BÀI THỰC HÀNH 6
JavaScript nâng cao - OOP

Bài: **Thêm tính năng vào game Bóng nẩy**.

1. *Nội dung:*

Trong bài tập này, các bạn sinh viên sẽ được kiểm tra kiến thức về cách dùng JavaScript hướng đối tượng để hoàn thiện chức năng game bóng nẩy trên ứng dụng web.

Yêu cầu	Trước khi làm bài tập này, sinh viên cần xem lại nội dung bài học Chương 4 - JavaScript OOP
Mục tiêu	Kiểm tra kiến thức JavaScript hướng đối tượng.

2. *Dữ liệu khởi đầu:*

Sinh viên tải dữ liệu khởi đầu ở đây bao gồm 3 tập tin:

- Tập tin index-finished.html:

```html
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Bouncing balls</title>
  <link rel="stylesheet" href="style.css">
</head>

<body>
  <h1>bouncing balls</h1>
  <canvas></canvas>

  <script src="main-finished-es6.js"></script>
</body>

</html>
```

- Tập tin style.css:

```css
html,
body {
  margin: 0;
}

html {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  height: 100%;
}
```

```
body {
  overflow: hidden;
  height: inherit;
}

h1 {
  font-size: 2rem;
  letter-spacing: -1px;
  position: absolute;
  margin: 0;
  top: -4px;
  right: 5px;

  color: transparent;
  text-shadow: 0 0 4px white;
}
```

- Tập tin main-finished.js:

```
// setup canvas

const canvas = document.querySelector("canvas");
const ctx = canvas.getContext("2d");

const width = (canvas.width = window.innerWidth);
const height = (canvas.height = window.innerHeight);

// function to generate random number

function random(min, max) {
  const num = Math.floor(Math.random() * (max - min + 1)) + min;
  return num;
}

// define Ball constructor

function Ball(x, y, velX, velY, color, size) {
  this.x = x;
  this.y = y;
  this.velX = velX;
  this.velY = velY;
  this.color = color;
  this.size = size;
}

// define ball draw method
```

```
Ball.prototype.draw = function () {
  ctx.beginPath();
  ctx.fillStyle = this.color;
  ctx.arc(this.x, this.y, this.size, 0, 2 * Math.PI);
  ctx.fill();
};

// define ball update method

Ball.prototype.update = function () {
  if (this.x + this.size >= width) {
    this.velX = -this.velX;
  }

  if (this.x - this.size <= 0) {
    this.velX = -this.velX;
  }

  if (this.y + this.size >= height) {
    this.velY = -this.velY;
  }

  if (this.y - this.size <= 0) {
    this.velY = -this.velY;
  }

  this.x += this.velX;
  this.y += this.velY;
};

// define ball collision detection

Ball.prototype.collisionDetect = function () {
  for (let j = 0; j < balls.length; j++) {
    if (!(this === balls[j])) {
      const dx = this.x - balls[j].x;
      const dy = this.y - balls[j].y;
      const distance = Math.sqrt(dx * dx + dy * dy);

      if (distance < this.size + balls[j].size) {
        balls[j].color = this.color =
          "rgb(" +
          random(0, 255) +
          "," +
          random(0, 255) +
          "," +
          random(0, 255) +
```

```
          ")";
      }
    }
  }
};

// define array to store balls and populate it

let balls = [];

while (balls.length < 25) {
  const size = random(10, 20);
  let ball = new Ball(
    // ball position always drawn at least one ball width
    // away from the adge of the canvas, to avoid drawing errors
    random(0 + size, width - size),
    random(0 + size, height - size),
    random(-7, 7),
    random(-7, 7),
    "rgb(" + random(0, 255) + "," + random(0, 255) + "," + random(0, 255) + ")",
    size
  );
  balls.push(ball);
}

// define loop that keeps drawing the scene constantly

function loop() {
  ctx.fillStyle = "rgba(0,0,0,0.25)";
  ctx.fillRect(0, 0, width, height);

  for (let i = 0; i < balls.length; i++) {
    balls[i].draw();
    balls[i].update();
    balls[i].collisionDetect();
  }

  requestAnimationFrame(loop);
}

loop();
```

- Tập tin main-finished-es6.js cho bạn nào muốn code theo phong cách mới:

```
// setup canvas

const canvas = document.querySelector("canvas");
const ctx = canvas.getContext("2d");
```

```
const width = (canvas.width = window.innerWidth);
const height = (canvas.height = window.innerHeight);

// function to generate random number

function random(min, max) {
  const num = Math.floor(Math.random() * (max - min + 1)) + min;
  return num;
}

class Ball {
  constructor(x, y, velX, velY, color, size) {
    this.x = x;
    this.y = y;
    this.velX = velX;
    this.velY = velY;
    this.color = color;
    this.size = size;
  }

  draw() {
    ctx.beginPath();
    ctx.fillStyle = this.color;
    ctx.arc(this.x, this.y, this.size, 0, 2 * Math.PI);
    ctx.fill();
  }

  update() {
    if (this.x + this.size >= width) {
      this.velX = -this.velX;
    }

    if (this.x - this.size <= 0) {
      this.velX = -this.velX;
    }

    if (this.y + this.size >= height) {
      this.velY = -this.velY;
    }

    if (this.y - this.size <= 0) {
      this.velY = -this.velY;
    }

    this.x += this.velX;
    this.y += this.velY;
```

```
  }

  collisionDetect() {
    for (let j = 0; j < balls.length; j++) {
      if (!(this === balls[j])) {
        const dx = this.x - balls[j].x;
        const dy = this.y - balls[j].y;
        const distance = Math.sqrt(dx * dx + dy * dy);

        if (distance < this.size + balls[j].size) {
          balls[j].color = this.color =
            "rgb(" +
            random(0, 255) +
            "," +
            random(0, 255) +
            "," +
            random(0, 255) +
            ")";
        }
      }
    }
  }
}

var balls = [];

while (balls.length < 25) {
  const size = random(10, 20);
  let ball = new Ball(
    // ball position always drawn at least one ball width
    // away from the edge of the canvas, to avoid drawing errors
    random(0 + size, width - size),
    random(0 + size, height - size),
    random(-7, 7),
    random(-7, 7),
    "rgb(" + random(0, 255) + "," + random(0, 255) + "," + random(0, 255) + ")",
    size
  );

  balls.push(ball);
}

function loop() {
  ctx.fillStyle = "rgba(0, 0, 0, 0.25)";
  ctx.fillRect(0, 0, width, height);

  for (let i = 0; i < balls.length; i++) {
```

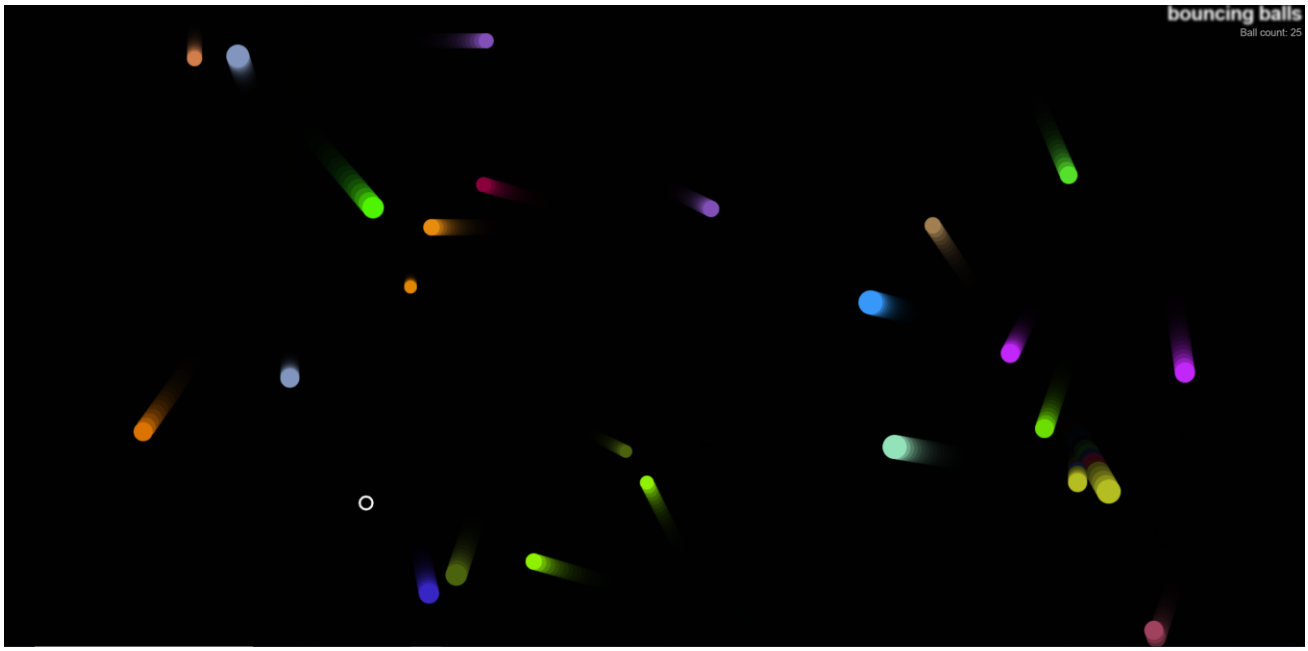
```
    balls[i].draw();
    balls[i].update();
    balls[i].collisionDetect();
  }

  requestAnimationFrame(loop);
}

loop();
```

3. *Tóm tắt các việc thực hiện:*

- Khi mở ứng dụng lên (trang index-finished.html) bạn sẽ thấy giao diện gồm các quả bóng nẩy và thay đổi màu khi chạm vào nhau. Trong bài tập này, các bạn sinh viên cần thêm một ít tính năng bao gồm: một "vòng tròn ma quỷ" (EvilCircle) sẽ "ăn" các quả bóng nếu chúng chạy ngang qua. Đồng thời, bài thực hành cũng kiểm tra sự hiểu biết cách tạo đối tượng (object) của sinh viên thông qua việc tạo một đối tượng generic Shape() và các đối tượng bóng nẩy (ball) và "vòng tròn ma quỷ" sẽ kế thừa từ đối tượng này. Cuối cùng, bài thực hành cũng yêu cầu sinh viên thêm tính năng đếm các quả bóng còn lại chưa bị "ăn".
- Dưới đây là màn hình kết thúc khi thực hiện xong chương trình:



4. *Hướng dẫn thực hiện*

4.1. *Tạo đối tượng mới:*

Đầu tiên, thay đổi hàm dựng (constructor) Ball() để nó trở thành hàm dựng Shape() và thêm mới hàm dựng Ball():

- Hàm dựng Shape() cần định nghĩa các thuộc tính: x, y, velX, velY tương tự như cách hàm dựng Ball() đã làm, lưu ý là không có thuộc tính color và size.
- Cần định nghĩa một thuộc tính mới gọi là exists, thuộc tính này được dùng để kiểm tra sự tồn tại của các quả bóng trong chương trình (những quả bóng chưa bị "ăn" bởi "vòng tròn ma quỷ"). Kiểu của nó là Boolean (true/false).
- Hàm dựng Ball() cần kế thừa các thuộc tính x, y, velX, velY và exists từ hàm dựng Shape().
- Hàm dựng Ball() cũng cần định nghĩa các thuộc tính color và size, giống như hàm dựng Ball() nguyên thủy (code ban đầu) đã làm.
- Nhớ thiết lập prototype và hàm dựng của Ball() một cách hợp lý.
- Phương thức collisionDetect() của prototype Ball cần một sự thay đổi nhỏ. Bởi vì nếu giữ nguyên mã code thì "vòng tròn ma quỷ" sẽ bắt đầu ăn những quả bóng nẩy bằng cách đặt thuộc tính exists bằng false. Điều đó sẽ làm giảm số lượng bóng liên quan đến phát hiện va chạm. Một quả bóng chỉ cần được xem xét để phát hiện va chạm nếu thuộc tính exists là true. Vì vậy, cần thay đổi mã nguồn của phương thức collisionDetect() bằng mã sau:

```
Ball.prototype.collisionDetect = function () {
  for (let j = 0; j < balls.length; j++) {
    if (!(this === balls[j]) && balls[j].exists) {
      const dx = this.x - balls[j].x;
      const dy = this.y - balls[j].y;
      const distance = Math.sqrt(dx * dx + dy * dy);

      if (distance < this.size + balls[j].size) {
        balls[j].color = this.color =
          "rgb(" +
          random(0, 255) +
          "," +
          random(0, 255) +
          "," +
          random(0, 255) +
          ")";
      }
    }
  }
```

```
};
```

Như đề cập ở trên thì các bạn chỉ cần thêm mã nguồn để kiểm tra sự tồn tại của quả bóng - bằng cách thêm balls[j].exists trong dòng lệnh if.

- Giữ nguyên 2 phương thức draw() và update() như nguyên thủy.
- Bạn cần thêm một tham số (parameter) mới vào hàm dựng new Ball() khi nó được gọi và thuộc tính exists này nên nằm ở vị trí thứ 5, và giá trị được thiết lập là true.

4.2. *Định nghĩa "vòng tròn ma quỷ" EvilCircle():*

- Trong chương trình này chỉ có một "vòng tròn ma quỷ" và các bạn cần phải định nghĩa nó bằng cách sử dụng hàm dựng được kế thừa từ Shape().
- Hàm dựng EvilCircle() cần kế thừa các thuộc tính x, y, velX, velY và exists từ hàm dựng Shape(), nhưng velX và velY phải luôn có giá trị bằng 20.
- Gợi ý: bạn có thể làm giống như sau: Shape.call(this, x, y, 20, 20, exists);
- Nó cũng phải định nghĩa giá trị cho các thuộc tính của nó như:
    - color - 'white'
    - size - 10
- Các bạn nhớ định nghĩa các thuộc tính thông qua các tham số (parameter) trong hàm dựng, và thiết lập các thuộc tính của prototype và constructor một cách chính xác.

4.3. *Định nghĩa các phương thức của EvilCircle():*

Gồm 4 phương thức sau:

- draw(): phương thức này có mục đích giống với phương thức draw() trong Ball(). Nó vẽ một bản thể đối tượng trong một canvas. Vì sự giống nhau, nên bạn có thể sao chép định nghĩa từ Ball.prototype.draw. Và thêm một số sự thay đổi như sau:
    - Chương trình muốn "vòng tròn ma quỷ" rỗng, chỉ có đường viền ngoài mà thôi. Bạn có thể đạt được điều này bằng cách cập nhật fillStyle và fill() thành strokeStyle và stroke().
    - Chương trình cũng muốn đường viền dày hơn để bạn nhìn thấy "vòng tròn ma quỷ" dễ hơn. Bạn có thể làm được điều này bằng cách thiết lập giá trị cho lineWidth ở nơi gọi hàm beginPath() (giá trị 3 là hợp lý).
- checkBounds(): phương thức này sẽ có chức năng giống với phần đầu tiên của phương thức update() trong Ball() - kiểm tra xem "vòng tròn ma quỷ" có ra khỏi giới hạn khung

hình và ngăn chặn việc đó. Bạn có thể sao chép định nghĩa từ Ball.prototype.update, và thay đổi một ít trong mã nguồn như sau:

- Xóa 2 dòng cuối - vì chương trình không mong muốn việc cập nhật vị trí của "vòng tròng ma quỷ" theo mỗi khung hình, bởi vì chúng ta di chuyển nó bằng cách khác, bên dưới có hướng dẫn.
- Trong câu lệnh if(), nếu if trả về true, chương trình không cập nhật velX/velY; thay vào đó, chương trình muốn thay đổi giá trị x/y để "vòng tròn ma quỷ" được nẩy trên màn hình một chút. Thêm hoặc bớt (giá trị phù hợp) thuộc tính size của "vòng tròn ma quỷ" sẽ làm điều này dễ dàng được nhận ra.

- setControls(): phương thức này sẽ thêm một lắng nghe sự kiện onkeydown cho đối tượng window để khi các phím tương ứng được nhấn nó sẽ di chuyển "vòng tròn ma quỷ". Mã nguồn sau đây cần đặt vào trong phương thức này:

```
let _this = this;
window.onkeydown = function (e) {
  if (e.key === "a") {
    _this.x -= _this.velX;
  } else if (e.key === "d") {
    _this.x += _this.velX;
  } else if (e.key === "w") {
    _this.y -= _this.velY;
  } else if (e.key === "s") {
    _this.y += _this.velY;
  }
};
```

- Khi một phím tương ứng được nhấn, thì sự kiện nhấn này được lắng nghe và ghi nhận phím nào vừa được nhấn. Nếu nó nằm trong 4 phím được chỉ định trong mã nguồn ở trên thì "vòng tròn ma quỷ" sẽ được di chuyển tương ứng theo trái/phải/lên/xuống.
- *Bạn nên tìm hiểu và lý giải đoạn mã nguồn let _this = this; tại sao đặt ở vị trí đó? và có một số thứ có thể làm với phạm vi của phương thức.*

- collisionDetect(): phương thức này sẽ có chức năng giống phương thức collisionDetect() trong Ball(), nên bạn có thể sao chép nó và thay đổi một số khác biệt như sau:

- Trong câu lệnh if() bên ngoài bạn không cần kiểm tra xem quả bóng hiện tại trong vòng lặp có giống với quả bóng đang thực hiện kiểm tra hay không - bởi vì nó không còn là quả bóng bình thường nữa, nó là "vòng tròn ma quỷ"! Thay vào đó, bạn cần thực hiện kiểm tra để xem quả bóng đang được kiểm tra có tồn tại hay không (ta có thể thực hiện việc này với thuộc tính nào?). Nếu nó không tồn tại, thì nó đã bị "ăn" bởi "vòng tròn ma quỷ" rồi nên không cần thiết phải kiểm tra lại.
- Trong câu lệnh if() bên trong, bạn không còn muốn việc thay đổi màu sắc khi có sự va chạm - thay vào đó, bạn cần thiết lập việc bất kỳ quả bóng nào va chạm với "vòng tròn ma quỷ" sẽ không còn tồn tại nữa.

4.4. *Mang "vòng tròn ma quỷ" vào chương trình:*

Để mang "vòng tròn ma quỷ" vào chương trình, ta cần thay đổi một số thứ trong phương thức loop().

- Đầu tiên, tạo một bản thể của đối tượng "vòng tròn ma quỷ" (nhớ chỉ định các giá trị tham số phù hợp), sau đó gọi phương thức setControls() của nó. Bạn chỉ cần thực hiện việc này duy nhất 1 lần chứ không phải trong tất cả các vòng lặp.
- Tại nơi mà bạn lặp qua tất cả bóng và gọi phương thức draw(), update(), collisionDetect() cho mỗi quả, bạn cần thiết lập các phương thức này chỉ được gọi khi quả bóng còn tồn tại.
- Gọi phương thức draw(), checkBounds() và collisionDetect() của bản thể "vòng tròn ma quỷ" ở mỗi vòng lặp.

4.5. *Hiện thực chức năng đếm điểm:*

- Trong nội dung tập tin HTML, thêm phần tử <p> ngay bên dưới phần tử <h1> chứa nội dung "Ball count: ".
- Trong nội dung CSS, thêm luật (nguyên tắc) bên dưới:

```
p {
  position: absolute;
  margin: 0;
  top: 35px;
  right: 5px;
  color: #aaa;
}
```

- Trong nội dung tập tin JS, cập nhật các thay đổi sau:

- Tạo một biến tham chiếu tới phần tử <p> vừa thêm vào.
- Giữ việc hiển thị số lượng bóng trên màn hình.
- Tăng số đếm và hiển thị cập nhật số bóng mỗi lần bóng được thêm vào trang web.
- Giảm số đếm và hiển thị cập nhật số bóng mỗi lần bóng bị "ăn" bởi "vòng tròn ma quỷ" (vì nó không còn tồn tại nữa).