Swat Ninja created by Yael Borger

Backstory

The main concept was originally a recreation of Fruit Ninja that I would be able to play, for entirely selfish reasons. The "problem" I intended to solve was that Fruit Ninja became a pay-to-play kind of game, but I was nostalgic and decided to make my own. Unfortunately, after spending plenty of time designing the main functionality, my friend noted that the pathing function made the fruit object seem to fly around the screen like a bug, and I actually quite liked the premise of swatting a bug instead of slicing fruit, in order to add a "Swat" twist into the game.

Principles

In regards to the principles of the class and what course content was applicable to the idea, my main focus was investigating how feature detection could be connected to a kind of augmented reality.

Throughout the course, there was plenty of content regarding how the camera captured different details and how different feature detection algorithms work. Since there had been so much course coverage on how feature detection really works, I felt confident using a library that detected hands (ml5.handpose) and used the knowledge from class to determine how that actually worked and what would be best practice for using it. Since it is a *hand* detection, it did the detection of each part of the hand, meaning when a hand is turned sideways in front of the camera, it would have a bit more difficulty capturing all of the points to determine it is a hand. Additionally, the feature detection we learned in class often could not handle quick movements and would not register those properly. As a result, the "swatting" seemed to make much more sense than "slicing" to have consistency in the results of the game.

For augmented reality, the "fruit" object is a two-dimensional object with its own parts of the code that interact with the user. The biggest challenge in this regard was figuring out how to make the augmented feature actually exist in the same field as the user when they have completely different builds and points of reference on the screen.

Design

As an overview, the game is meant to start by the push of a button. I used the spacebar, but the alternative gameplay is using the 'p' key to not have the distraction particles floating around with the main object. Upon starting up the script, the user is immediately met with a live capture of themselves on the screen. Then, upon pressing the spacebar, the user is given the opportunity to play the game. In the game, the user's hand is lit up with the green dots of identification, and a two-dimensional drawing of a randomly chosen bug from an array of them that I had drawn. The bug will fly around the screen in random directions, smoothly bouncing around the screen, with a

particle system closely following it as a distraction for the user (it felt too easy). The user can "swat" the bug by waving their hand (preferably palm out as if swatting the fly on the screen) and overlapping with the bug. The bug will then be randomly changed once more and pop into a new place on the screen. This is an endless sequence.

Implementation (how)

For clarity, I will refer to things by their function or file names and then explain how they all are put together in the "draw" function at the end of this section of the write-up. Since I am definitely over the word count, I wanted to make this as easy to read or understand as possible so it is divided up neatly and every call is explained as much as I thought necessary.

Additional Files: Pathing.js and Fruit.js

Pathing.js

- create an array of points and add points in vector format to the array

Fruit.js

- **constructor** - predetermined values for acceleration (0, no speeding up), speed, radius, maximum speed, and maximum force
    - set the position of the fruit in a vector form
- **follow (path)** - predict 25 points between the current position of the fruit and where the path indicates going, then approach that point
- **getNormalPoint (point, a, b)** - normalize the point based on where the next point would be to create a smooth path rather than one with many sharp turns
- **seek (target)** - go to target point by calculating the distance needed between the two points and moving in intervals of the speed
- **update()** - update all of the constructor values based on the movement of the fruit
- **getPos()** - return the x and y coordinates, and the direction of the fruit (not utilized)

Sketch.js:

preload()

Preload the calibration grid, as there were moments where the camera was unable to pick up the hand shape and it was quickly solved with the calibration grid. There is also the preload of the font that has been commented out.

As a note: I wrote most of my code prior to changing the theme and by the time I changed the theme I could not go back and change all my variable names without a significant amount of time spent on that. For this explanation I will be referring to the bug image that floats across the screen as "fruit."

setup()

Creation of:

- the canvas using WEBGL,
- the capture for the camera,
    - the dimensions of the capture's size,
- creating the "handpose" object to read the handpose library,
    - turning on the ability to read the hand based on predictions of where the hand's details would be in the camera's adjusted size
- the particle system ("slice_effect"),
- the Fruit ("fruit"),
    - the image of the fruit, the size of the fruit, the x and y coordinates of the fruit,
- fruitpath() method creates a Path

modelReady()

- quite literally just saying the program is ready and running

initCV()

- setting up opencv to work, not often necessary in this program but very useful for calibration when there were issues of the hand not appearing in detection

drawPoints(arrPredict)

- draw the hand based on an array of points collected at "landmarks" of the hand determined by the feature detection of the ml5.handpose library
- draw an ellipse on the landmark spots

handCheck(arrPredict)

- following the idea of drawPoints(arrPredict), round the keypoints of the hand and determine if that is within about 10 pixels of the fruit's x and y coordinates (global variables that get adjusted as the fruit moves)
- return TRUE if hand points are within 10 pixels of the x and y of the fruit
- I am unsure why but everything works perfectly fine without a "return false" case

addScore(points)

- the scoring system is fully implemented, but not up to my standards and were changed to just be a console print statement

fruitReset()

- delete all data in the fruit object
- reset all of the global fruit variables to new values
- create a new Fruit object
- randomly assign which image would be displayed from an array that holds the strings of each image file

- global pts variable set to the respective point quantity according to the fruit chosen
- call fruitpath()

fruitpath()
- two attempts made: one is commented out but it was utilizing the p5.js curve method
- create a new Pathing object
- add 12 points to the path

Floaties and System are the particle system and involve many methods that were covered in a previous lesson. It is also very similar to how the Fruit object works! I also realize I am very over the word limit and want to avoid extra unnecessary explanation for you.

draw()
Note: this actually comes up right after initCV() in the code itself!
- translate (-width/2, -height/2) - immediately to draw the capture in the top left of the screen because WEBGL resets where (0,0) is located to the middle of the space
- background of the greater screen to black
- COMMENTED OUT: would display text of the score, it was not working but the issue was resolved and it is now commented out because it was not up to par with my own definition of satisfactory for this project
- image - display the capture
- if (key == 'c') - calibration using the grid image preloaded
- else if (key == 't') - mostly used for a testing of basic parts
    - displays the image capture (in hindsight the first two lines were not necessary in this section)
    - places the fruit image on the screen
    - drawPoints()
    - handCheck()
- else if (key == ' ') - upon pressing the spacebar…
    - display the image capture
    - fruit object does its follow(array of points) method
    - fruit object updates to move along the path
    - grab the current fruit position
    - push! this is the current task:
        - translate to focus on fruit and its movement relevant to its current position
        - grab the coordinates of the fruit
        - add particles for the particle system and run it to follow the fruit
        - pop! okay all translating from above is not relevant to the stuff below!
    - drawPoints(predictions)
    - if (handCheck(predictions)) – if true it will addScore(pts) and fruitReset()

- else if (key == ‘p’) - all the same as if spacebar were hit but without the particle system added in because of visual overstimulation
- else if (key == ‘f’) - feedback → from Keith’s previously shown code
- else if (key == ‘h’) - just draw the hand (drawPoints(predictions)) to make sure handpose is working

Evaluation and Reflection of Improvements

I achieved all of my main tasks for the actual project and I learned so much about how different kinds of elements interact since they often do not interpret the screen in the same way, but I definitely had things I wanted to add that came to me too late to fully implement. With more time, I would probably have made an opening menu screen, perhaps other kinds of gameplay to choose from, some kind of genuine score-keeping that was not just for the sake of knowing the game was understanding each object as a different object. At the moment it is endless and randomized scores that do not actually do anything besides remind you that you are getting points and the program is updating. Since the scoring did not do much, I made the decision to only print to the console instead of print on the screen. As I continued working I would come up with a new idea to add in that would excite me, but I would notice how much more work was necessary to reach the level that I was hoping for, and due to time constraints I was not going to achieve. This is easily noticeable based on the “score” system that is implemented and functional but not quite at the level that I was happy with so I removed it from the main screen.

DEMO VIDEO:
https://drive.google.com/file/d/1XJH757ZmmMb55yz_SSzKTBMm-acjWeGq/view?usp=drivesdk