Jakob Hachigian-Kreutzer
Mr. Ettlin
AP Computer Science Principles, Period 1
23 October 2018

PaddleBall Game Questions

1. The PaddleBall Game is a game based around collecting balls to achieve victory created in Javascript. The video reveals the perspective a player that has just begun the game. They are prompted with instructions: "Collect 50 balls!" in the upper right and have a starting score of 0, as seen in the upper left. The game is, at the same time, has spawned in balls. As the paddle is moved such that balls hit the top, those balls disappear. As other balls hit the bottom of the paddle, all the balls reset and are spawned in in a larger quantity. Finally, a total of 50 balls has hit the top of the paddle and thus the a congratulations message is shown and, for fun, all the balls are reset and an extra 1000 are spawned in.
2. The creation of the game was highly incremental and iterative. I begun the project with code from the ball acceleration lab and the collision detection lab, this was my base. Form here I incrementally added code. First the paddle, then the collision detection, then the splicing for the top of the paddle, then a reset for the bottom of the paddle, then finally a score and instructions. Through this process the creation of the game was also iterative. Between every addition the code had to be run and rerun, constantly checked for bugs and success. More specifically, the creation of the splice for the top of the paddle and the reset for the bottom of the paddle are perfect representations of an incremental and iterative process. As I built the processes which decided how to splice balls that hit the top of the paddle I begun by creating 2 paddles, the first of the two would, whenever a ball got very close to it, splice the ball. This was my first attempt and at I had to go through the iterative process of testing and debugging the program such that finally, it worked. The same thing was done for the bottom of the paddle and resetting the balls, piece by piece they were created. Sadly, I was not yet done. With this method I had a lot of code in the sketch file. For both the splice and reset functions I had to delete everything I had just done and try again in the ball file. Here, I - once again - incrementally and iteratively created a code that could splice and reset the balls. The progress for both done here was completely independent.
3. An example of an algorithm within my code is the ball file, and functions relating to how the ball operates.

```
function Ball(loc, vel, rad, col, sp){
  // Instance variables
  this.loc = loc;
  this.vel = vel;
this.rad = rad;
this.col = col;
this.sp = sp;
this.acc = createVector(0, .1);
//this function calls other functions
this.run = function(){
  this.checkEdges();
  this.update();
  this.render();
  this.checkPaddle();
}
//This function changes the location of the ball
//by adding speed to x and y
this.update = function(){
  this.loc.add(this.vel);
  this.vel.add(this.acc);
  this.loc.add(this.vel);
  this.loc.mag();
}
//checkEdges() reverses speed when the ball touches an edge
//keeps shit from going off the edge
this.checkEdges = function(){
  if(this.loc.x < 0) this.vel.x = -this.vel.x;
  if(this.loc.x > width) this.vel.x = -this.vel.x;
  if(this.loc.y < 0) this.vel.y = -this.vel.y;
  if(this.loc.y > height) this.vel.y = -this.vel.y;
}

//render() draws the ball at the new location
this.render = function(){
  fill(this.col);
  ellipse(this.loc.x, this.loc.y, rad, rad);
}

//checking when the ball hits the paddle
this.checkPaddle = function(){
  //takes location of center of ball - paddle y + 1/2(paddle's length)
  var distY = abs(this.loc.y - 560)
  //looking for if the ball is hitting the top of the bottom of the paddle
```

```
    if((distY < 10) && (this.loc.x > mouseX - 125) && (this.loc.x < mouseX + 125) &&
(this.vel.y > 0)){
      this.sp = 1
    }
    if((distY < 10) && (this.loc.x > mouseX - 125 ) && (this.loc.x < mouseX + 125) &&
(this.vel.y < 0)){
      this.sp = 2
    }
  }
}
```

The overarching algorithm here is the ball function. The ball function controls the movement of the balls, describes how the balls interacts with the paddle, and the ball overall. The individual algorithms within this larger one are the update, checkEdges, render, and checkPaddle function. The update function controls how the movements of the balls. checkEdges keeps the balls from going off the canvas and allows them to instead bounce off the edges. Render creates the ball itself. Finally, checkPaddle decides how the ball interacts with the paddle: if the ball(s) should be spliced or reset. The checkPaddle function is an example of the utilization of mathematical concepts. The function utilizes a parameter "sp". The name stands for splice and determines how the ball should act when in contact with the paddle. If sp == 3 then the ball is not touching the paddle, if sp == 2 then the ball has touched the bottom of the paddle and all the balls should reset, if sp == 1 then the ball should be spliced from the array and thus removed from the game. Sp begins as 3 and the checkPaddle algorithm decides if it should be redefined as 2 or 3. Sp is redefined as 2 or 3 based on math. Depending on distance from the paddle in y-coords, the x-coord location of the ball, and the velocity of the ball sp is refined.These are the functions of each algorithm independently. All together they create the ball algorithm that can create a ball which moves with acceleration, remains on the canvas, and interact with the paddle in the game. This algorithm is key to the entirety of the program. With it the game would not function. The balls created are what the player must collect in order to win. They are the objective within the game. Without any of these algorithms of the ball algorithm all together, there would be no game.

4. An example of abstraction is the creation of many balls from the single ball function.

```
function loadBalls(numBalls){
  for(var i = 0; i < numBalls; i++){
    //where the balls are spawned in
    var loc = createVector(random(100, 600), 20);
    var vel = createVector(random(-3, 3), random(-3, 3));
    var rad = 25
    var col = color(random(0, 255), random(0, 255), random(0, 255));
    var sp = 3
    var b = new Ball(loc, vel, rad, col, sp);
    //add balls to the array
    Balls.push(b);
  }
}
```

Code was written for the creation of a single ball, but it was utilized to create many. A single ball function was written but the program can, using the function, create as many as needed. Abstraction allows for the file to only contain one function and its function call to create as many balls as needed. An approach to the program without abstraction would be the creation of hundreds of functions, each creating its own singular ball. Thus, abstraction allows from extreme simplification.

5. □**:** algorithm
▮**:** abstraction

Ball.js

```
/*
** Ball Constructor Function
** Jakob Hachigian-Kreutzer
** 10/4/18
*/

□
//function creating balls, utilized through abstraction
function Ball(loc, vel, rad, col, sp){
  // Instance variables
  this.loc = loc;
  this.vel = vel;
  this.rad = rad;
  this.col = col;
  this.sp = sp;
  this.acc = createVector(0, .1);
  //this function calls other functions
  this.run = function(){
    this.checkEdges();
    this.update();
    this.render();
    this.checkPaddle();
  }
  //This function changes the location of the ball
  //by adding speed to x and y
  this.update = function(){
    this.loc.add(this.vel);
    this.vel.add(this.acc);
    this.loc.add(this.vel);
    this.loc.mag();
  }
  //checkEdges() reverses speed when the ball touches an edge
  //keeps shit from going off the edge
  this.checkEdges = function(){
    if(this.loc.x < 0) this.vel.x = -this.vel.x;
    if(this.loc.x > width) this.vel.x = -this.vel.x;
    if(this.loc.y < 0) this.vel.y = -this.vel.y;
    if(this.loc.y > height) this.vel.y = -this.vel.y;
  }

  //render() draws the ball at the new location
  this.render = function(){
    fill(this.col);
    ellipse(this.loc.x, this.loc.y, rad, rad);
  }

  //checking when the ball hits the paddle
  this.checkPaddle = function(){
    //takes location of center of ball - paddle y + 1/2(paddle's length)
    var distY = abs(this.loc.y - 560)
    //looking for if the ball is hitting the top of the bottom of the paddle
    if((distY < 10) && (this.loc.x > mouseX - 125) && (this.loc.x < mouseX + 125) &&
(this.vel.y > 0)){
      this.sp = 1
    }
    if((distY < 10) && (this.loc.x > mouseX - 125 ) && (this.loc.x < mouseX + 125) &&
(this.vel.y < 0)){
      this.sp = 2
    }
  }
}
```

paddle.js

```
/*
** Paddle
** Jakob Hachigian-Kreutzer
**
*/

function Paddle(loc, vel, width, length, col){
  // Instance variables
  this.loc = loc;
  this.vel = vel;
  this.w = width;
  this.l = length;
  this.col = col;
  //this function calls other functions
  this.run = function(){
    this.checkEdges();
    this.update();
    this.render();
  }
//lerp -- paddle follows mouse
  this.update = function(){
    //make paddle lerp to middle of rectangle instead of corner
    paddleLength = width/2
    this.loc.x = lerp(this.loc.x, mouseX-paddleLength, .15)
  }
  //checkEdges() reverses speed when the rectangle touches an edge
  this.checkEdges = function(){
    if(this.loc.x < 0) this.vel.x = -this.vel.x;
    if(this.loc.x > width) this.vel.x = -this.vel.x;
    if(this.loc.y < 0) this.vel.y = -this.vel.y;
    if(this.loc.y > height) this.vel.y = -this.vel.y;
  }

  //render() draws the paddle at the new location
  this.render = function(){
    fill(this.col);
    rect(this.loc.x, this.loc.y, this.w, this.l);
  }
}
```

sketch.js

```
//Global variables
var Balls = [];
var paddle;
var score = 0;
//setup canvas
function setup(){
  var cnv = createCanvas(800, 800);
  cnv.position((windowWidth-width)/2, 30);
  background(20, 20, 20);
  //# of balls loaded
  numBalls = 20;
  loadBalls(numBalls);
  //
  //creating the lerping paddle
  //
  var loc = createVector(400, 550)
  var vel = createVector(0, 0);
  var width = 250;
  var length = 20;
  var col = color(random(0, 255), random(0, 255), random(0, 255))
  paddle = new Paddle(loc, vel, width, length, col);
}
//
//load balls
//

function loadBalls(numBalls){
  for(var i = 0; i < numBalls; i++){
    //where the balls are spawned in
    var loc = createVector(random(100, 600), 20);
    var vel = createVector(random(-3, 3), random(-3, 3));
    var rad = 25
    var col = color(random(0, 255), random(0, 255), random(0, 255));
    var sp = 3
    var b = new Ball(loc, vel, rad, col, sp);
    //add balls to the array
    Balls.push(b);
  }
}


//draw balls + mouse controlled paddle
function draw(){
  background(20, 20, 20, 6000);
  //control the score
  textSize(32);
  fill(random(0,255), random(0,255), random(0,255));
  text("score = " + score, 50, 50);
  //instructions
  if(score < 50){
    fill(random(0,255), random(0,255), random(0,255));
    text("Collect 50 Balls!", 500, 50);
  }
  //if instructions are completed
  if(score >= 50 && score <= 100){
    fill(random(0,255), random(0,255), random(0,255));
    textSize(120);
    text("You Win!", 150, 400);
  }
  //if instructions are completed
  if(score == 50){
    score = score + 1
    //prize
    var numBalls = 1000;
    Balls = []
    loadBalls(numBalls);
    for(var i = 0; i < numBalls; i++){
      Balls[i].run();
    }
  }
  //get rid of outlines
  noStroke();
  paddle.run();
  for(var i = 0; i < Balls.length; i++){
    Balls[i].run();
    var aBalls = Balls[i];
    //splice the balls if they have touched the top of the paddle
    if(aBalls.sp == 1){
      Balls.splice(i,1);
      //adds to score for every ball
      score = score + 1;
    }
    //"reset" the balls if a ball hits the buttom
    if(aBalls.sp == 2){
      //decides how many balls are going to be in the next "reset"
      var numBalls = Balls.length + 25;
      //resets the array (deleted all the current balls)
      Balls = []
      loadBalls(numBalls)
      for(var i = 0; i < Balls.length; i++){
        Balls[i].run();
      }
    }
  }
}
```