so **moves** can hold another coordinate for the next time the algorithm is called. Lastly, it repositions the fruit to a random, unique coordinate which is ensured by the loop checking if **moves** has the fruit's coordinate.

2d)

```
74
75  function deathAction() {
76      stopTimedLoop();
77      snake.didCrash = true;
78
79      if (doGameOver == 0) {
80          doGameOver++;
81          gameOver();
82      }
83  }
```

This manages the complexity of the program by allowing other functions to call **deathAction** in situations where the snake crashed which should cause the ending of the game. **deathAction** stops the timedloop to stop calling the function **snakeObj** which will cause the snake to stop moving. After stopping the timedloop, it sets the **didCrash** to true to prevent any code in **snakeObj** from executing unexpectedly. Next, since the variable **doGameOver** is defaulted to zero every time the game resets, it will always run the first time until the player plays again. **doGameOver** is incremented to prevent any unexpected, repeated executions to the function **gameOver** which could cause visual errors and double the speed of the snake. **gameOver** is lastly called and shows the UI elements for the ending screen such as their score that game and the play again button.