TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

186.A25 Building Interaction Interfaces

Project 2B - “Hacking the Kinect”

Banafsheh Alinezhad - 1229630

Miguel Brito - 1428574

Pedro Santos - 1428569

Introduction

This report focuses on presenting and documenting the project developed on the scope of the Building Interaction Interfaces course. This project had as main goals to get students familiarized with Kinect and in which ways we could interact with it and build some application taking part of what Kinect offers to the programmer. The following chapters describe in a detailed way the main aspects of this project.

Project description

This project has two main components, being them:

- **Kinect device** - this device is responsible for capturing the players involved in the game and provides mechanisms to the programmer to he could extract information from those movements;
- **Processing application** - this application is what extracts the information that Kinect's transmitting and processes it so the game can flow normally. The graphism of the game and its logic are also built using processing. On this component two libraries were used, one to help managing the information received from the kinect and another to help controlling the physics part of this project. The libraries are "**Kinect4WinSDK"** and "**Fisica**" respectively;

So, after the components are explained, next the concept of the game is going to be detailed.

This is a game where each user has a racquet and there is one (or more, depending on the game mode) ball and the users must hit the ball with their racquet in order not to let it fall. As said before, there are two game modes, being:

- **One vs One** - This game mode starts when there are two users playing and there exists only one ball, and the objective is to compete and make the ball fall down in the opponent's side;
- **All Around** - On this game mode every player has a ball (each time that a player enters, a new ball is added to the game) and the objective is not to let your ball fall down, in order to get more points than the adversaries. You get one point for each time that you hit the ball and lose all of the points when the ball falls.

The racquet appears on the right hand of each player and once it appears is handled by him. Also, the racquet is rotated as the user rotates his pulse/hand so that he can hit the ball with different angles.

Tutorials used and Information sources

To start this project we checked a few tutorials to understand better how to work with some components of the system.

- **Fisica documentation** (http://www.ricardmarxer.com/fisica/) - Get information about fisica's library, its objects and their methods/properties;

- **Processing documentarion** (https://www.processing.org/reference/) - Get information about some of the processing functions, and how to use some functionalities (Text on the screen, for example);
- **Fisica example** (http://www.ricardmarxer.com/fisica/examples/ContactRemove/applet/index.html) - this example was used to see how fisica works and how to setup an application with fisica and some posible uses of it;
- **Connecting Kinect to Processing example** (http://www.magicandlove.com/blog/research/kinect-for-processing-library/) - An example of how to connect the Kinect to a Processing application. From this tutorial we used the setup component and saw how to extract some of the information (like the position of some body parts, for example);

Processing code

Main functionalities

- **Creation of the racquet** - when a body appears (identified by the appear event), the racquet is placed on the right hand of the player (by omission all players are right-handed), using the coordinates read from the Kinect for the right hand of the player;
- **Movement of the racquet** - Every time that there's a move event, the corresponding player is identified and the position/angle of the racquet from that player is updated based on the right hand of that player;
- **Processing the contact between the racquet and the ball** - When the contact event is detected, it's applied an impulse on the ball to make it move and bounce. This impulse has a direction which is calculated through the angle of the racquet;
- **Choosing the game modes** - On the beginning of the game there are 2 selection areas on the screen, where the user can choose the game mode. To do this, one of the active players have to put their right hand inside the box that they want to choose. After this the game is started;

Plain code

Next is presented the plain code from de Processing application.

```
import fisica.*;
import kinect4WinSDK.*;
```

```
final int HEIGHT = 720;
final int WIDTH = 960;
int ballCount = 0;

Kinect kinect;
ArrayList <SkeletonData> bodies;
HashMap <Integer, FBox> racquets;
HashMap <Integer, Integer> result;
HashMap <Integer, Integer> best;
HashMap <Integer, FCircle> balls;

PFont f;

GameState state;

FWorld world;

int RED_X = 100;
int BLUE_X = 500;
int MENU_Y = 10;
int MENU_WIDTH = 300;
int MENU_HEIGHT = 150;

void setup()
{
 size(WIDTH, HEIGHT);
 smooth();

 state = GameState.INITIAL_MENU;

 kinect = new Kinect(this);
 bodies = new ArrayList<SkeletonData>();

 racquets = new HashMap<Integer, FBox>();
 Fisica.init(this);
 world = new FWorld();

 result = new HashMap<Integer, Integer>();
 best = new HashMap<Integer, Integer>();
 balls = new HashMap<Integer, FCircle>();

 f = createFont("Arial",26,true);
 world.setGravity(0,500);
}

void draw()
{
  background(255,255,255);
  try{
   image(kinect.GetImage(), 0, 0, WIDTH, HEIGHT);
  } catch (Exception e) {
   println("No Kinect");
  }
  drawResults();
  try{
    synchronized(world){
      world.draw();
      world.step();
    }
  } catch (Exception e){
    println("No world");
  }
  checkTheBalls();
  /*synchronized(bodies){
    for (int i=0; i<bodies.size (); i++)
    {
      drawSkeleton(bodies.get(i));
    }
  }*/
  if(state == GameState.INITIAL_MENU){
   checkAndInside();
   drawMenu();
  }
}

void checkTheBalls(){

 ArrayList<Integer> toRemove = new ArrayList<Integer>();
 HashMap<Integer, FCircle> toAdd = new HashMap<Integer, FCircle>();

 synchronized(balls){
  for(Integer id : balls.keySet()/*FCircle ball : balls*/){
    if(balls.get(id).getX() < 0 || balls.get(id).getX() > width ||
balls.get(id).getY() > height){

     synchronized(world){
      world.remove(balls.get(id));
     }

     synchronized(result){
      if(result.containsKey(id)){
        result.put(id,0);

        FCircle ball = new FCircle(25);
        synchronized(bodies){
          for(int i = 0; i < bodies.size(); i++){
           if(bodies.get(i).dwTrackingID == id){
             ball.setPosition(bodies.get(i).position.x*width/*random(0+10,
width-10)*/, 0);
           }
          }
        }

        ball.setRestitution(0.75);
        ball.setFill(255,102,0);

        synchronized(world){
          world.add(ball);
        }

        toAdd.put(id, ball);
     } else {
       toRemove.add(id);
       ballCount--;
     }
    }
   }
  }

  for(Integer i : toRemove){
   balls.remove(i);
  }

  balls.putAll(toAdd);
 }
```

```
}

void drawResults(){
 synchronized(bodies){
   for(SkeletonData _s : bodies){
    synchronized(result){
     textFont(f,26);
     fill(0,0,255);

text(result.get(_s.dwTrackingID),_s.skeletonPositions[Kinect.NUI_SKELETON_P
OSITION_HEAD].x*width,_s.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HEA
D].y*height-50);
    }
    synchronized(best){
     textFont(f,26);
     fill(255,0,0);

text(best.get(_s.dwTrackingID),_s.skeletonPositions[Kinect.NUI_SKELETON_POS
ITION_HEAD].x*width,_s.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HEAD]
.y*height-75);
    }
   }
 }
}

boolean isInsideOneVsOne(float x, float y){
 if( x > RED_X && x < (RED_X + MENU_WIDTH) && y > MENU_Y && y < (MENU_Y +
MENU_HEIGHT))
    return true;
 else
   return false;
}

boolean isInsideAllAround(float x, float y){
  if( x > BLUE_X && x < (BLUE_X + MENU_WIDTH) && y > MENU_Y && y < (MENU_Y
+ MENU_HEIGHT))
    return true;
 else
    return false;
}
void checkAndInside(){
 synchronized(bodies){
   for(int i = 0; i < bodies.size(); i++){
    float x =
bodies.get(i).skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].x*
width;
    float y =
bodies.get(i).skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].y*
height;

    if(isInsideOneVsOne(x,y)){
      state = GameState.ONEVSONE;
      println("State changed to onevs one");
    }
    if(isInsideAllAround(x,y)){
      state = GameState.ALLAROUND;
      startAllAround();
      println("State changed to allaround");
    }
   }
 }
}

void startAllAround(){
  synchronized(bodies){
    for(int i = 0; i < bodies.size(); i++){
      FCircle ball = new FCircle(25);
      ball.setPosition(bodies.get(i).position.x*width/*random(0+10,
width-10)*/, 0);
      ball.setRestitution(0.75);
      ball.setFill(255,102,0);

      synchronized(world){
        world.add(ball);
      }

      synchronized(balls){
       balls.put(bodies.get(i).dwTrackingID, ball);
      }
      ballCount++;

      synchronized(result){
        result.put(bodies.get(i).dwTrackingID, 0);
      }

      synchronized(best){
        best.put(bodies.get(i).dwTrackingID, 0);
      }
    }
  }
}

void drawMenu(){
 fill(255,0,0);
 rect(RED_X,MENU_Y,MENU_WIDTH,MENU_HEIGHT);

 textFont(f,26);
 fill(255);
 text("One vs One",175,100);

 fill(0,0,255);
 rect(BLUE_X,MENU_Y,MENU_WIDTH,MENU_HEIGHT);

 textFont(f,26);
 fill(255);
 text("All Around",575,100);

 textFont(f,26);
 fill(255);
 text("With your right hand, pick a game mode!",400,200);

}

void appearEvent(SkeletonData _s)
{
  if (_s.trackingState == Kinect.NUI_SKELETON_NOT_TRACKED)
  {
    return;
  }
  synchronized(bodies) {
    bodies.add(_s);
    //println("A body appeard. Id is " + _s.dwTrackingID + ". The number of
bodies is " + bodies.size());
    synchronized(world){
      synchronized(racquets){
       FBox _r = new FBox(200, 50);

_r.setPosition(_s.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT
].x*width,
_s.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].y*height);
       _r.setStatic(true);
       _r.setFill(0);
       _r.setRestitution(0);
       world.add(_r);
       racquets.put(_s.dwTrackingID, _r);
      }
      if((state == GameState.ONEVSONE && ballCount == 0) || state ==
GameState.ALLAROUND){
        FCircle ball = new FCircle(25);
        ball.setPosition(_s.position.x*width/*random(0+10, width-10)*/, 0);
        ball.setRestitution(0.75);
        ball.setFill(255,102,0);
        world.add(ball);

        synchronized(balls){
         balls.put(_s.dwTrackingID, ball);
        }

        ballCount++;
      }
      synchronized(result){
        result.put(_s.dwTrackingID, 0);
      }

      synchronized(best){
        best.put(_s.dwTrackingID, 0);
      }
    }
  }
}

void disappearEvent(SkeletonData _s)
{
  int id = _s.dwTrackingID;
  //println("A body disappeard. Id was " + _s.dwTrackingID);
  synchronized(bodies) {
    for (int i=bodies.size ()-1; i>=0; i--)
    {
      if (0 == bodies.get(i).dwTrackingID)
      {
        bodies.remove(i);
        synchronized(racquets){
          synchronized(world){
            world.remove(racquets.get(id));
          }
          racquets.remove(id);
        }

        synchronized(result){
          result.remove(id);
        }

        synchronized(best){
          best.remove(id);
        }
      }
    }
  }
  //println("Bodies size is " + bodies.size());
}

void moveEvent(SkeletonData _b, SkeletonData _a)
{
  float deltaY, deltaX, angle;
  if (_a.trackingState == Kinect.NUI_SKELETON_NOT_TRACKED)
  {
    return;
  }
  synchronized(racquets){
    try{
      synchronized(world){

racquets.get(_a.dwTrackingID).setPosition(_a.skeletonPositions[Kinect.NUI_S
KELETON_POSITION_HAND_RIGHT].x*width,
_a.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].y*height);
        deltaY =
_a.skeletonPositions[Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT].y -
_a.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].y;
        deltaX =
_a.skeletonPositions[Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT].x -
_a.skeletonPositions[Kinect.NUI_SKELETON_POSITION_HAND_RIGHT].x;
        if(deltaX != 0){
          angle = (float)Math.atan(Math.toRadians(deltaY/deltaX))*180/3.14;
        } else {
          if(deltaY > 0){
           angle = 90;
          } else {
           angle = 270;
          }
        }
        racquets.get(_a.dwTrackingID).setRotation(angle);
      }

    } catch (Exception e){
      println("não falhes :(");
    }
  }

  try{
    synchronized(bodies) {
      for (int i=bodies.size ()-1; i>=0; i--)
      {
        if (_b.dwTrackingID == bodies.get(i).dwTrackingID)
        {
          bodies.get(i).copy(_a);
          break;
        }
      }
    }
  } catch (Exception e) {
   println("SCHEEEEIIIIIZE");
  }
}

void contactStarted(FContact c) {
  FBody body1 = c.getBody1();
  FBody body2 = c.getBody2();
  synchronized(racquets){
    synchronized(world){
      if(racquets.containsValue(body1)){
       if(!racquets.containsValue(body2)){
          body2.addImpulse(0,200);

          if(state == GameState.ALLAROUND){
            incrementResultOfBody(body1);
          }
       }
      }else{
        if(racquets.containsValue(body2)){
          body1.addImpulse(0,200);
          if(state == GameState.ALLAROUND){
            incrementResultOfBody(body2);
          }
       }
      }
    }
  }
}

void incrementResultOfBody(FBody body){
  for(Integer id : racquets.keySet()){
   if(racquets.get(id) == body){
    synchronized(result){
      result.put(id, result.get(id) + 1);

      synchronized(best){
        if(result.get(id) > best.get(id)){
          best.put(id, result.get(id));
        }
      }
      break;
     }
   }
  }
}

//DEBUG
void drawSkeleton(SkeletonData _s)
{
  // Body
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HEAD,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_CENTER);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_CENTER,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_CENTER,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_CENTER,
  Kinect.NUI_SKELETON_POSITION_SPINE);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_LEFT,
  Kinect.NUI_SKELETON_POSITION_SPINE);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_RIGHT,
  Kinect.NUI_SKELETON_POSITION_SPINE);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SPINE,
  Kinect.NUI_SKELETON_POSITION_HIP_CENTER);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HIP_CENTER,
  Kinect.NUI_SKELETON_POSITION_HIP_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HIP_CENTER,
  Kinect.NUI_SKELETON_POSITION_HIP_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HIP_LEFT,
  Kinect.NUI_SKELETON_POSITION_HIP_RIGHT);

  // Left Arm
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_LEFT,
  Kinect.NUI_SKELETON_POSITION_ELBOW_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_ELBOW_LEFT,
  Kinect.NUI_SKELETON_POSITION_WRIST_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_WRIST_LEFT,
  Kinect.NUI_SKELETON_POSITION_HAND_LEFT);

  // Right Arm
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_SHOULDER_RIGHT,
  Kinect.NUI_SKELETON_POSITION_ELBOW_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_ELBOW_RIGHT,
  Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT,
  Kinect.NUI_SKELETON_POSITION_HAND_RIGHT);
```

```
  // Left Leg
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HIP_LEFT,
  Kinect.NUI_SKELETON_POSITION_KNEE_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_KNEE_LEFT,
  Kinect.NUI_SKELETON_POSITION_ANKLE_LEFT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_ANKLE_LEFT,
  Kinect.NUI_SKELETON_POSITION_FOOT_LEFT);

  // Right Leg
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_HIP_RIGHT,
  Kinect.NUI_SKELETON_POSITION_KNEE_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_KNEE_RIGHT,
  Kinect.NUI_SKELETON_POSITION_ANKLE_RIGHT);
  DrawBone(_s,
  Kinect.NUI_SKELETON_POSITION_ANKLE_RIGHT,
  Kinect.NUI_SKELETON_POSITION_FOOT_RIGHT);
}

void DrawBone(SkeletonData _s, int _j1, int _j2)
{
  noFill();
  stroke(255, 0, 0);
  if (_s.skeletonPositionTrackingState[_j1] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKED &&
    _s.skeletonPositionTrackingState[_j2] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKED) {
    line(_s.skeletonPositions[_j1].x*width,
    _s.skeletonPositions[_j1].y*height,
    _s.skeletonPositions[_j2].x*width,
    _s.skeletonPositions[_j2].y*height);
  }
}
```

Auxiliary GameState.java file

```
public enum GameState{
 INITIAL_MENU, ONEVSONE, ALLAROUND
}
```