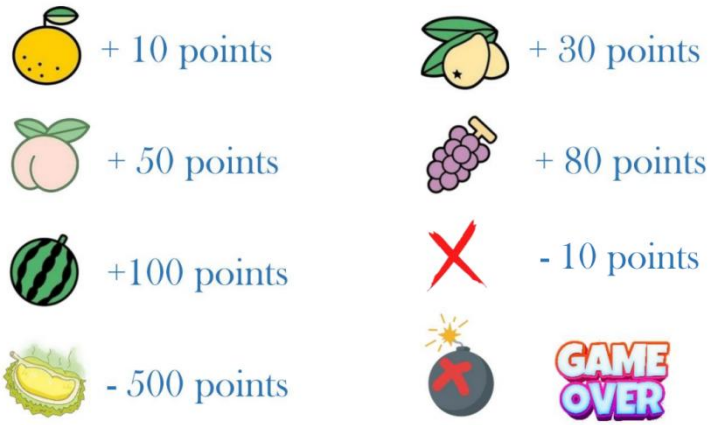
CS 4187 Mini Project Report

Fruits-Geek

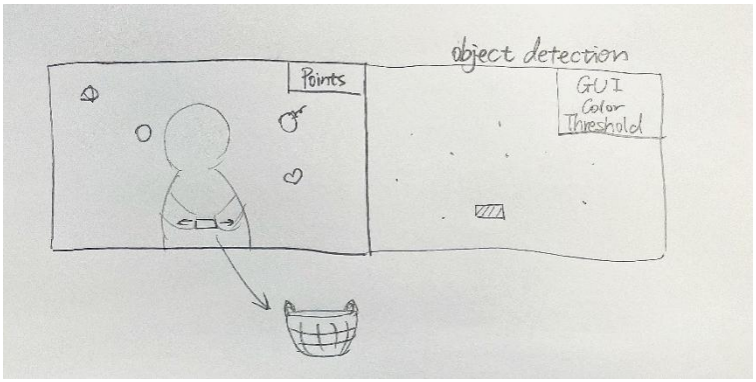
MAO Yaxuan
56641467

➢ Features Description

This is a game called *Fruits-Geek*. Users may take an object detected as a basket  to catch the fruits dropping down. Each fruit has a unique point as shown below:



And there's a game design prototype shown below:



The view on the right is the main game screen and there will be fruits (or bombs) dropping down randomly. The left view is the HSV adjustment

area, where the player adjusts the parameters according to the color of the object in hand that he wants to interact with the computer. When the left view shows white for only the object position and black for the other positions, the parameter is proven to be adjusted to the appropriate value. After adjusting the parameters and starting the game, a basket will appear on the right interface in the center of the detected object. The player moves the object in his hand to manipulate the basket on the screen to catch the fruit. When catching fruit, avoid receiving durians and bombs, and points will be deducted if the fruit is missed.

- System Guideline

1. Bring the selected object (the more special the color, the better) to the camera and adjust the parameters according to the display on the left screen.
2. Click the key *s* to start the game.
3. Move the object to manipulate the basket on the screen to catch the fruit. If you receive an orange  , mango  , peach  , grape  , or watermelon  , you score 10, 30, 50, 80, and 100 points respectively. 10 points will be deducted if the fruit is missed ✕. Receiving a durian  will result in a 500-points deduction. Receiving a bomb  is a direct game over. When the total score is less than 0, it will also result at the end of the game.

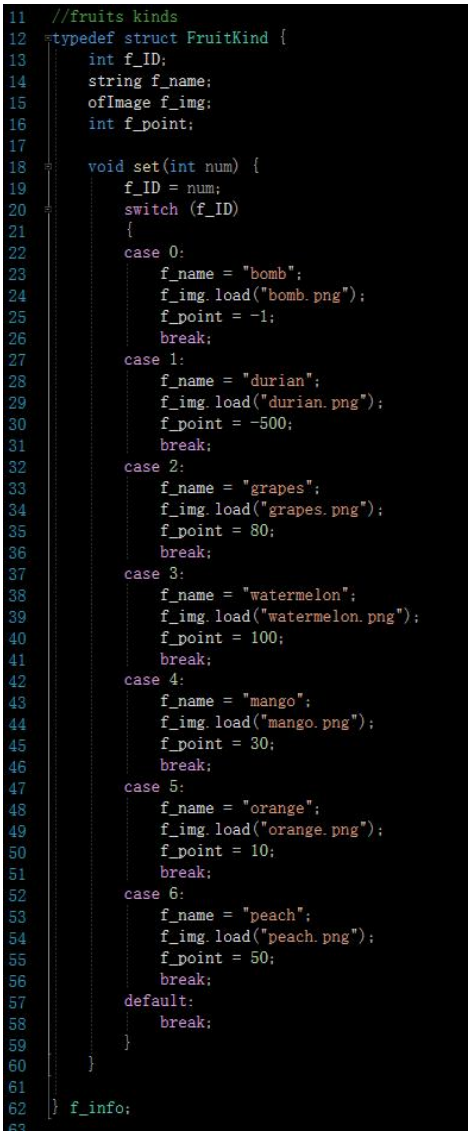
4. Other operators:

    a) Click p to pause the game (only can use when the game is playing).

    b) Click q to exit the game (can use when the game is playing and the game is over). It will bring the player to the initial page.

- Implement

Start by establishing a typed struct in the header file, which is used to record the details of each fruit (object) that drops down, including the fruit's ID, name, image, and scores.



```
//fruits kinds
typedef struct FruitKind {
    int f_ID;
    string f_name;
    ofImage f_img;
    int f_point;

    void set(int num) {
        f_ID = num;
        switch (f_ID)
        {
        case 0:
            f_name = "bomb";
            f_img.load("bomb.png");
            f_point = -1;
            break;
        case 1:
            f_name = "durian";
            f_img.load("durian.png");
            f_point = -500;
            break;
        case 2:
            f_name = "grapes";
            f_img.load("grapes.png");
            f_point = 80;
            break;
        case 3:
            f_name = "watermelon";
            f_img.load("watermelon.png");
            f_point = 100;
            break;
        case 4:
            f_name = "mango";
            f_img.load("mango.png");
            f_point = 30;
            break;
        case 5:
            f_name = "orange";
            f_img.load("orange.png");
            f_point = 10;
            break;
        case 6:
            f_name = "peach";
            f_img.load("peach.png");
            f_point = 50;
            break;
        default:
            break;
        }
    }

} f_info;
```

Then, after some basic parameters (such as camera parameters, GUI and threshold parameters, blob parameters, image parameters, etc.) have been defined, some parameters regarding the game interface are defined.

```
public:
    void setup();
    void update();
    void draw();
    void keyPressed(int key);
    void randomDropDown(f_info& fruit);
    int randomDropDownPos();
    int randomDropDownTim();

    //Get the camera equipment
    ofVideoGrabber vidGrabber;
    ofImage vidFrame;
    Mat vidFrameMat;

    Mat mat_HSV;
    Mat mat_HSV_Threshold;

    //Get GUI
    ofxPanel gui;
    //threshold
    ofxIntSlider minH;
    ofxIntSlider maxH;
    ofxIntSlider minS;
    ofxIntSlider maxS;
    ofxIntSlider minV;
    ofxIntSlider maxV;

    Mat result;

    ofxCvContourFinder contourFinder;

    int objCentre_x = -1;
    int objCentre_y = -1;

    //images
    ofImage imgbasket;
    ofImage imgmiss;
    ofImage imggameover;
    ofImage imgpause;
```

The *struct FRUIT* contains some parameters of the fruit that appears on the screen each frame (information about the fruit, the speed of fall, the x and y coordinates, and the y coordinates of the previous second). And these *FRUIT* structs are stored in an array called *fruArr[ ]*. And the *int time* represents the time (in frames) between the last fruit to appear and the present.

```
struct FRUIT
{
    f_info info;//fruits' info (including fruits ID, fruits name, fruits img, fruits score)

    int tempY;
    int speed;
    int posX;
    int posY;
    bool state;

};

int time = 0;
struct FRUIT fruArr[20];
int totalScore = 0;

int gameStateID = 0; // 0:initial  1:game playing  2:game over  3:pause
```

In addition, for the CPP file, set up the GUI, camera, and images first.

```
//--------------------------------------------------------------
void ofApp::setup() {
    //set up the GUI panel
    gui.setup();
    //add GUI elements
    gui.add(minH.setup("min H", 0, 0, 180));
    gui.add(maxH.setup("max H", 180, 0, 180));
    gui.add(minS.setup("min S", 0, 0, 255));
    gui.add(maxS.setup("max S", 255, 0, 255));
    gui.add(minV.setup("min V", 0, 0, 255));
    gui.add(maxV.setup("max V", 255, 0, 255));

    //Camera setting
    vidGrabber.setDeviceID(0);
    vidGrabber.setDesiredFrameRate(30);
    vidGrabber.initGrabber(640, 480);

    //load images
    imgbasket.load("basket.png");
    imgmiss.load("miss.png");
    imggameover.load("GAME_OVER.png");
    imgpause.load("pause.png");
}
```

The next update function mainly implements blob detection, which is the main method of interaction in this game. Blob detection is aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions.

```
//--------------------------------------------------------------
void ofApp::update() {
    vidGrabber.update();

    //blob detecction
    if (vidGrabber.isFrameNew()) {
        vidFrame.setFromPixels(vidGrabber.getPixels());
        vidFrameMat = toCv(vidFrame);
        cvtColor(vidFrameMat, mat_HSV, CV_BGR2HSV);
        inRange(mat_HSV, Scalar(minH, minS, minV), Scalar(maxH, maxS, maxV), mat_HSV_Threshold);
        erode(mat_HSV_Threshold, mat_HSV_Threshold, Mat());
        dilate(mat_HSV_Threshold, mat_HSV_Threshold, Mat());

        ofImage im_temp;
        ofxCvGrayscaleImage im_temp_gray;

        toOf(mat_HSV_Threshold, im_temp);

        im_temp_gray.setFromPixels(im_temp.getPixels());

        contourFinder.findContours(im_temp_gray, 5, (640 * 480) / 4, 4, false, true);
    }
}
```

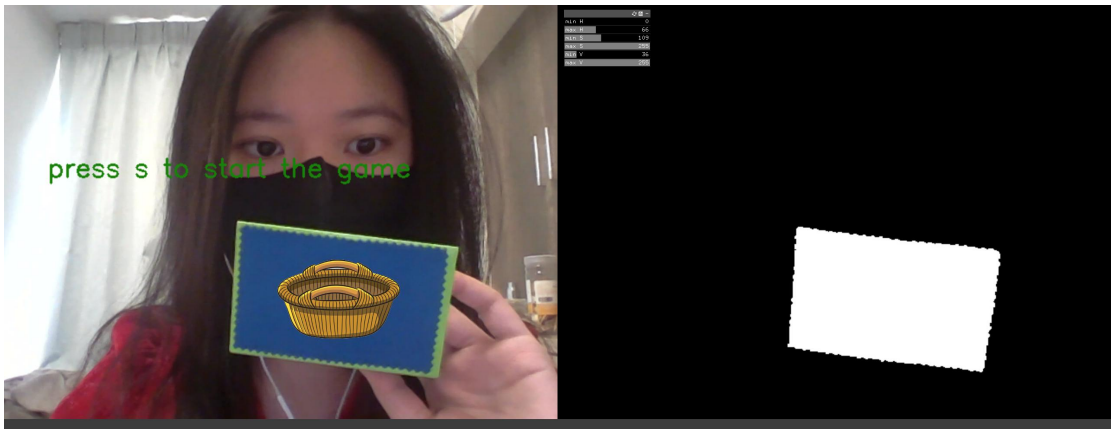
The draw function is the most important part of a game's presentation. It is divided into four main parts by means of if conditions -- initial, game playing, game over, and pause.

· Initial:

This part of the screen shows the player's webcam and the blob detect screen, where the player has to adjust the GUI so that the object they are

holding is detected. In this case, the basket of fruit will appear in the centre of the detected object.

```
// initial
if (gameStateID == 0) {
    putText(vidFrameMat, "press s to start the game", Point(50, 200), FONT_HERSHEY_SIMPLEX, 1, Scalar(25, 130, 10), 2, 4);
    // blob detection
    ofSetColor(255, 255, 255);
    drawMat(vidFrameMat, 0, 0, 1280, 960);
    for (int i = 0; i < contourFinder.nBlobs; i++) {
        objCentre_x = contourFinder.blobs.at(i).centroid.x;
        objCentre_y = contourFinder.blobs.at(i).centroid.y;
    }
    ofSetColor(255, 255, 255);
    drawMat(mat_HSV_Threshold, vidFrameMat.cols * 2, 0, 1280, 960);

    //draw the basket
    if (objCentre_x > 0 && objCentre_y > 0) {
        imgbasket.draw(objCentre_x * 2 - 150, objCentre_y * 2 - 100, 300, 200);
    }
}
```



Press s to start the game.

```
void ofApp::keyPressed(int key) {
    if (gameStateID == 0 || gameStateID == 3) {
        if (key == 's') {
            gameStateID = 1;
        }
    }
```

· Game Playing

The interface for playing the game not only displays the player shots and the blob detection screen but also automatically generates fruit and keeps score. The fruits on the screen are recorded in fruArr[], an array that is subject to change at any time. When a new fruit is generated, the information about this new fruit is stored in this array and its speed is defined randomly as a value from 7 to 15. The speed of the fruit is set as a

parameter to determine whether the fruit is valid or not. When the speed of the fruit is 0, the fruit is considered invalid and does not need to be printed.

When the fruit (except durian) is within the basket, the fruit disappears and the score increases. If the location of the fruit (except durian) is greater than 800 (missing), the fruit disappears, a red fork appears and the score decreases. If the durian coordinates are within the basket, the durian disappears and the score decreases. If the score is less than 0 or if the bomb coordinates are within the basket, the game state ID is assigned to 2 and the game over state is entered.

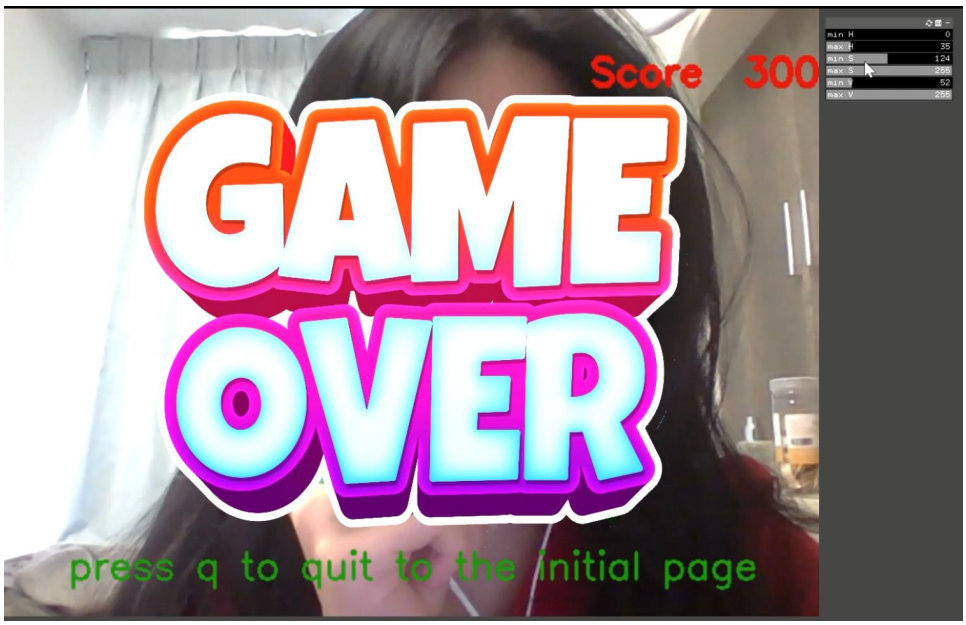
```
72      // game playing
73      if (gameStateID == 1) {
74          putText(vidFrameMat, "Score  " + to_string(totalScore), Point(460, 60), FONT_HERSHEY_SIMPLEX, 1, Scalar(205, 23, 10), 4, 8);
75          // blob detection
76          ofSetColor(255, 255, 255);
77          drawMat(vidFrameMat, 0, 0, 1280, 960);
78          for (int i = 0; i < contourFinder.nBlobs; i++) {
79              objCentre_x = contourFinder.blobs.at(i).centroid.x;
80              objCentre_y = contourFinder.blobs.at(i).centroid.y;
81          }
82          ofSetColor(255, 255, 255);
83          drawMat(mat_HSV_Threshold, vidFrameMat.cols * 2, 0, 1280, 960);
84
85          //draw the basket
86          if (objCentre_x > 0 && objCentre_y > 0) {
87              imgbasket.draw(objCentre_x * 2 - 150, objCentre_y * 2 - 100, 300, 200);
88          }
89
90          //draw the fruits
91          for (int i = 0; i < 20; i++) {
92              if (fruArr[i].speed != 0) {
93                  fruArr[i].tempY = fruArr[i].posY;
94                  fruArr[i].posY += fruArr[i].speed;
95                  fruArr[i].info.f_img.draw(fruArr[i].posX, fruArr[i].posY, 150, 150);
96                  if (fruArr[i].tempY >= 800) {
97                      fruArr[i].speed = 0;
98                  }
99                  if (fruArr[i].tempY >= 800 && fruArr[i].info.f_ID != 0 && fruArr[i].info.f_ID != 1) {
100                     imgmiss.draw(fruArr[i].posX, fruArr[i].posY);
101                     totalScore -= 10;
102                 }
103                 if (((fruArr[i].posX < objCentre_x * 2 + 150) && (fruArr[i].posX > objCentre_x * 2 - 150))
104                     && ((fruArr[i].posY < objCentre_y * 2 + 100) && (fruArr[i].posY > objCentre_y * 2 - 100))
105                     && objCentre_x != -1 && objCentre_y != -1) {
106                     if (fruArr[i].info.f_ID == 0) {
107                         gameStateID = 2;
```

```
                    }
                    totalScore += fruArr[i].info.f_point;
                    fruArr[i].speed = 0;
                }
                if (totalScore < 0) {
                    gameStateID = 2;
                }
            }
        }
        if (time > randomDropDownTim()) {
            for (int i = 0; i < 20; i++) {
                if (fruArr[i].speed == 0) {

                    randomDropDown(fruArr[i].info);//random generate the fruits' information

                    fruArr[i].posX = randomDropDownPos();
                    fruArr[i].posY = 0;
                    fruArr[i].tempY = 0;
                    fruArr[i].speed = rand() % 8 + 7;
                    fruArr[i].info.f_img.draw(fruArr[i].posX, fruArr[i].posY, 150, 150);
                    break;
                }
            }
            time = 0;
        }
        else {
            time++;
        }
    }

```

· Game over

When you enter the game over state, the game over a symbol is displayed and all the fruit on the screen disappears. Also, zero out the total score. Then set the elements in fruArr[ ] to be invalid.

```
    // game over
    if (gameStateID == 2) {
        putText(vidFrameMat, "press q to quit to the initial page", Point(50, 450), FONT_HERSHEY_SIMPLEX, 1, Scalar(25, 130, 10), 2, 4);
        drawMat(vidFrameMat, 0, 0, 1280, 960);
        imggameover.draw(0, 0, 1280, 960);
        for (int i = 0; i < 20; i++) {
            fruArr[i].speed = 0;
        }
        totalScore = 0;
    }
```



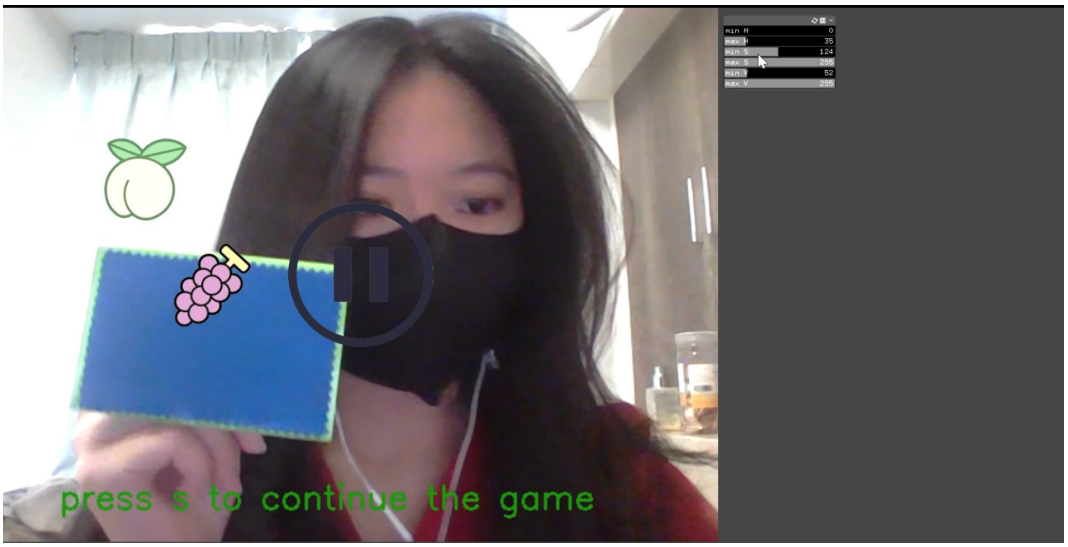
When the game is playing, users can press p to pause the game.

```
if (gameStateID == 1) {
    if (key == 'p') {
        gameStateID = 3;
    }
}
```

· Pause

Entering the pause state displays the pause sign and prints the last position of the fruit.

```
149     // pause
150     if (gameStateID == 3) {
151         putText(vidFrameMat, "press s to continue the game", Point(50, 450), FONT_HERSHEY_SIMPLEX, 1, Scalar(25, 130, 10), 2, 4);
152         drawMat(vidFrameMat, 0, 0, 1280, 960);
153         for (int i = 0; i < 20; i++) {
154             if (fruArr[i].speed != 0) {
155                 fruArr[i].info.f_img.draw(fruArr[i].posX, fruArr[i].posY, 150, 150);
156             }
157         }
158         imgpause.draw(640 - imgpause.getWidth() / 2, 480 - imgpause.getHeight() / 2);
159     }
```



Other functions:

· keypressed function

When pressing some unique key, the game state ID will be changed, and draw the different game interfaces.

```
165  //--------------------------------------------------------------
166 void ofApp::keyPressed(int key) {
167     if (gameStateID == 0 || gameStateID == 3) {
168         if (key == 's') {
169             gameStateID = 1;
170         }
171     }
172
173     if (key == 'q') {
174         gameStateID = 0;
175     }
176
177     if (gameStateID == 1) {
178         if (key == 'p') {
179             gameStateID = 3;
180         }
181     }
182
183 }
```

· functions to randomly generate the objects (fruits)

The fruit_generate_set[18] is to set the probability of each object (fruit) appearing. The chances of occurrence: bomb(1/18), durian(1/9), grapes(1/9), watermelon(1/18), mango(2/9), orange(5/18), peach(1/6). The

drop-down position is randomly generated in positions 60 - 1110, at intervals of 30. The random drop-down time is generated between 20 -50.

```
185  //--------------------------------------------------------------
186 void ofApp::randomDropDown(f_info& fruit) {
187     int fruit_generate_set[18] = { 0, 1, 1, 2, 2, 6, 6, 6, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5 };//represents fruit ID
188
189     /*f_name        f_ID
190       "bomb"        0
191       "durian"      1
192       "grapes"      2
193       "watermelon"  3
194       "mango"       4
195       "orange"      5
196       "peach"       6;*/
197
198     int k = rand() % 18;
199     fruit.set(fruit_generate_set[k]);
200 }
201
202  //--------------------------------------------------------------
203 int ofApp::randomDropDownPos() {
204     int k = (rand() % 35 + 2) * 30;
205     return k;
206 }
207
208
209  //--------------------------------------------------------------
210 int ofApp::randomDropDownTim() {
211     int k = rand() % 30 + 20;
212     return k;
213 }
```

- Future Work

Can try more options for color tracking and pick a better one. At this stage, color tracking is still a little unstable. Also, sound effects can be added. Each time a fruit is received or missed, a specific sound effect can be applied.

- Video Link

https://youtu.be/rMBQbMb1KYY