Bouncy Ball Coding Tutorials

Wojciech Golaszewski Cosmin Vladianu David Al Mjali Adris Khan

Tutorial 3 – Maze Ball Game

Coding Curriculum Group Project

UCL ENGINEERING
Change the world

UCL

Maze Ball

The third and last part of this tutorial will be about learning how to create your own “maze” which you will be able to play yourself. For this purpose, we will be using the supporting material from the previous two parts of the tutorial, which comprise of making the ball react to user input (pressing the keys “W”, “A”, “D”), bounce off the environment boundaries and detect object collision. Additionally, we will show you how to modify the collision detection so that the ball can bounce off objects. Last, but not least, we will also teach you how to make an object move around, so that your maze can be more interesting.

Maze Ball

1/5 Movement of The Ball

First steps 1/2

We will “borrow” the main structure of the previous two parts of the tutorial. As shown on the right, for creating a ball that moves according to the user input, the structure of the code remains roughly the same. However, this will have to be modified accordingly as more features are added.

```
var ball;

function setup()
{
  createCanvas(960, 540);
  fill(255);
  noStroke();
  Initialise();
}

function Initialise()
{
  ball = new Ball ();
}

function draw ()
{
  background(0);

  ball.move();

  ball.show();
}
```

First steps 2/2

The variables we use for showing the ball and making it move are roughly the same. However, we have decided to change the gravity and yspeed values, and added “friction”, which will determine how the ball speed changes when it is touching an object, simulating real life friction. We will explain more of how friction works shortly.

```
function Ball () {
  this.r = 15;
  this.x = width/10;
  this.y = height-30;

  var yspeed = 6;
  var xspeed = 0;
  var gravity = 0.2;
  var friction = 0.85;
  this.dw = 0;

  this.show = function() {
      fill(255);
       ellipse(this.x, height-this.y, 2*this.r, 2*this.r);
  }
```

Maze Ball

2/5 Block Collisions and Stop Condition

Environment collision 1/2

Remember the moving function in the first part of the tutorial, Bouncing Ball? We use the same idea here, except that we add a limit for the horizontal speed, increasing it by 1 for each time you press "A" or "D" and capping it at 3. This is done in order to be able to properly control the ball movement. Likewise, the vertical speed doesn't get increased either, but it is just set at a specific value, in this case 8. Moreover, we added a variable "dw", which ensures that you can only jump when you are touching the ground, simulating real life jumping.

Consequently, you can only make one jump at a time. Namely, when the ball touches the ground, dw gets set to 0, so the next time "W" is pressed, if the ball is touching the ground(this.dw==0), it "jumps". Afterwards this.dw becomes 1, as we have to check again if the ball is touching the ground.

```
this.move = function() {

      yspeed -= gravity;

      if (keyIsDown("W".charCodeAt(0)) || mouseIsPressed)
      {
          if (this.dw == 0) yspeed = 8;
      }
      this.dw=1;

      if (keyIsDown("D".charCodeAt(0)))
      {
          xspeed+=1;
          if(xspeed>3)
            xspeed=3;
      }

      if (keyIsDown("A".charCodeAt(0)))
      {
        xspeed-=1;
        if(xspeed<-3) xspeed=-3;
      }
```

Environment collision 2/2

As previously, we need to make sure our ball interacts with the environment's boundaries properly. As such, when the ball hits the left or the right "walls", it bounces off with a reduced and opposite speed (xspeed*=-coef). Similarly, when the ball touches the "roof" of our environment, it will start falling down (yspeed= 0; this.y= height - this.r;). However, when the ball touches the floor, we mark it (this.dw=0) in order to be able to tell when the ball can jump by pressing "W". The ball's interaction with the floor is roughly the same as before, but we added friction. For each moment the ball is touching the ground, its horizontal speed gets reduced (xspeed*=friction), emulating real life friction.

```
    var coef = 0.6;
    if ( this.x-this.r+xspeed < 0 || this.x+this.r+xspeed > width)
          xspeed *= -coef;

    coef = 0.7;

    if ( this.y-this.r+yspeed < 0 )
      this.dw=0;

    if (this.dw==0)
    {
      yspeed *= -coef;
      xspeed *= friction;
    }

    if ( this.y+this.r+yspeed > height)
    { yspeed= 0; this.y= height - this.r; }

    this.y += yspeed;
    this.x += xspeed;
}
```

Challenge!
As you learnt how to control the ball around, modify the code so you can have double jumps!

Block movement and colour

Remember how in the flappy ball tutorial the game stopped when we touched a block? We will use the same code, but will add a couple of features. Firstly, as we want our blocks to be able to move as well, we need to add a function for this (this.moveV = function(y, yF)), which makes a block move on the vertical axis in the range of y and yF, which are given as parameters to the function. When it hits one of the range limits, the bock starts moving in the opposite direction(yspeed*= -1). Furthermore, we can modify the "show" function so that you can make blocks have a specific colour. Thus, if we want the block to be a certain colour, we have to give the function some parameters otherwise it will be green by default. If a!== "undefined", it means the function was called with parameters and thus the block has to have a specific colour.

```
function Block(x, y , wid , hei){

    this.x = x;
    this.y = y;
    this.wid = wid;
    this.hei = hei;

    var yspeed = 5;

    this.moveV = function(y, yF) // the area of movement
    {
      if(this.y + yspeed < y || this.y + yspeed > yF)
      {
        yspeed *= -1;
      }
      this.y += yspeed;
    }

    this.show = function(a,b,c)
    {
        if(typeof a !== "undefined")
        fill(a,b,c);
        else
          fill(23, 145, 23);  //light green
        rect(this.x, height - this.y, this.wid, -this.hei);
    }

    this.checkcollision = function(){
      return rect_coll(this.x, this.y, this.wid, this.hei);
    }
}
```

Block Collisions and Stop Condition

Stop condition

As we want the game to stop when you touch the ball, we need a variable which tells the ball to move (or not). We declare a variable “lost”, which we set as false (lost=false;), meaning the game has not ended yet. We will store the “blocks”, or obstacles, in an array(blocks = [];). Similarly to the Flappy Ball tutorial, we iterate through each block and check if it collides with the ball. If it does, than the game is lost, the variable that manages this is changed (lost=true) and as a consequence, the ball will not move anymore(if(!lost) will be false, as we lost).

Challenge!
Add moving obstacles!
Change color of obstacles!

```
var ball;
var lost;

function Initialise()
{
  ball = new Ball ();
  blocks = [];
  lost=false;

  blocks.push(new Block(300,100,100,50));

}

function draw ()
{
  background(0); //blue

if(!lost)
    ball.move();

for(let i=0;i<blocks.length;i++)
  {
    blocks[i].show();
    if(blocks[i].checkcollision())
      lost=true;
  }

  ball.show();
}
```

“getter” methods 1/2

We are now going to introduce you to one handy principle of objective oriented programming. While in the flappy ball tutorial “xspeed” and “yspeed” were declared as “this.xspeed” and “this.yspeed”, this is not actually recommended while making bigger applications, because this would mean you can modify them from anywhere in your program. Thus, we will introduce 2 functions which will provide us the speed values which we need. However, we will not make functions that allow us to change them, as we do not need this (now). While this might mean code that looks more complicated, it makes it harder to modify variables without explicitly wanting to do so, which means it is less likely that we you have bugs in your code. Those are called “getter” methods (or functions), which are generally used with “setter” methods, which modify the variables values.

```
var yspeed = 6;
var xspeed = 0;
var gravity = 0.2;
var friction = 0.85;
this.dw = 0;

this.yspeed = function()
{
  return yspeed;
}

this.xspeed = function()
{
  return xspeed;
}
```

"getter" methods 2/2

In the function "rect_coll" there will be a couple of small changes, namely instead of "ball.xspeed" we will need "ball.xspeed()" and the same for "yspeed". Yes, it makes code look less appalling and maybe slightly confusing, but it helps you avoid accidentally creating bugs in your code. Thus, writing more code in order to avoid mistakes is a worthy sacrifice. Apart from this replacement, the code in this function remains the same as in the second part of our tutorial. Do not forget about the "check_dist" function!

```
function rect_coll(x, y, wid, hei)
{
    //if crossed horizontal wall
    if (ball.x+ball.xspeed() > x && ball.x+ball.xspeed() < x + wid)
        if (ball.y+ball.yspeed()+ball.r>max(y,y+hei) && ball.y+ball.yspeed()-ball.r<max(y,y+hei))
            {
              return true;
            }
        else if(ball.y+ball.yspeed()+ball.r>min(y,y+hei) && ball.y+ball.yspeed()-ball.r<min(y,y+hei))
        {
          return true;
        }
```

Maze Ball

3/5 Obstacles and Game Restart

Maze Ball

Adding restart

Okay, so the game ends, but we do not want to have to refresh our browser each time we start again. Consequently, we added the option of restarting the game, which is also present in the Flappy Ball tutorial. If the ball touches an obstacle and you press "R", the game will restart, by reinitialising the environment :

```
if (lost == true && keyIsDown("R".charCodeAt(0))) Initialise();
```

We also added a new function, which will tell you when you lose and what to do to restart it (showFinal();). We also display the obstacles with red, to denote the fact that you should not touch them.

```
function draw ()
{
  background(0); //blue

  if(!lost)
    ball.move();
  else
    showFinal();

    for(let i = 0; i<obstacles.length; i++)
    {
      obstacles[i].show(200,0,0); //red
      if(obstacles[i].checkcollision())
      {
        lost = true;
      }
    }

  ball.show();

  if (lost == true && keyIsDown("R".charCodeAt(0)))
       Initialise();

}
```

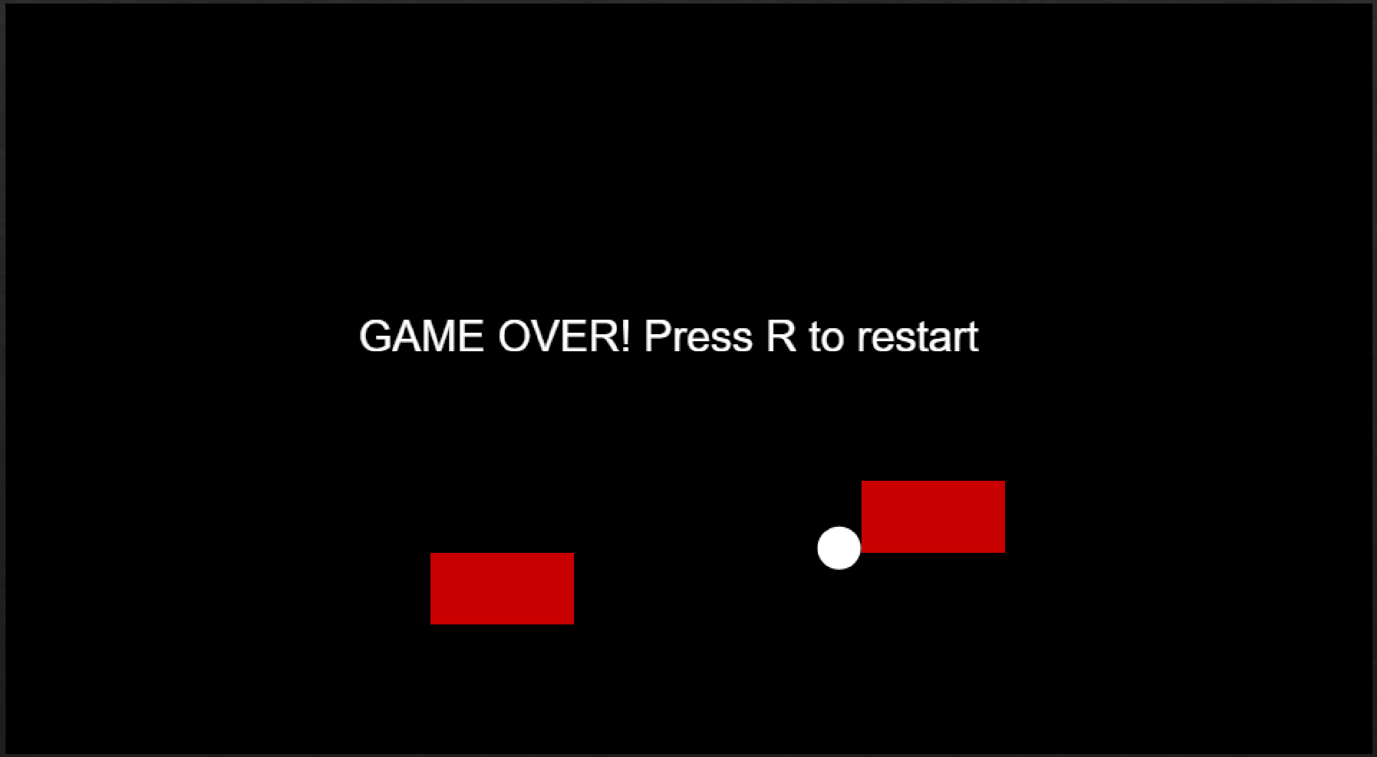
Displaying a message

The function is fairly straight forward, but it presents another useful feature of pi5.js, namely the "text()" function. You firstly have to select the text colour and size, we chose white and 30 respectively, and after that you have to use this format: text("Message to show", x, y), where x and y are the coordinates where the message will be shown on the screen.

```
function showFinal()
{
  fill(255,255,255);
  textSize(30);
  text("GAME OVER! Press R to restart",250,250);
}
```

Try it yourself!
Change the text color/ size!
Move the text around the screen!

How your game should look by now



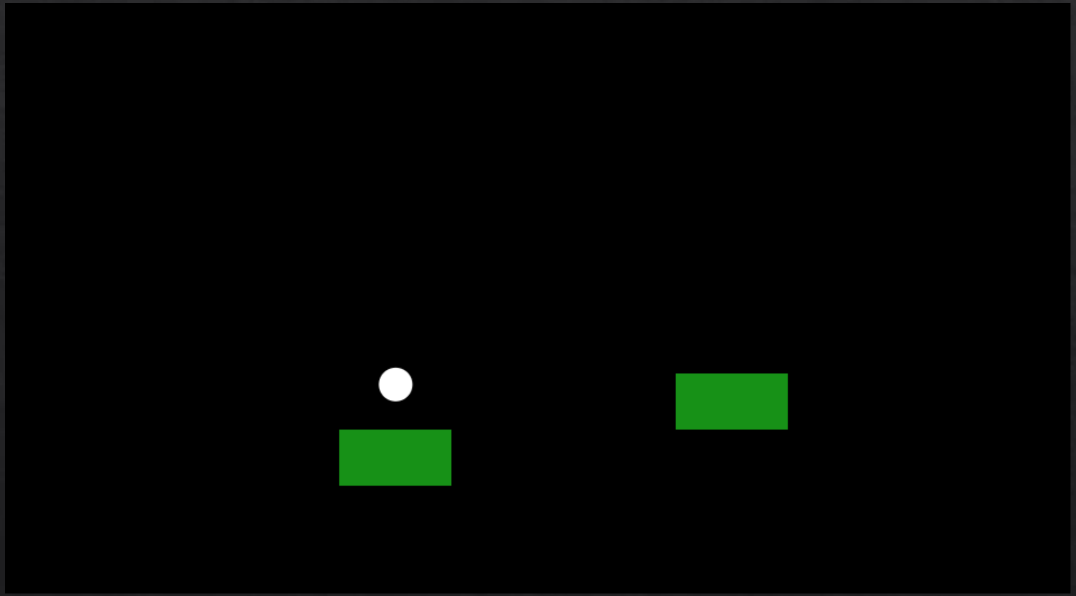
GAME OVER! Press R to restart

Maze Ball

4/5 Making the Ball Bounce

Block Bouncing

Okay, so now we can lose a game when we touch a block, but how about we make the ball bounce off a block as well? Remember how we check block collision? For each block, if we touch it, the game ends. But now, if the ball touches the block, we need to figure out how it touches it, and make it react properly. Thus, we will enable the ball to be able to bounce off a block by creating functions that allow it to do so.



Bouncing on a Block

Let’s start with the function that tells the ball what to do when it bounces on top of a block. Naturally, it will be very similar with bouncing off the ground. Thus, “d” takes the place of “this.dw” in order to be able to bounce only once, and only when the ball touches the block. You may notice that we make “yspeed” equal to a negative value here, as opposed to a positive one in the “move” function. We will leave it to you as homework to figure out why this happens. HINT:

```
if ( this.y-this.r+yspeed < 0 )
       this.dw=0;

     if (this.dw==0)
     {
       yspeed *= -coef;
       xspeed *= friction;
     }
```

```
this.bounceUp = function(y)
 {
   yspeed -= gravity;
   var d=0;
   if (keyIsDown("W".charCodeAt(0)) || mouseIsPressed)
   {
       if(d==0) yspeed=-9;
   }
   d=1;

   coef = 0.7;
   if(this.y-this.r+yspeed<y)
   {
     yspeed *= -coef;
     xspeed *= friction;
     d=0;
   }

   this.y += yspeed;
   this.x += xspeed;
 }
```

Bouncing Beneath a Block

Next comes the one that tells the ball what to do when you bounce into a block from beneath it, which is exactly the same as the previous one, but without the jumping part, and with the inverted sign we told you to think bout (yspeed *= -coef). You may have (hopefully) noticed that both of them have a parameter, "y". This acts as the ground level (in this case roof), as bouncing down on or up to a block is no different from hitting the floor or the upper limit of the environment. This is like saying that the upper side of the block is considered as being the floor and the lower side, the "roof". Furthermore, adding the speed of the ball into the equation increases accuracy, as it basically means "Will the ball hit the block the next time it moves?" instead of "Does the ball hit the block now?".

```
this.bounceDown = function(y)
 {
   yspeed -= gravity;

   coef = 0.7;
   if(this.y+this.r+yspeed>y)
   {
     yspeed *= -coef;
     xspeed *= friction;
   }

   this.y += yspeed;
   this.x += xspeed;
 }
```

Corner Collision 1/2

Last, but not least, there comes the collision that needs most effort to comprehend. We really encourage you to go through the calculus yourself, as it is a fun and useful exercise. In order to figure out how the ball behaves in the event of a corner collision, we need to first find the angle of the collision, which we store in the “angle” variable. Pi5.js has a math library which provides us with the arctan function, which we use in this sense:

```
function get_angle(x,y)
{
  return Math.atan2(y,x);
}
```

```
this.bounceCorner = function(xdif,ydif)
 {
   angle=get_angle(xdif,ydif);
   cosa=Math.cos(angle);
   sina=Math.sin(angle);

   //get the y and x speeds in the newly defined inclined plane
   v1=xspeed*cosa-yspeed*sina; //the OX axis in the inclined plane
   v2=xspeed*sina+yspeed*cosa; //the OY axis in the inclined plane

   //get the resulting speeds in the original (normal) plane
   xspeed=v1*cosa-v2*sina;
   yspeed=v2*cosa+v1*sina;

   coef=0.9;
   xspeed*=coef;
   yspeed*=coef;
   this.y += yspeed;
   this.x += xspeed;
 }
```

Corner Collision 2/2

Next, we recommend to either go through it yourselves! The function takes 2 parameters, which represent the x and y axis difference between the centre of our ball and the edge with which it collides. After computing the collision angle, we also need the sine and cosine of this angle in order to be able to translate “xspeed” and “yspeed” in a plane rotated to the left by that angle, which are saved into two variables (v1 and v2). Further on, those composed values will have to be translated back to the original speeds, which is the same as translating “v1” and “v2” in a plane rotated by “angle” to the right. Those values have to be lowered to depict a collision properly, and are thus multiplied by a coefficient (preferably lower than 1, we chose 0.9).

```
this.bounceCorner = function(xdif,ydif)
 {
   angle=get_angle(xdif,ydif);
   cosa=Math.cos(angle);
   sina=Math.sin(angle);

   //get the y and x speeds in the newly defined inclined plane
   v1=xspeed*cosa-yspeed*sina; //the OX axis in the inclined plane
   v2=xspeed*sina+yspeed*cosa; //the OY axis in the inclined plane

   //get the resulting speeds in the original (normal) plane
   xspeed=v1*cosa-v2*sina;
   yspeed=v2*cosa+v1*sina;

   coef=0.9;
   xspeed*=coef;
   yspeed*=coef;
   this.y += yspeed;
   this.x += xspeed;
 }
```

Modifying Ball Position

You may ask yourselves, why do we have to modify the position of the ball each time it bounces off a block, isn't it enough to just let it be in the "move" function? Our answer is, try doing that, and see how it goes. Programming is a lot of trial and error in order to find the optimal solution.

```
this.y += yspeed;
this.x += xspeed;
```

Challenge!
Is our corner collision working properly?
Improve it!

Collision detection

Now that we have added ways of bouncing the ball, all that is left to do is improve the collision system so that instead of checking if there is a collision, we emulate the actual collision! But keep in mind that the final goal is to bounce off some of the blocks, and avoid touching some of the other blocks. So how do we modify our code so that you can do both, depending on which want you want to accomplish?

Well, if you want the ball to bounce, you need to tell it to bounce (and how to bounce) before you finish the function call, hence it needs to bounce before the “return”s in our “rect_coll” function, or otherwise it will not get to bounce at all!

Maze Ball

Vertical Collision

If the ball bounces on top of the block, then the top of the block will momentarily become the floor, from the ball's point of view. Thus, we will need to send the ball the coordinates of this new floor, which will be the block's y coordinate + it's height: ball.bounceUp(y+hei).

A similar idea goes for the bouncing beneath the block, only that y will be the coordinate of the roof for the ball: ball.bounceDown(y).

```
function rect_coll(x, y, wid, hei)
{
    //if crossed horizontal wall
    if (ball.x+ball.xspeed() > x && ball.x+ball.xspeed() < x + wid)
        if (ball.y+ball.yspeed()+ball.r>max(y,y+hei) && ball.y+ball.yspeed()-ball.r<max(y,y+hei))
            {
              ball.bounceUp(y+hei); //does collision
              return true;
            }
        else if(ball.y+ball.yspeed()+ball.r>min(y,y+hei) && ball.y+ball.yspeed()-ball.r<min(y,y+hei))
        {
          ball.bounceDown(y); //does collision
          return true;
        }
```

Lateral Collision

If we thing about lateral collision, the replacement will be roughly the same, except for the fact that we do not need to parse the function any parameters at all.

```
if (ball.y+ball.yspeed()>min(y,y+hei) && ball.y+ball.yspeed()<max(y,y+hei))
        if (ball.x+ball.xspeed()+ball.r >= x && ball.x+ball.xspeed()-ball.r <= x )
            {
              ball.bounceLat(); //does collision
              return true;
            }
       else if (ball.x+ball.xspeed()+ball.r >= x + wid && ball.x+ball.xspeed()-ball.r <= x + wid )
             {
               ball.bounceLat(); //does collision
               return true;
             }
```

Corner Collision

Last, but not least, we also need to modify the parts of the code that check corner collision. The parameters which will be parsed to the collision function are the same ones that are sent to the check_dist function to check if collision occurs. If none of these collisions happened, then nothing was returned, so we need to return false, so we know that no collision occurred.

```
if (check_dist(ball.x-x+ball.xspeed(), ball.y-y+ball.yspeed(), ball.r))
   {
     ball.bounceCorner((ball.x-x+ball.xspeed()),ball.y-y+ball.yspeed());
     return true;
   }
   if (check_dist((ball.x-x-wid+ball.xspeed()), ball.y-y+ball.yspeed(), ball.r))
   {
     ball.bounceCorner(ball.x-x-wid+ball.xspeed(),ball.y-y+ball.yspeed());
     return true;
   }
   if (check_dist(ball.x-x+ball.xspeed(), ball.y-y-hei+ball.yspeed(), ball.r))
   {
     ball.bounceCorner((ball.x-x+ball.xspeed()),ball.y-y-hei+ball.yspeed());
     return true;
   }
   if (check_dist(ball.x-x-wid+ball.xspeed(), ball.y-y-hei+ball.yspeed(), ball.r))
   {
     ball.bounceCorner((ball.x-x-wid+ball.xspeed()),ball.y-y-hei+ball.yspeed());
     return true;
   }

   //wheeee!
   return false;
```

Final collision steps

Now that we are able to stop the game when we touch one particular block, but bounce on another block, let's see how you manage doing these 2 different tasks on two different sets of blocks.
All that you have to do is to put the piece of code only detects collision (on a set of obstacles which you should not touch while you are playing), together with a slightly modified version of it, that instead of checking the value of the collision, it just does the collision. This theoretically means that the ball would bounce off an obstacle as well, but we do not really mind that, as the game ends when this occurs anyway.

```
function draw ()
{
  background(0); //blue

if(!lost)
    ball.move();

      ball.show();

 for(let i = 0; i<obstacles.length; i++)
    {
      obstacles[i].show(200,0,0); //red
      if(obstacles[i].checkcollision())
        lost = true;

    }
    for(let i = 0; i<blocks.length; i++)
    {
      blocks[i].show();
      blocks[i].checkcollision();

    }

  if (lost == true && keyIsDown("R".charCodeAt(0))) Initialise();
}
```

Maze Ball

5/5 Final – Full Maze

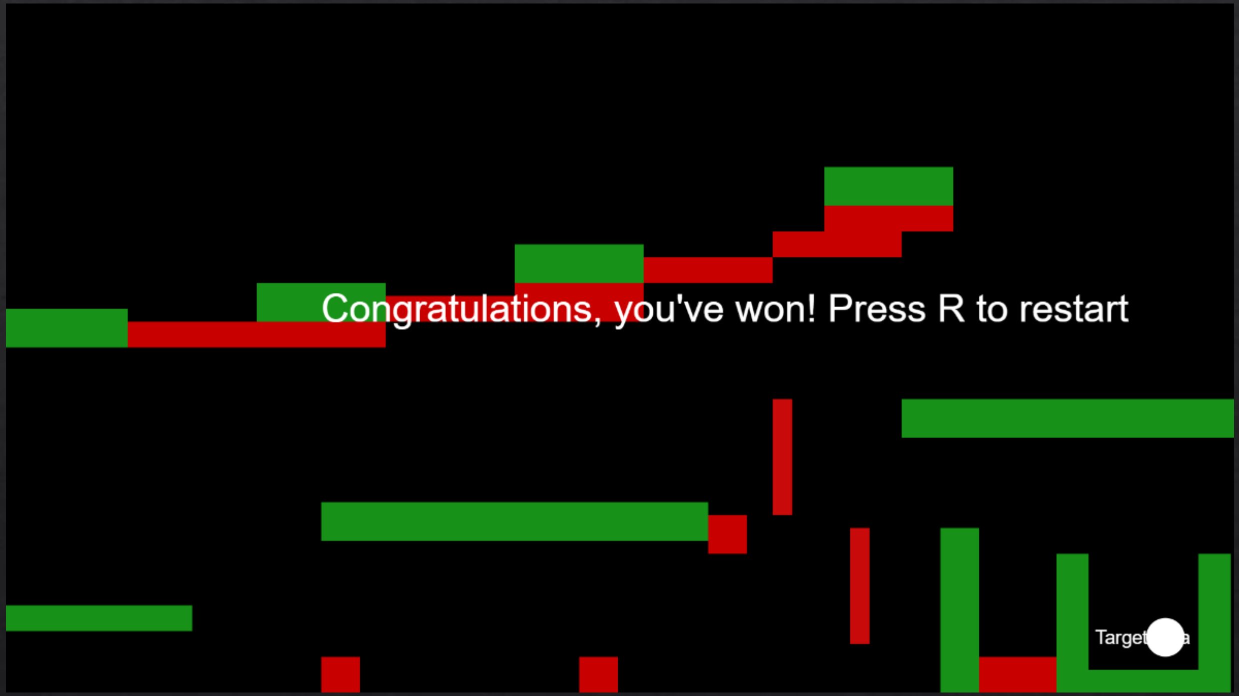
III

Maze Ball

Final – Full Maze

5/5

Challenge!
Do your own game!
Find a way of including a win condition!



Congratulations, you've won! Press R to restart