**Lab5 CS171**

Meer Akbar Zohair Moosavi

15.11.2022

**Part 1:**

**Network Communication**

**Graphical user interface, text

Description automatically generated**

**Room\_Client.pde:**

import javax.swing.JOptionPane;

import processing.net.\*;

Client c;

String input;

int data[];

void **setup**()

{

  size(50, 50);

  background(204);

  c = new Client(this, "192.168.1.53", 12345); // Replace with your IP and port

}

void **draw**()

{

  String r = JOptionPane.showInputDialog(null,"1223", "Text input", JOptionPane.QUESTION\_MESSAGE);

  c.write(r+"\n");

}

**Room\_Server.pde:**

import processing.net.\*;

import java.net.InetAddress;

InetAddress inet;

Server s;

Client c;

String input;

int data[];

int line=0;

void **setup**()

{

  size(600, 500);

  background(204);

  textSize(48);

  fill(0);

  s = new Server(this, 12345); // Start a simple server on a port 12345

}

void **draw**()

{

  // Receive data from client

  c = s.available();

  if (c != null)

  {

    input = c.readString();

    input = input.substring(0, input.indexOf("\n")); // Only up to the newline

String myIP;

  try {

    inet = InetAddress.getLocalHost();

    myIP = inet.getHostAddress();

  }

  catch (Exception e) {

    e.printStackTrace();

    myIP = "couldnt get IP";

  }

  textSize(30);

  text("IP address: "+myIP,290,40);

    textSize(50);

    text(input,10,48+48\*(line));

    line++;

    if(line==10) {line=0; background(204);}

  }

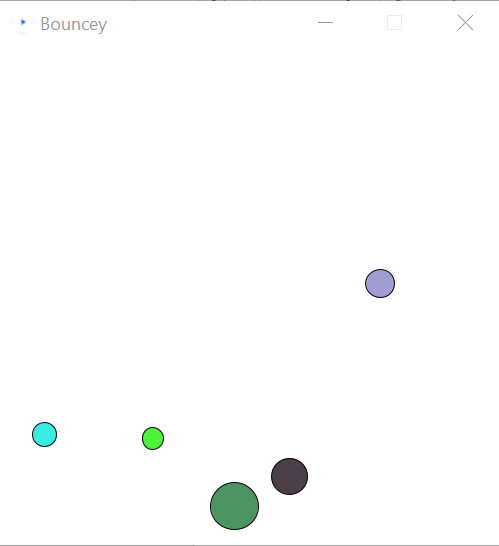
}

**Pong Game**

Answer 1: The client already contains the IP address so there is no need for the server to contain an IP address, since they are sharing and communicating using the same IP.

Answer 2: The code is checking if the string contains exactly 5 pieces of data (length of string must be 5), this way each piece of data is represented as a different function for the game for example, score\_left, score\_right, etc. This fault could occur if either nothing is entered for the string, or the length of the string is not 5 exactly.

**Part 2:**

****

**Figure 2.1** 5 balls with random colours and sizes bouncing around the screen with collision physics (balls bounce off each other)

**Part 3:**

**Diagram

Description automatically generated with medium confidence**

**Figure 3.1** 6 (addition of 1 structure) Bodies defined in Box2D (modified and using the same physics)

import shiffman.box2d.\*;

import org.jbox2d.collision.shapes.\*;

import org.jbox2d.common.\*;

import org.jbox2d.dynamics.\*;

// A list for all of our rectangles

ArrayList<Box> boxes;

Box2DProcessing box2d;

void **setup**()

{

  size(600,300);

  smooth();

   // Initialize and create the Box2D world

  box2d = new Box2DProcessing(this);

  box2d.createWorld();

  // Turn on collision listening!

  box2d.listenForCollisions();

  // Create ArrayLists

  boxes = new ArrayList<Box>();

  Box p = new Box(430,height-61,20,100,0);

  boxes.add(p);

  p = new Box(400,height-61,20,100,0);

  boxes.add(p);

  p = new Box(460,height-61,20,100,0);

  boxes.add(p);

  p = new Box(430,height-122,100,20,0);

  boxes.add(p);

  p = new Box(430,height-5,100,10,1);

  boxes.add(p);

  p = new Box(40,height-52,30,30,0);

  boxes.add(p);

  p.body.setAngularVelocity(-4);

  p.body.setLinearVelocity(new Vec2(30, 10));

}

void **draw**()

{

  background(255);

  // We must always step through time!

  box2d.step();

  // Display all the boxes

  for (Box b: boxes)

  {

    b.display();

  }

}

// A rectangular box

class Box

{

  // Instead of any of the usual variables,

  // we will store a reference to a Box2D Body

  Body body;

  float w,h;

  Box(float x, float y, float w\_,float h\_, int type)

  {

    w = w\_;

    h = h\_;

    // Build Body

    BodyDef bd = new BodyDef();

    if(type==0)

    {

     bd.type = BodyType.DYNAMIC; // Was DYNAMIC

    }

    else

    {

      bd.type = BodyType.STATIC;

    }

    bd.position.set(box2d.coordPixelsToWorld(x,y));

    body = box2d.createBody(bd);

   // Define a polygon (this is what we use for a rectangle)

    PolygonShape sd = new PolygonShape();

    // Box2D considers the width and height of a

    // rectangle to be the distance from the

    // center to the edge (so half of what we

    // normally think of as width or height.)

    float box2dW = box2d.scalarPixelsToWorld(w/2);

    float box2dH = box2d.scalarPixelsToWorld(h/2);

    sd.setAsBox(box2dW, box2dH);

    // Define a fixture

    FixtureDef fd = new FixtureDef();

    fd.shape = sd;

    // Parameters that affect physics

    fd.density = 1;

    fd.friction = 0.3;

    fd.restitution = 0.5;

    // Attach Fixture to Body

    body.createFixture(fd);

  }

  void display()

  {

    // We need the Body’s location and angle

    Vec2 pos = box2d.getBodyPixelCoord(body);

    float a = body.getAngle();

    // Using the Vec2 position and float angle to

    // translate and rotate the rectangle

    pushMatrix();

    translate(pos.x,pos.y);

    rotate(-a);

    fill(175);

    stroke(0);

    rectMode(CENTER);

    rect(0,0,w,h);

    popMatrix();

  }

}  // End of the Box Class description

**Part 4:**

square(0,100,20);

square(100,100,20);

square(100,0,20);

square(0,0,20);