**SNAKE GAME**

**Jared:**

**First Slide:**

Hi, today, we are going to present our version of the Snake Coding Challenge. We are going to show you how we made the game, as well as discuss about the game and some related computer science concepts.

**Background:**

To start, we’re going to talk a bit about the history of Snake. Most think of Snake as a game itself. However, Snake is actually a common name used for this game concept. The Snake we all know today really originated from a 1976 arcade game called Blockade, which was widely adapted and cloned by other companies for numerous platforms. For example, the first known PC version was Worm, released in 1978 for the TRS-80. Snake eventually grew massively in popularity after it was preloaded on Nokia mobile phones, and it is being constantly adapted even today.

**Snake GIF:**

This is our version of Snake that we coded in Processing. Today, we will be walking you through the steps to recreate the game.

**Snippet 1 - Drawing a Grid:**

**Open Processing:**

Now, to start coding the game, this, of course, begins with opening Processing.

**Snippet 1:**

We’re going to begin by drawing a graph on the screen. Although this isn’t necessary, it really helps with identifying where the coordinates are while we code, and we can always remove it afterwards.

**Grid:**

So, any guesses on how we are going to accomplish this?… Right (or wrong), we are going to be using a for loop. Now, we need to define the spacing of each square, so with a canvas size of 600 by 600 pixels, we will draw a 30 by 30 grid, with the internal squares being 20 pixels wide.

**Coding the Grid:**

So, we start off here with our standard setup and draw functions. We’re going to create a new integer called widthOfSquare, and we’re going to set it to 20. In setup(), our canvas size is 600 by 600 pixels. In draw(), we want to continuously draw a white background, and we want to run a function that we will create next called drawGrid().

**drawGrid:**

In drawGrid(), we have a for loop that runs 30 times. Notice that this is the same number as how the size of our grid we said earlier. We’re going to draw two lines each time the loop runs, one that goes from the top to the bottom, and another one that goes from the left to the right. Each time the loop runs, the coordinate where the line draws increases each time. This gives us our final product of a grid.

**Snippet 1 link:**

Because our code will eventually become long, we will be redirecting you to a website that contains the code for our snippets. This is the link to draw the graph.

**Not Jared:**

**Snippet 2 - Moving the Snake:**

**Class Introduction:**

Next, we are going to make the snake and allow it to move. To do this, we need to create something called a class.

A class is essentially a separate file that creates an object and is used in your main file. They act as a blueprint for similar objects that can be altered. We can use cars as an example of a class. Although every car performs similar functions, they all have different variables, for example, their size or their maximum speed.

**Class example 1:**

I’ve created a simple program to help explain and demonstrate how a class functions. A class acts as a blueprint from which individual objects, in this case, our cars, are created. Inside a class, we can create attributes to differentiate our cars, and we can also create functions that run the cars’ tasks. In your main file, you can define the attributes for each car and run these functions. Using the car example, we can, set the maximum speed for one car and for another car.

**Class example 2:**

This is the example program I made, with the code that sets the car attributes above. On the right, I’ve created a basic grid that shows you the coordinates of every 100 pixels both vertically and horizontally. If you compare the image with the code, you can see that attributes I set for the cars correspond with what is being drawn. Notice how they are similar, in which both “cars” are rectangles and have the same fill colour, but they have different coordinates, sizes and maximum speeds set. This is essentially what a class is: a file that creates similar objects that can be altered. If you wanted to see this example in depth, I’ve uploaded this onto our repository.

**Class example 3:**

Now, to start making the snake move, we are going to create a class for it. To create a class in Processing, click the downwards-facing arrow right next to the sketch name near the top, then click New Tab. We’re going to call it SnakeCharacter.

**Grid Explanation:**

Now, we have the problem of drawing the snake. We want to be able to draw the snake within the grid’s squares, as well as when we move and grow the snake. Because the size of our grid is 20 pixels wide and tall, we therefore want the point to always be a multiple of 20. So, for example, we want the snake to spawn right in the middle of our 30 by 30 grid. This means that the center of the grid is at (15, 15). However, if we simply code the coordinates of (15, 15), you will notice that the snake spawns at the upper left corner as the coordinates are exactly 15 pixels from this entire 600-pixel canvas. So, by multiplying the coordinates by 20, it ensures that the snake will always be drawn within the squares.

**Drawing the Snake:**

Now, inside of the class, we’re going to first define the coordinates of the snake. Notice that we’re multiplying the coordinates by 20 here. We’re going to make a function called create. In here, we will draw the snake, which is a simple rectangle. Because it’s a snake, we’re also going to add a fill of green, but this can of course be any colour you desire.

**Moving the Snake:**

Perfect. Now that we have the snake there, we want it to be moving across the screen. We will start by setting the snake’s direction. We will need two new integers that we will xSpeed and ySpeed. This will be under a new method called direction. In this method, we’ll have to pass in 2 variables, an x integer and a y integer. Then, we’ll have to set them equal to each other. By default, we want the square to move to the right, so we’ll set xSpeed to be 1 by default. If you’d like, you can change this to make it go down, up, or to the right.

Now, this won’t actually do anything yet because we haven’t told the snake rectangle to move, rather, we just told it which way to move. So, to solve this, we will make a new method called move. This takes the xSpeed we just set and multiplies it by 20 so that the snake always moves to the next square of the grid.

Next, we will call the method from our main file, and if we run the program, you should see that the snake moves.

**keyPressed:**

Now, we want to be able to control the snake with our arrows. Anyone have any ideas on how to do that? Yeah, make a keyPressed function. Inside there, you want to change the xSpeed and ySpeed variables that we set, because those define the snake’s directions. So, for example, if you press the up arrow, you want to set the new direction to 0, -1. This subtracts 1 from the y-value, making the snake move up. The rest of the code look something like this.

Run the program, and you can now make the snake move around!

**Fruit:**

Finally, we’re going to make a new class called Fruit. In this class, we will have a method that draws a fruit, which is a black square, at a random location.

Once again, we will call the method from our main file, and the fruit should appear when you run the program.

**Snippet 2 Link:**

Here is the link to the code that we have just discussed.

**Snippet 3 - Growing the Snake:**

Now that we have that out of the way, we are going to move on and focus on making the snake grow.

**ArrayList:**

In order to make the snake grow, we need to understand a new concept. We need to know what an ArrayList is. An ArrayList is sort of like an array, as it is in the name. The second part of the word, list, is probably new to most of you. An ArrayList is like an array, except you can run different commands on it, such as .get, .size, .remove and .add.

For our game, we have to make a new class called Point. We will give it an x value, a y value and 2 constructors. One constructor will be empty, and the other one will take in an x and y value. Note that when adding to an ArrayList, you don’t have to specify the index.

**Create Function:**

Next, we’re going to have to modify our create function that we made before. We will be adding a special for loop, that just loops over the whole array list.

**Initializing a Point instance:**

Now, we have to initialize the snake’s point. To do that, we have to first declare the ArrayList. Then, we write a constructor, and all that is inside of it, is a new starting point. So, whenever you create the snake, it’ll add a point at (15, 15), which is really just the centre of the screen.

**Move:**

Now, for the hardest part of the code, the move functionnnnn! :/ To start, just remove everything you have inside the move function as of right now. Boom, clear slate.

Now, we are going to find the head of the snake. Does anyone know what function we will use to call it? Anyone? Remember there are get, remove, add, size.

So, we are going to get the head. Once you have the head, set it to a new xHead and yhead. Then, change the speed based off of what direction you are going in. Now, whenever you move, you have to make it so that the tail of the snake goes to the top of the snake and becomes the new head.

Next, we are going to add a remove function. If we don’t have this, then the snake will keep on going on forever. So, if there is a collision, which is a new function that we will get into in a second. So, once we got that, we are going to have to add to the arrayList, then, return whether the collision was there for later purposes.

**Collision with Fruit?**

So, we want a collision, when? Anyone? What is the purpose of the game? To grow right? So, when we touch a fruit, we want a boolean called isFruitTouched to be set to true. So when the snakeHead point is equal to that of the fruit, it will return true. Otherwise, it will return false. Boom

**Instance of Fruit:**

Next, we are going to create a declare an instance for fruit so that it can continue to regenerate. We are also going to create a method called setFruit. What this does is that it simply sets the new fruit as a temporary object in which we assign a random coordinate. We set this as a temporary object because we don’t want the fruit to regenerate at the same spot.

**Calling the Methods:**

Finally, we are going to call these methods in our main .pde file. First, we generate a fruit when the program starts. Then, we set the boolean isCollision to snake.move(). We then have an if statement that regenerates the fruit if the snake touches an existing fruit when it moves.

**Snippet 4 - Completing the Game:**

Finally, with the essentials down, we’re going to add a few more functions to truly complete the game.

**Moving Restrictions:**

In Snake, you can’t move in the opposite direction that you’re facing as you would otherwise eat yourself and lose. So, to program this, we first have to determine which way the user is facing. We create four booleans and set them each time the user changes direction. Then, with if statements, you can prevent the user from moving in the opposite direction.

**Score:**

To display the score, we simply make a text statement that displays the size of the point. Because this essential outputs the length of the snake, we subtract 1 because we want to output how many fruit was eaten instead.

**Collision with Border?:**

Next, you want to make a boolean that checks whether the border of the canvas was touched, that is, 0 or 30.

**Snake Touch**

You also want to make a boolean that checks whether the snake head touches any point of the snake.

**Is the Game Over?**

Now that we detect every way to lose, we want to be able to check the state of the game from within our main .pde file, as all the previous booleans were within our SnakeCharacter class. In this code, we’re creating a boolean and returning whether any object was touched or not.

**If the Game Isn’t Over…:**

We are going to add a Game Over function that displays if you hit an object. However, we only want it to display if the game is actually over. With the boolean we just made, we can use it to check whether the user lost or not, and then run the gameOver function accordingly.

**Game Over**

This function runs when the isGameOver boolean is set to true. We simply tell the user that they lost, and if they want to restart the game, they can press ENTER or RETURN. We experienced a bug in which if the user loses twice, the game would automatically reset when the gameOver function ran. We solved this by setting the keyCode to something else so that it wouldn’t continuously press Enter in the background.

**Reset**

Finally, when the user decides to play again after a Game Over, we want to reset the state of the game. This includes the length of the snake, and the direction and coordinate that they start in. We simple just reset the variables and instances in this function.

**Jared:**

**Connecting with Computer Science Research**

Computer science research can be connected to Snake through machine learning and artificial intelligence. An example of this is in this GIF, in which a computer learned how to and played a perfect game of Snake. It can also be used to add new features to the game. For example, the fruit could move away from the snake based off of its moving algorithms. Seeing as this is a more basic usage for AI, as well as the fact that it is a game, it is more likely that students would find this more interesting compared to, for example, a technology company.

**Connecting with Post Secondary Opportunities**

Snake could spur on more learning of computer science, especially for students or children who are interested in developing a game of their own. As well, with the flexibility and creative nature of code, anyone can modify the functions, which could lead to more learning.

In this presentation, we learned about two new computer science concepts, including Classes and ArrayLists, which could be useful for more advanced programming in the future, such as in the ICS4U course or in the workfield.

**Creative Ways to Code:**

Before we conclude, we want to share with you all creative ways to change our code. We have three examples which we will show you.

In Jared’s example, he changed the colour of his snake, increased the canvas size, and displayed two fruits at once.

In Michael’s example, he changed the colour scheme, made the fruit into a circle, and reversed the controls.

Finally, in Vinay’s example, he changed the colour of his snake.

**Twitter:**

We would also encourage you all to share your Coding Challenges with us in our GitHub repo, and maybe even tweet it to Shiffman, who might retweet you.

Thank you all for listening.