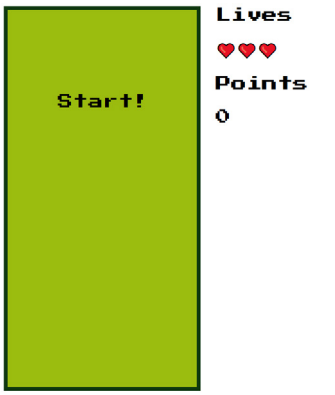
HOUR OF CODE INSTRUCTIONS

JAVASCRIPT // BRICK BREAKER

1. Open the file Game.html. It will use your computer's default browser. You should see something like this. What happens when you click on "Start!"?
2. This game is missing all collision and controls, and those need to be fixed before the game will run properly. Open the script.js file in a code editor.

   You'll see several sections of code with nothing in them but long comments.




```
document.addEventListener("keydown", event => {
    //  use switch case to check if the right or left arrow is pressed, take the parameter event.key

    //  first case - if the right arrow "ArrowRight" is pressed, the paddle has the moveRight() function applied to it, and the loop is broken (break;)

    //  second case - if the left arrow "ArrowLeft" is pressed, the paddle has the moveLeft() function applied to it, and the loop is broken (break;)

    //  default case

});
```

3. Let's start with this section, the "event listener" for "keydown", this part of the code is always going to check for keys being pressed, and is how you're going to add controls to this game. See if you can follow the hints to guess what the code will do before moving on.
4. The first comment (line 10) suggests using switch case with the input parameter "event.key". In code, that looks like this:

```
switch (event.key) {

}
```

5. There are two cases under this switch statement - if the "ArrowRight" is pressed, or if the "ArrowLeft" is pressed. There is also a default case.

   The "moveRight()" and "moveLeft()" functions are already written, you just need to apply them. Inside the switch statement, write the following code:

```
case "ArrowRight":
  paddle.moveRight();
  break;

case "ArrowLeft":
  paddle.moveLeft();
  break;

default:
  break;
```

```
document.addEventListener("keydown", event => {
    //  use switch case to check if the right or le
    switch (event.key) {
    //  first case - if the right arrow "ArrowRight
      case "ArrowRight":
        paddle.moveRight();
        break;
    //  second case - if the left arrow "ArrowLeft"
      case "ArrowLeft":
        paddle.moveLeft();
        break;
    //  default case
      default:
        break;
    }
});
```

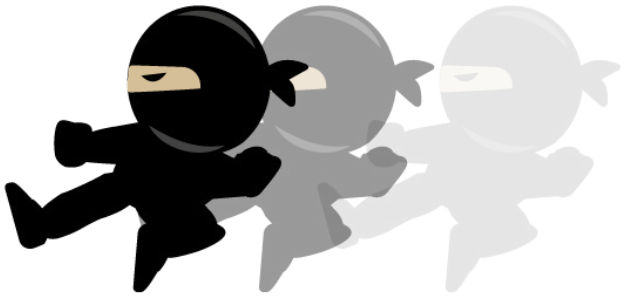
CODE NINJAS®

HOUR OF CODE INSTRUCTIONS
JAVASCRIPT // BRICK BREAKER

HOUR OF CODE

6. Save the script (most code editors will let you use "Ctrl+S" to save), and then refresh the page (press the F5 key) you opened earlier containing the Game.html file.

   You can press left or right to move the paddle but the paddle never stops moving! Refresh the page to restart it. Open the script back up, it's time to code the "keyup" event listener.



7. The "keyup" code is identical to the "keydown" code, except this time you're using the "stop()" function instead of the "moveLeft()" and "moveRight()" functions.

```
switch (event.key) {
  case "ArrowRight":
    paddle.stop();
    break;
  case "ArrowLeft":
    paddle.stop();
    break;
  default:
    break;
}
```

```
26  document.addEventListener("keyup", event => {
27      // all that needs to be different here is
28      // use switch case to check if the right
29      switch (event.key) {
30      //  first case - if the right arrow is rel
31        case "ArrowRight":
32          paddle.stop();
33          break;
34      //  second case - if the left arrow is rel
35        case "ArrowLeft":
36          paddle.stop();
37          break;
38      //  default case
39        default:
40          break;
41      }
42  });
```

8. Save the script, and refresh the Game.html page. The controls stop when you need them to now, but the paddle can go over the sides of the screen. Let's block that using the constant we're calling "paddleCheck".

9. First, let's make an if statement that stops the paddle's x-value going lower than 0. This means the paddle can't pass any further left in the game space than the left border.

```
if (paddle.x <= 0) {
  paddle.x = 0;
}
```

10. Figuring out the right border is a little harder, we can measure the width of the scene and use that exact value, but there's a better way to do it! We can use "defaults.width" to check the width of the game, and paddle.w to check the width of the paddle object.

```
if (paddle.x >= defaults.width
- paddle.w) {
  paddle.x = defaults.width -
  paddle.w;
}
```

CODE NINJAS®

HOUR OF CODE INSTRUCTIONS
JAVASCRIPT // BRICK BREAKER

11. The completed "paddleCheck" should look like the screenshot to the right.

    After saving the script and refreshing the Game.html page, your paddle should no longer be able to leave the screen.

12. Now it's time to fix the most obvious problem the game has left - the collision! We've already done one collision check - "paddleCheck" test collision between the walls and the paddle. What other collisions are left?

    The ball will collide with the blocks at the top of the screen, the ball will collide with the player's paddle, and the ball will collide with the bounds of the wall.

    Inside the code for ballsBlockCheck, we're going to add some new code that will apply to every block that's ever in the scene, even if the ones from a previous level are deleted and the new levels have many more blocks.

```
for (const block of game.blocks) {

}
```

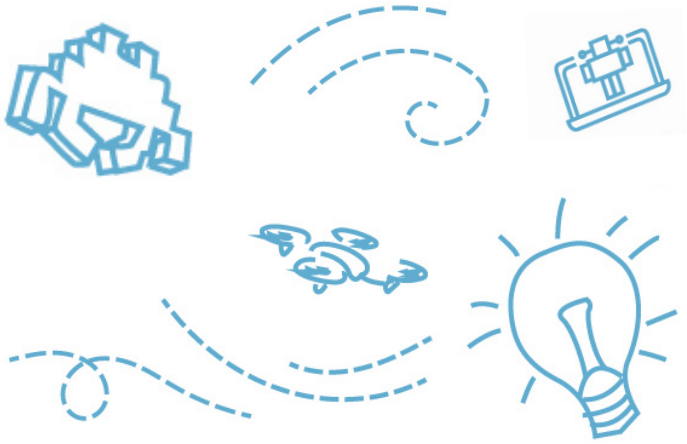
13. So what happens when the ball is touching a block?

    First, it's going to bounce. Then, we're going to have the game play a sound, audio feedback adds a fun layer to games. Then, the player scores a point for hitting a block. Last, the block needs to be removed.

    Those 4 actions can be broken down into 4 functions:

```
if (ball.isTouching(block)) {
  ball.bounceWith(block);
  synth.playBlockSound();
  game.addPoints();
  game.remove(block);
}
```

```
const paddleCheck = () => {
    //  the object "paddle" needs a left boundary
    if (paddle.x <= 0) {
      paddle.x = 0;
    }
    //  and a right boundary, the "paddle" object
    if (paddle.x >= defaults.width - paddle.w) {
      paddle.x = defaults.width - paddle.w;
    }
};
```



```
const ballBlockCheck = () => {
    //  for all blocks in the game - thi
    for (const block of game.blocks) {
    //  if the object "ball" isTouching
      if (ball.isTouching(block)) {
    //  then the object "ball" will bou
        ball.bounceWith(block);
    //  then the object "synth" will pl
        synth.playBlockSound();
    //  then to the game object, apply
        game.addPoints();
    //  then to the game object, apply
        game.remove(block);
      }
    }
};
```

Check your script against the image above, save it, and then refresh the Game.html to test it. Well, there's no testing you can do if it's going through the paddle; let's code that next.

CODE NINJAS®

HOUR OF CODE INSTRUCTIONS
JAVASCRIPT // BRICK BREAKER

14. Move onto the "ballPaddleCheck" statement.

    Inside, we want to add code telling the ball what to do when it hits the paddle. Just 2 things this time: it needs to bounce and play a sound.

```
if (ball.isTouching(paddle)) {
  ball.bounceWith(paddle);
  synth.playPaddleSound();
}
```

15. Save the script, refresh Game.html and playtest your game a little now. You might be able to play for a little bit, but as soon as the ball hits any of the boundaries of the screen, you'll run into problems.

    Go back to the script.js file, and find the blank "ballBoundsCheck". There are several comments here to guide you. Can you follow their hints and write all of the code for the "ballBoundsCheck"? The next few steps go over the code you need to write here.

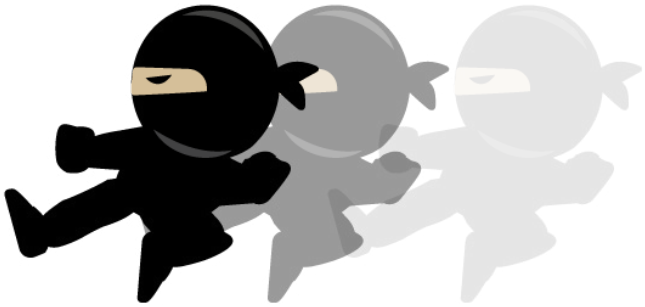
16. First, let's go over the code you need for what happens when the ball hits the left wall. A boolean already exists for it, "ballHitLeftWall()", and the ball has properties vx for horizontal motion, and vy for vertical motion.

```
if (ballHitLeftWall()) {
  ball.x = 0;
  ball.vx = -ball.vx;
  synth.playWallSound();
}
```

17. The code for what happens when the ball hits the right wall is very similar to when it hits the left wall.

```
if (ballHitRightWall()) {
  ball.x = game.width;
  ball.vx = -ball.vx;
  synth.playWallSound();
}
```

```
const ballPaddleCheck = () => {
  //  if the object "ball" isTouchin
  if (ball.isTouching(paddle)) {
  // then the object "ball" will bou
    ball.bounceWith(paddle);
  // and then the object "synth" wil
    synth.playPaddleSound();
  }
};
```



```
if (ballHitLeftWall()) {
  //  then the object "bal
  ball.x = 0;
  //  then the object "bal
  ball.vx = -ball.vx;
  //  then the object "syn
  synth.playWallSound();
}
```

```
if (ballHitRightWall()) {
  //  then the object "bal
  ball.x = game.width;
  //  then the object "bal
  ball.vx = -ball.vx;
  //  then the object "syn
  synth.playWallSound();
}
```

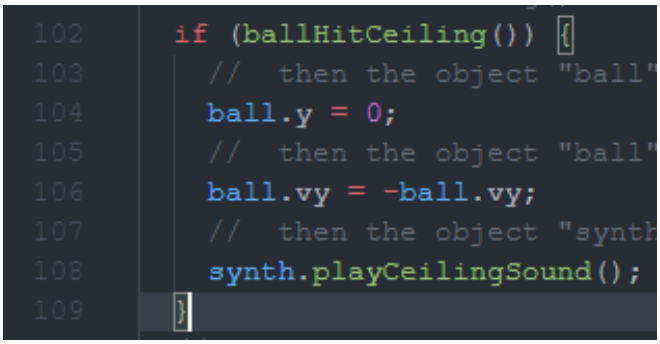
CODE NINJAS®

HOUR OF CODE INSTRUCTIONS
JAVASCRIPT // BRICK BREAKER

HOUR OF CODE

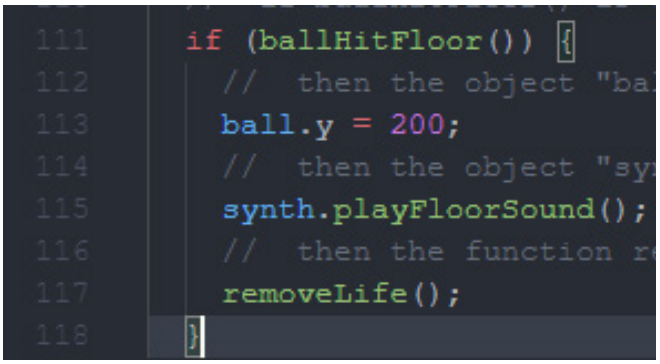
18. What happens when the ball hits the ceiling? It's similar to what happens when the ball hits the walls, but along the y-axis instead of the x-axis.

```
if (ballHitCeiling()) {
  ball.y = 0;
  ball.vy = -ball.vy;
  synth.playCeilingSound();
}
```

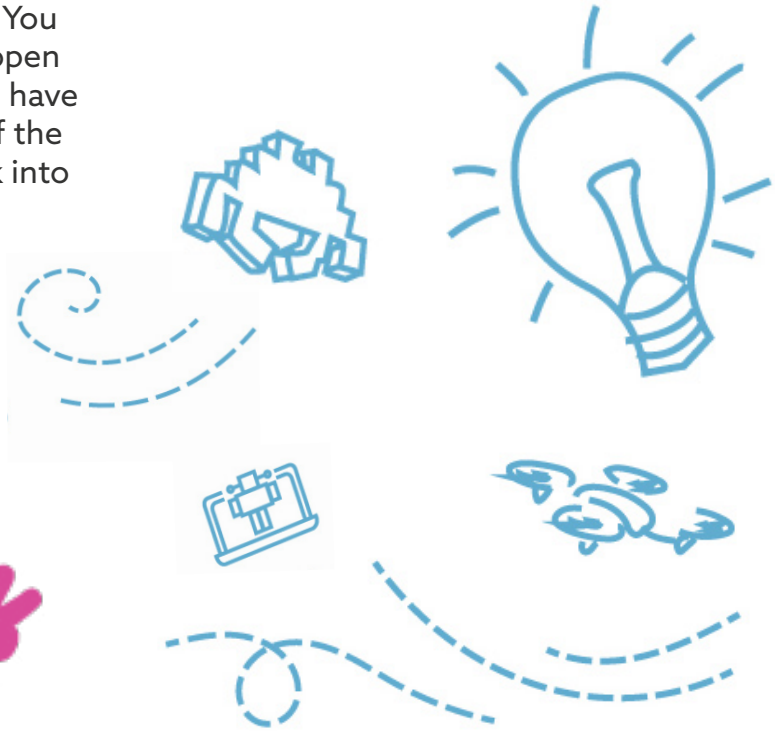


19. The last line of "ballBoundsCheck" has one thing that's very new - the removeLife() function. If the player lets the ball go past the paddle and hit the floor, then they lose a life. Here's how to code that:

```
if (ballHitFloor()) {
  ball.y = 200;
  synth.playFloorSound();
  removeLife();
}
```



20. Save the script.js file. You're done! You can refresh the Game.html file or open it again to play Brick Breaker. If you have time, you can even try out some of the bonus activities that have you look into the code for the Game.html and setup.js files.



CODE NINJAS®