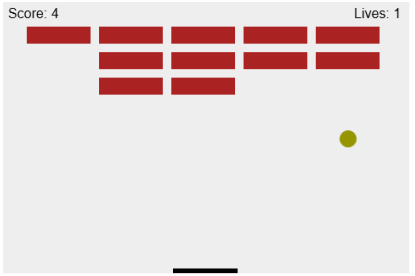
HTML5/Javascript Game Development

The Game

A simplified version of Breakout
https://en.wikipedia.org/wiki/Breakout_(video_game)



What you need

- Basic HTML5/CSS/Javascript knowledge
- Open mind, eager to learn

What you will learn

- Build a simple yet full-fledged game in Javascript
- Essential elements in any game, such as animation, collision detection, building monsters (bricks), scoring, lives, winning/losing conditions
- Advanced Javascript concepts such as event handling, JSON, anonymous function

Documentations

- W3School - HTML, CSS, Javascript searchable documentation at https://www.w3schools.com/
- MDN - Mozilla Developer Network searchable documentation on Web technology at https://developer.mozilla.org/en-US/docs/Web

Directory structure

Use the following structure to keep things neat and allow for future expansion

```
my_game_dir
 |
 |-- css
 |    |
 |    |-- main.css
 |
 |-- js
 |    |
 |    |-- main.js
 |
 |-- index.html
```

1. Draw objects

The HTML5 Canvas

`index.html`

```
 1   <html>
 2   <head>
 3       <meta charset="utf-8" />
 4       <title>My HTML5 Canvas</title>
 5       <link rel="stylesheet" type="text/css" href="css/main.css">
 6   </head>
 7   <body>
 8   <!-- The canvas element in which the game is drawn -->
 9   <canvas id="myCanvas" width="480" height="320"></canvas>
10   <!-- The script must come after the Canvas declaration -->
11   <script src="js/main.js"></script>
12   </body>
13   </html>
```

`css/main.css`

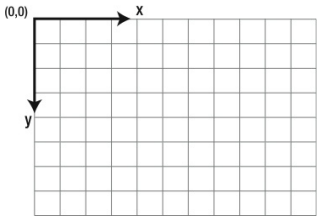
```
1   * { padding: 0; margin : 0; }
2   canvas { background: #eee; display: block; margin: 0 auto; }
```

Getting a handle in Javascript

`js/main.js`

```
1   // Get a handle to the 2D context of the Canvas element
2   let canvas = document.getElementById("myCanvas");
3   let context = canvas.getContext("2d");
```

Canvas Coordinate System



Drawing various shapes

```
1     // Draw a line
2     context.moveTo(200, 200);
3     context.lineTo(300, 300);
4     context.strokeStyle = "rgba(0, 0, 255, 0.5)";
5     context.stroke();
6
7     // Draw a rectangle filled with color
8     context.beginPath();
9     context.rect(20, 40, 50, 50);
10    context.fillStyle = "#FF0000";
11    context.fill();
12    context.closePath();
13
14    // Draw a circle
15    context.beginPath();
16    context.arc(240, 160, 20, 0, Math.PI*2);
17    context.fillStyle = "green";
18    context.fill();
19    context.closePath();
```

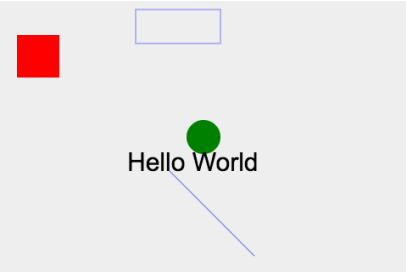
Drawing with different style

```
1     // Draw a rectangle with border only
2     context.beginPath();
3     context.rect(160, 10, 100, 40);
4     context.strokeStyle = "rgba(0, 0, 255, 0.5)";
5     context.stroke();
6     context.closePath();
```

Drawing text

```
1     // Write text
2     context.font = "30px Arial";
3     context.textAlign = "left"
4     context.fillStyle = "black";
5     context.fillText("Hello World", 150, 200);
```

Checkpoint 1



Exercise 1

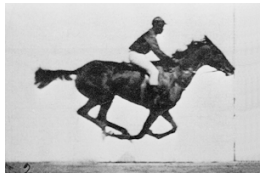
Functions allow us to reuse code (and type less!). Create the following functions using the signatures provided, and use them.

```
1   function drawRect(ctx, x, y, width, height, fillStyle, strokeStyle)
2   {
3       ...
4   }
5
6   function drawCircle(ctx, x, y, radius, fillStyle, strokeStyle)
7   {
8       ...
9   }
10
11  function writeText(ctx, text, x, y, alignment, font, fillStyle)
12  {
13      ...
14  }
```

2. Move objects

Draw loop

The perception of motion is achieved by exploiting the Phi Phenomenon.



In game dev, we use a draw loop:

```
1   function draw(param1, param2, ...)
2   {
3       ...
4   }
5
6   window.setInterval(draw, 10, param1, param2, ...);
```

`window.setInterval()` documentation

Moving the ball - 1

What is the ball's starting location?

```
1      // Ball's initial location
2      let xBall = canvas.width / 2;
3      let yBall = canvas.height - 30;
```

What is the ball's size and color?

```
1     const COLOR_BALL_FILL_STYLE = "#959500";
2     const SIZE_BALL_RADIUS      = 10;
```

Draw the ball

```
1    drawCircle(context, xBall, yBall,
2                SIZE_BALL_RADIUS, COLOR_BALL_FILL_STYLE, "");
```

Moving the ball - 2

```
1    /**
2     * Main draw loop of the game
3     * @param {object} ctx The 2D context of a Canvas
4     */
5    function main(ctx)
6    {
7        drawCircle(ctx, xBall, yBall,
8                   SIZE_BALL_RADIUS, COLOR_BALL_FILL_STYLE, "");
9    }
10
11   window.setInterval(main, 10, context);
```

Moving the ball - 3

Update the ball's position by adding small offsets to its x- and y-coordinates. In physics, they are called velocities.

```
1       // Ball's initial velocities
2       let xBallVelocity =  2;
3       let yBallVelocity = -2;
4
5       /**
6        * Update the ball's current status
7        */
8       function updateBall()
9       {
10          xBall += xBallVelocity;
11          yBall += yBallVelocity;
12      }
```

Quiz: what is the angle of movement implied by the initial velocities?

Moving the ball - 4 *What's Wrong?*

```
1    function main(ctx)
2    {
3        drawCircle(ctx, xBall, yBall,
4                   SIZE_BALL_RADIUS, COLOR_BALL_FILL_STYLE, "");
5        updateBall();
6    }
7
8    window.setInterval(main, 10, context);
```

Moving the ball - 5

Clear the canvas!

```
1   function main(ctx)
2   {
3       ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
4       drawCircle(ctx, xBall, yBall,
5                  SIZE_BALL_RADIUS, COLOR_BALL_FILL_STYLE, "");
6       updateBall();
7   }
```

Bouncing off walls - detection

Against right wall

```
1     xBall + xVelocity >= ctx.canvas.width
```

Against left wall

```
1       xBall + xVelocity <= 0
```

Against top wall

```
1       yBall + yVelocity <= 0
```

Against bottom wall

```
1     yBall + yVelocity >= ctx.canvas.height
```

Bouncing off walls - reflection

Against right and left walls

```
1       xVelocity = -xVelocity;
```

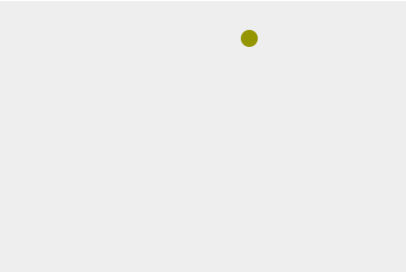
Against top and bottom walls

```
1       yVelocity = -yVelocity;
```

Bouncing off walls - putting it together

```
 1  /**
 2   * Update the ball's current status
 3   * @param {object} ctx  The 2D context of a Canvas
 4   */
 5  function updateBall(ctx)
 6  {
 7      // change direction of velocity when hitting the wall
 8      if ( xBall + xBallVelocity > (ctx.canvas.width - SIZE_BALL_RADIUS) ||
 9           xBall + xBallVelocity < SIZE_BALL_RADIUS )
10      {
11          xBallVelocity = -xBallVelocity;
12      }
13      if ( yBall + yBallVelocity > (ctx.canvas.height - SIZE_BALL_RADIUS) ||
14           yBall + yBallVelocity < SIZE_BALL_RADIUS )
15      {
16          yBallVelocity = -yBallVelocity;
17      }
18
19      // update ball's location
20      xBall += xBallVelocity;
21      yBall += yBallVelocity;
22  }
```

Checkpoint 2



Exercise 2

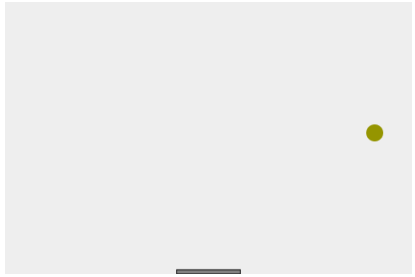
Make the ball change color when it bounces off a wall.

3. Process user input

Drawing the paddle - 1

```
1    // Paddle color and sizes
2    const COLOR_PADDLE_FILL_STYLE = "#808080"
3    const SIZE_PADDLE_HEIGHT      =  5;
4    const SIZE_PADDLE_WIDTH       = 75;
5
6    // Paddle's initial location
7    let xPaddle = (canvas.width - SIZE_PADDLE_WIDTH)/2;
8    let yPaddle = canvas.height - SIZE_PADDLE_HEIGHT;
9
10   function main(ctx)
11   {
12       ...
13       // draw the paddle
14       drawRect(ctx, xPaddle, yPaddle,
15               SIZE_PADDLE_WIDTH, SIZE_PADDLE_HEIGHT,
16               COLOR_PADDLE_FILL_STYLE, "");
17   }
```

Drawing the paddle - 2



Registering event handlers

`document.addEventlistener()` documentation

```
1   // register keyboard and mouse event handlers
2   document.addEventListener("keydown",   keyDownHandler,   false);
3   document.addEventListener("keyup",     keyUpHandler,     false);
4   document.addEventListener("mousemove", mouseMoveHandler, false);
```

Responding to key presses - 1

```
1       let isKeyRightPressed = false;
2       let isKeyLeftPressed  = false;
3
4       /**
5        * Handles a KeyDown event
6        * @param {object} evt KeyDown event
7        */
8       function keyDownHandler(evt)
9       {
10          switch(evt.key)
11          {
12              case "ArrowRight":
13              case "Right":
14                  isKeyRightPressed = true;
15                  break;
16
17              case "ArrowLeft":
18              case "Left":
19                  isKeyLeftPressed = true;
20                  break;
21          }
22      }
```

Exercise 3

Write the `keyUpHandler()` function.

Hint: mimic `keyDownHandler()`.

Responding to key presses - 2

```
1      /**
2       * Handles a KeyUp event
3       * @param {object} evt KeyUp event
4       */
5      function keyUpHandler(evt)
6      {
7          switch(evt.key)
8          {
9              case "ArrowRight":
10             case "Right":
11                 isKeyRightPressed = false;
12                 break;
13
14             case "ArrowLeft":
15             case "Left":
16                 isKeyLeftPressed = false;
17                 break;
18         }
19     }
```

List of Key Values here

Moving the paddle with the keyboard

```
1    // how much the paddle moves when key is depressed
2    const SIZE_PADDLE_NUDGE = 7;
3
4    /**
5     * Update the paddle's current status
6     */
7    function updatePaddle()
8    {
9        if ( isKeyRightPressed )
10       {
11           xPaddle += SIZE_PADDLE_NUDGE;
12       }
13       else if ( isKeyLeftPressed )
14       {
15           xPaddle -= SIZE_PADDLE_NUDGE;
16       }
17   }
```

Keeping it within the canvas

```
1    function updatePaddle(ctx)
2    {
3        if ( isKeyRightPressed &&
4            xPaddle < ctx.canvas.width - SIZE_PADDLE_WIDTH )
5        {
6            xPaddle += Math.abs(xPaddleNudge);
7        }
8        else if ( isKeyLeftPressed &&
9                  xPaddle > 0 )
10       {
11           xPaddle -= Math.abs(xPaddleNudge);
12       }
13   }
14
15   function main(ctx)
16   {
17       ...
18       updateBall(ctx);
19       updatePaddle(ctx);
20   }
```

Controlling it with the mouse - 1

Key issue

How to obtain the relative position of the pointer *within* the canvas?

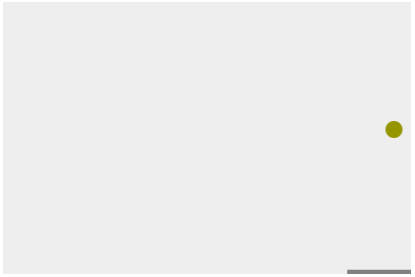
Solution

```
1    let relativeX = MOUSE_EVENT.clientX - canvas.offsetLeft;
```

Controlling it with the mouse - 2

```
1   /**
2    * Handles a MouseMove event
3    * @param {object} evt MouseMove event
4    */
5   function mouseMoveHandler(evt)
6   {
7       let relativeX = evt.clientX - canvas.offsetLeft;
8       let outLeft  = relativeX - SIZE_PADDLE_WIDTH/2 <= 0;
9       let outRight = relativeX >= canvas.width - SIZE_PADDLE_WIDTH/2;
10      if ( !outLeft && !outRight )
11      {
12          xPaddle = relativeX - SIZE_PADDLE_WIDTH/2;
13      }
14      else if ( outLeft )
15      {
16          xPaddle = 0;
17      }
18      else if ( outRight )
19      {
20          xPaddle = canvas.width - SIZE_PADDLE_WIDTH;
21      }
22  }
```

Checkpoint 3



4. End and restart game

Losing the game - 1

Game is lost when ball hits the lower wall (ignore the paddle for now).

Modify the `updateBall()` function to detect this condition.

Exercise 4

```
1    let isGameOver = false;
2
3    function updateBall(ctx)
4    {
5        /* Define the isHitting* variables here */
6
7        if ( isHittingRightWall || isHittingLeftWall )
8        {
9            xBallVelocity = -xBallVelocity;
10       }
11       if ( isHittingUpperWall )
12       {
13           yBallVelocity = -yBallVelocity;
14       }
15       else if ( isHittingLowerWall )
16       {
17           isGameOver = true;
18       }
19
20       xBall += xBallVelocity;
21       yBall += yBallVelocity;
22   }
```

Losing the game - 2

Answer to Exercise 4

```
1   let isHittingRightWall = (xBall + xBallVelocity >
2                             (ctx.canvas.width - SIZE_BALL_RADIUS));
3   let isHittingLeftWall  = (xBall + xBallVelocity < SIZE_BALL_RADIUS);
4   let isHittingUpperWall = (yBall + yBallVelocity < SIZE_BALL_RADIUS);
5   let isHittingLowerWall = (yBall + yBallVelocity >
6                             (ctx.canvas.height - SIZE_BALL_RADIUS));
```

Restarting the game - the steps

The sequence of events when game is over is

1. `window.clearInterval()` is called to stop the game
2. player is asked to press `ENTER` key to continue
3. when `ENTER` key is pressed, game is restarted by calling `window.setInterval()`

Restarting the game - step 1

Stop the draw loop using `window.clearInterval()`. documentation

```
1    let mainGame = window.setInterval(main, 10, context);
2
3    function main(ctx)
4    {
5        ...
6
7        if (isGameOver)
8        {
9            window.clearInterval(mainGame);
10       }
11   }
```

Restarting the game - step 2

Prompt player to press the `ENTER` key

```
function main(ctx)
{
    ...

    if (isGameOver)
    {
        window.clearInterval(mainGame);
        writeText(ctx, "GAME OVER",
                  ctx.canvas.width/2, ctx.canvas.height/2,
                  "center", "40px Helvetica", "red");
        writeText(ctx, "Press the ENTER key to continue",
                  ctx.canvas.width/2, ctx.canvas.height/2+40,
                  "center", "12pt Helvetica", "red");
    }
}
```

Restarting the game - step 3

Restart game when `ENTER` is pressed

```
function keyUpHandler(evt)
{
    switch(evt.key)
    {
        ...
        case "Enter":
            if (isGameOver)
            {
                /* a. Reset game state, e.g. win/lose, score */

                /* b. Reset ball and paddle states */

                mainGame = window.setInterval(main, 10, context);
                return;
            }
            break;
    }
}
```

A bit of code refactoring

Code refactorization helps avoid code duplication.

- refactor code that initialize game states into one function
- refactor code that initialize ball and paddle states into one function

Restarting the game - step 3a

```
    let isGameOver;

    function resetGame()
    {
        isGameOver = false;
    }
```

Restarting the game - step 3b

```
let xBall, yBall;
let xBallVelocity, yBallVelocity;
let xPaddle, yPaddle;

/**
 * Initialize ball and paddle position
 * @param {object} ctx The 2D context of a Canvas
 */
function initBallPaddle(ctx)
{
    xBall = ctx.canvas.width / 2;
    yBall = ctx.canvas.height - 30;

    xPaddle = (ctx.canvas.width - SIZE_PADDLE_WIDTH)/2;
    yPaddle = ctx.canvas.height - SIZE_PADDLE_HEIGHT;

    xBallVelocity =  2;
    yBallVelocity = -2;
}
```

Restarting the game - step 3 completed

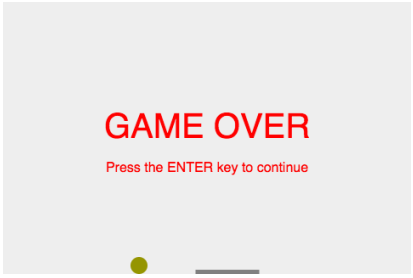
```
resetGame();
initBallPaddle(context);
mainGame = window.setInterval(main, 10, context);
...

function keyUpHandler(evt)
{
    switch(evt.key)
    {
        ...
        case "Enter":
            if (isGameOver)
            {
                resetGame();
                initBallPaddle(context);
                mainGame = window.setInterval(main, 10, context);
                return;
            }
            break;
    }
}
```

Losing the game - 3 *finally*

Now we take into consideration the paddle's position.

```
function updateBall(ctx)
{
    ...
    else if (isHittingLowerWall)
    {
        // ball is at the lower boundary of the canvas
        // is it hitting the paddle?
        let isPaddleInRange = (xBall >= xPaddle &&
                               xBall <= xPaddle + SIZE_PADDLE_WIDTH);
        if (isPaddleInRange)
        {
            // treat the paddle like a wall
            yBallVelocity = -yBallVelocity;
        }
        else
        {
            isGameOver = true;
        }
    }
}
```

Checkpoint 4



5. Add monsters (or bricks)

The brick object

In Javascript, objects are created in the JSON (JavaScript Object Notation) format. A primer on JSON

```
brick = {
            x     : 10,    // x-coordinate
            y     : 20,    // y-coordinate
            isHit : false  // has it been hit?
        }
```

A few constants on bricks

```
const SIZE_BRICK_HEIGHT        = 20;
const SIZE_BRICK_WIDTH         = 75;
const SIZE_BRICK_WALL_GAP_TOP  = 30;
const SIZE_BRICK_WALL_GAP_LEFT = 30;
const SIZE_BRICK_BRICK_GAP     = 10;
const COLOR_BRICK_FILL_STYLE   = "#AA2222";
```

Building a matrix of them

```
const SIZE_NUM_ROWS_BRICKS     =  5;
const SIZE_NUM_COLS_BRICKS     =  3;
/**
 * Build a 2D matrix (array of arrays) of bricks
 * @returns {array} a 2D matrix of bricks
 */
function buildBricks()
{
    let returnValue = [];
    for(let c = 0; c < SIZE_NUM_COLS_BRICKS; c++) {
        returnValue[c] = [];
        for(let r = 0; r < SIZE_NUM_ROWS_BRICKS; r++) {
            let xBrick = (r*(SIZE_BRICK_WIDTH  + SIZE_BRICK_BRICK_GAP)) +
                          SIZE_BRICK_WALL_GAP_LEFT;
            let yBrick = (c*(SIZE_BRICK_HEIGHT + SIZE_BRICK_BRICK_GAP)) +
                          SIZE_BRICK_WALL_GAP_TOP;
            let brick = {x: xBrick, y: yBrick, isHit: false};
            returnValue[c][r] = brick;
        }
    }
    return returnValue;
}
```

Drawing them

```
let bricks = buildBricks();

/**
 * Draw bricks - only unhit bricks are drawn
 * @param {object} ctx 2D context of a Canvas
 */
function drawBricks(ctx)
{
    for(let c = 0; c < SIZE_NUM_COLS_BRICKS; c++)
    {
        for(let r = 0; r < SIZE_NUM_ROWS_BRICKS; r++)
        {
            let brick = bricks[c][r];
            if (!brick.isHit)
                drawRect(ctx, brick.x, brick.y,
                         SIZE_BRICK_WIDTH, SIZE_BRICK_HEIGHT,
                         COLOR_BRICK_FILL_STYLE, "");
        }
    }
}
```

Detecting a hit - 1

Ball can hit brick on any of its four sides.

- When the top or bottom side is hit, `yBallVelocity` changes direction
- When the left or right side is hit, `xBallVelocity` changes direction

Detecting a hit - 2

```
/**
 * Update bricks status - if any of the brick is hit
 */
function updateBricks()
{
    for(let c = 0; c < SIZE_NUM_COLS_BRICKS; c++) {
        for(let r = 0; r < SIZE_NUM_ROWS_BRICKS; r++) {
            let b = bricks[c][r];
            if(b.isHit == false) {
                /* Define isHitFromTopOrBottom and isHitFromLeftOrRight */
                if (isHitFromTopOrBottom)
                {
                    yBallVelocity = -yBallVelocity; b.isHit = true;
                }
                else if (inHitFromLeftOrRight)
                {
                    xBallVelocity = -xBallVelocity; b.isHit = true;
                }
            }
        }
    }
}
```

Detecting a hit - 3

When hit from top or bottom

```
let xInRange = (xBall >= b.x &&
                xBall <= b.x + SIZE_BRICK_WIDTH);

let yInRangeFromTop
    = ((yBall + SIZE_BALL_RADIUS) >= b.y &&
       (yBall + SIZE_BALL_RADIUS) <  b.y + SIZE_BRICK_HEIGHT);

let yInRangeFromBottom
    = ((yBall - SIZE_BALL_RADIUS) >  b.y &&
       (yBall - SIZE_BALL_RADIUS) <= b.y + SIZE_BRICK_HEIGHT);

let isHitFromTopOrBottom = (xInRange &&
                            (yInRangeFromTop || yInRangeFromBottom));
```

Exercise 5

Write code to compute `isHitFromLeftOrRight`.

Detecting a hit - 4

Answer to Exercise 5 - when hit from left or right

```
let yInRange = (yBall >= b.y &&
                yBall <= b.y + SIZE_BRICK_HEIGHT);

let xInRangeFromLeft
    = ((xBall + SIZE_BALL_RADIUS) >= b.x &&
       (xBall + SIZE_BALL_RADIUS) <  b.x + SIZE_BRICK_WIDTH);

let xInRangeFromRight
    = ((xBall - SIZE_BALL_RADIUS) >  b.x &&
       (xBall - SIZE_BALL_RADIUS) <= b.x + SIZE_BRICK_WIDTH);

let inHitFromLeftOrRight = (yInRange &&
                            (xInRangeFromLeft || xInRangeFromRight));
```

Putting it in the draw loop

```
function main(ctx)
{
    ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);

    drawBricks(ctx);

    ...

    if (isGameOver)
    {
        ...
    }

    updatePaddle(ctx);

    updateBricks();
}
```

Rebuilding the bricks before restarting

```
function keyUpHandler(evt)
{
    ...
        case "Enter":
            if (isGameOver)
            {
                resetGame();
                bricks = buildBricks();
                initBallPaddle(context);
                mainGame = window.setInterval(main, 10, context);
                return;
            }
            break;
}
```

Checkpoint 5

6. Keep track of score, lives

Adding more game states

```
1       // is Game Over?
2       let isGameOver;
3
4       // number of bricks hit so far
5       let numBricksHit;
6
7       // number of lives left
8       let numLives;
9
10      // has player won?
11      let isGameWon;
12
13      ...
14
15      function resetGame()
16      {
17          isGameOver    = false;
18          numBricksHit  =  0;
19          numLives      =  3;
20          isGameWon     = false;
21      }
```

Keeping track of score

```
1   function updateBricks()
2   {
3       ...
4       if (xInRange && (yInRangeFromTop || yInRangeFromBottom))
5       {
6           yBallVelocity = -yBallVelocity;
7           b.isHit = true;
8           numBricksHit ++;
9       }
10      else if (yInRange && (xInRangeFromLeft || xInRangeFromRight))
11      {
12          xBallVelocity = -xBallVelocity;
13          b.isHit = true;
14          numBricksHit ++;
15      }
16  }
```

Keeping track of lives - 1 *What's wrong?*

```
1   function updateBall(ctx)
2   {
3           if (isPaddleInRange)
4           ...
5           else
6           {
7               numLives --;
8               if (numLives == 0)
9               {
10                  isGameOver = true;
11              }
12              else
13              {
14                  window.clearInterval(mainGame);
15                  writeText(ctx, "Live(s) left: " + numLives,
16                            ctx.canvas.width/2, ctx.canvas.height/2,
17                            "center", "40px Helvetica", "red");
18                  initBallPaddle(ctx);
19                  mainGame = window.setInterval(main, 10, ctx);
20              }
21          }
22  }
```

Keeping track of lives - 2

Use `window.setTimeout()` to delay the execution of a function.
documentation

To turn a statement into a function, we wrap it in an *anonymous function*.

```
1   window.setTimeout(function(c)
2                     {
3                         mainGame = window.setInterval(main, 10, c);
4                     },
5                     2000,
6                     context);
```

Displaying score and lives

```
1   function main(ctx)
2   {
3       ...
4       /* after drawing the ball and paddle */
5
6       // draw the score
7       writeText(ctx, "Score: " + numBricksHit, 8, 20,
8                 "left", "16px Helvetica", "black");
9
10      // draw number of lives remaining
11      writeText(ctx, "Lives: " + numLives, ctx.canvas.width-65, 20,
12                "left", "16px Helvetica", "black");
13
14      ...
15  }
```

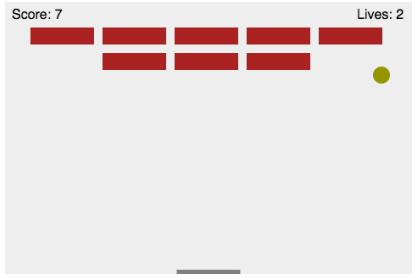
Winning the game - 1

```
1   function main(ctx)
2   {
3       ...
4       // check game-won condition
5       if (isGameWon)
6       {
7           window.clearInterval(mainGame);
8           writeText(ctx, "You Won!",
9                     ctx.canvas.width/2, ctx.canvas.height/2, "center", "40px Helvetica", "red");
10          writeText(ctx, "Press the ENTER key to continue",
11                    ctx.canvas.width/2, ctx.canvas.height/2+40, "center", "12pt Helvetica", "red");
12      }
13      updateBall(ctx);
14      ...
15      updateBricks(ctx);
16
17      if (numBricksHit == SIZE_NUM_COLS_BRICKS * SIZE_NUM_ROWS_BRICKS)
18          isGameWon = true;
19  }
```

Winning the game - 2

```
1   function keyUpHandler(evt)
2   {
3       ...
4           case "Enter":
5               if (isGameOver || isGameWon)
6               {
7                   resetGame();
8                   bricks = buildBricks();
9                   initBallPaddle(context);
10                  mainGame = window.setInterval(main, 10, context);
11                  return;
12              }
13              break;
14  }
```

Checkpoint 6



The END :-)