


#### Project Overview:

We made two different types of games. One is a frenzied whack-a-mole game where the player clicks on the blinking mole, and the other is a Guac-A-Mole action-packed game where the player slices avocado. These games are fundamentally similar but use different methods to win. We'll be presenting Guac-A-Mole as our official class demo.

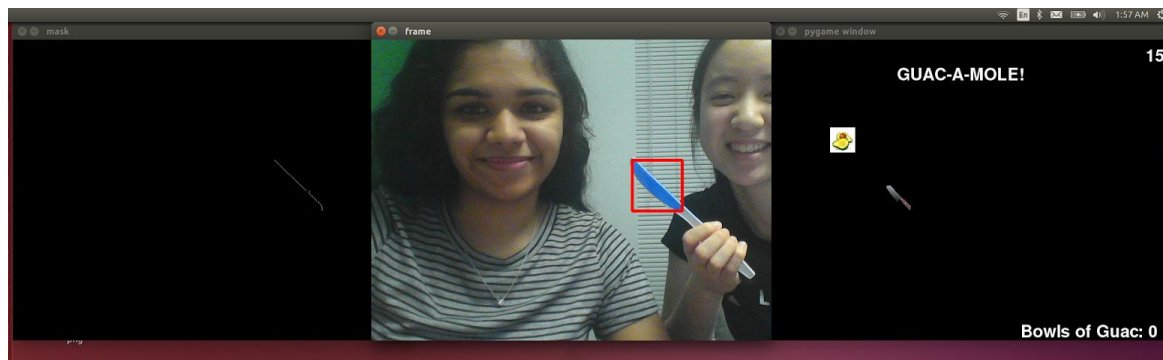
#### Results:

In these projects, we successfully integrated elements from OpenCV and PyGame to integrate color detection, to portray a blue knife as a pseudo-cursor, to "slice" the disappearing avocado for guacamole. This idea was a combination of our two most-popular ideas: fruit ninja and Whack-a-mole. We also made another game that utilizes the clicking features to decide when to whack the mole, which was our proposed MVP.

We also accomplished a series of algorithms that keeps two scripts running at the same time. These concurrent processes were essential to the results of our second game.

	<p>Our MVP, Mouse Controlled Whack-a-Mole:</p> <p>In the image to the left, the screen shows this Whack-a-Mole implementation where the user gets points by whacking the mole before it moves to a different position on the screen. The user also gets a "Failed Whack" point by clicking and missing the mole, and has up to 3 "Failed Whack" chances before the game is over. The goal is to get the highest score before getting three failed whacks.</p>
<p>Guac-a-Mole Implementation: In the image below, the screen on the left is a binary image showing everything blue as white and everything else as black. The second screen is the webcam with a red rectangle around the portion of the frame detected to be blue. We used the center of the rectangle as the position of the knife cursor seen in the third screen, which is the Guac-a-Mole game screen. It is here that</p>	

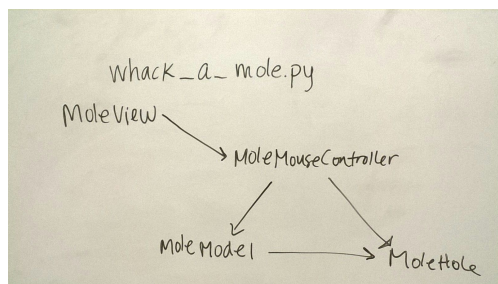
you can slice the avocados into guacamole. The goal of this version of the game is to make as many bowls of Guac as possible before the timer runs out.



## Implementation

PyGame was used to connect all the classes together. PyGame first has a MoleView class that makes the initial black screen, and draws all the objects to visualize the entire game. Pygame is also responsible for changing the images once MoleMouseController indicates the knife's intersection with the avocado. The MoleMouseController class manipulates the color input to move the knife cursor and checks to see if the knife's position is in the avocado. MoleModel stores the attributes of the objects, updates the status of the chosen "mole hole" and the timer, ends the game once time runs out, checks the click attribute to decide when the avocado has been sliced (which should call the guacamole image in the MoleView). The MoleHole class makes holes where the avocado pops out.

This UML implementation is constant among both of our games.



In the Guac-a-Mole game, the PyGame file also interacts with our OpenCV file through multiprocessing. The OpenCV file produces a list data type that stores the [x,y] position of the center point of the red rectangle (in the results section table). This list is used in the PyGame file as an attribute of the MoleModel class and is used to position the knife cursor that is displayed through whack\_a\_mole.py's view object.

In the `whack_a_mole.py` (the game with the OpenCV), we had various alternatives for lines 155-157 where we tried out multiple alternatives. We first tried:

```
if ticks%1000 == 0:  
    model.update()
```

This implementation was problematic because the program would not update when the time elapsed was not divisible by 1000. After that problem we tried making `ticks%1000 <= 50` to account for various elapsed times. Although that line made the program run better, the screen often froze for a while. As a result of those mistakes, we decided to replace those lines with:

```
155     if ticks - previous_update > 1000:  
156         previous_update = ticks  
157         model.update()
```

This way, we made sure that we had the most up-to-date code, because the `previous_update` would never be more than 1 second behind the current tick time.

### **Reflection:**

We were not sure how to write unit tests for our particular functions, so we were often confused about debugging our code. Some of these problems include getting an image to change quickly enough after a click or slice, and understanding attributes conceptually. In hindsight, this could have been solved if we tested out all different cursor or movement interactions before moving on to the next part. Although we did implement attributes, we weren't always sure how to call them at the right times and order.

On the other hand, we are happy to complete a project of this scope that implements parts from both OpenCV and PyGame. We learned a lot about integrating various packages and features together. If we were to do this project again, we would have completed the UML diagram beforehand and organized our code beforehand.

We decided to do pair programming, because this method worked well in ModSim. We definitely worked well together, but when one person tries to progress on her code while the other can't attend a meeting, though, the other person becomes unfamiliar with the code. To resolve this issue, we could ask each other what the purpose of each section is (to practice explaining the code and be more familiar with our project) in future projects.