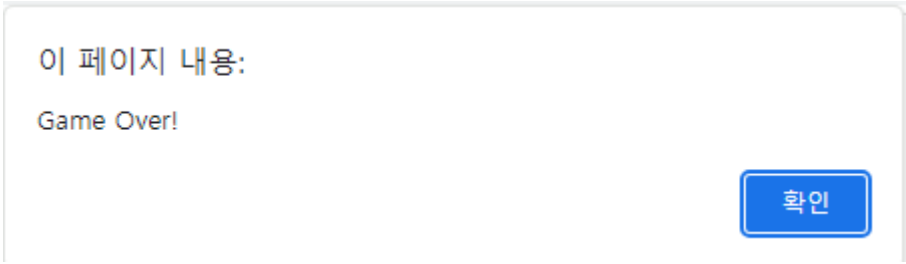
HTML GAME 응용

2020864003 이정환

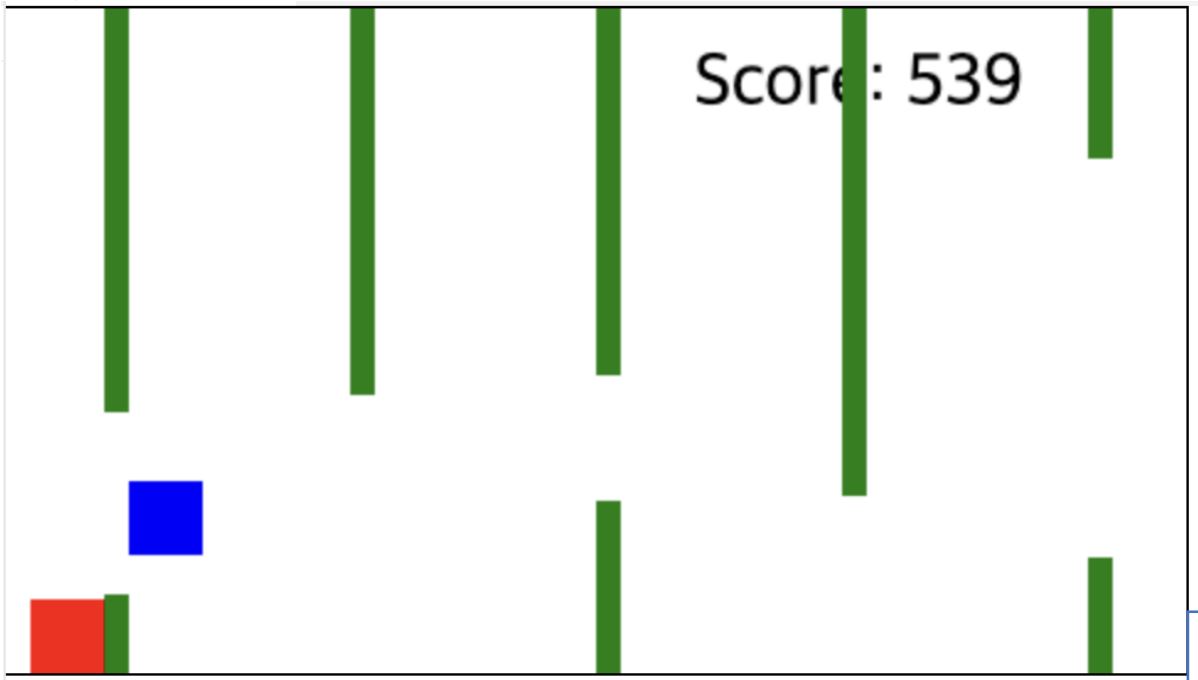
목차

실행 화면

첫 실행화면



게임 끝날때



게임 실행화면

코드 1

```
function startGame() {
myGamePiece = new component(30, 30, "red", 10, 120);
myGamePiece.gravity = 0.05;
myScore = new component("30px", "Consolas", "black", 280, 40, "text");
myObstacles = [];myVerticalPlayer = new component(30, 30, "blue", 50, 50);
myGameArea.start();
}
```

추가된 코드

코드 1

1. 게임 = **사각형 모양**의 Player 1(myGamePiece)와 **높이 30픽셀, 너비 30픽셀**의 Player 2 (myVerticalPlayer)로 이루어져 있음.

➔ 이 두 플레이어는 사용자에 의해 **조작됨**.

2. myGamePiece = new component(30, 30, "red", 10, 120);

➔ Player 1를 생성하고 초기 위치는 (10, 120)으로 설정함.

➔ 플레이어 1 의 크기는 30x30 픽셀이며, 색상은 **빨간색.**

3. myGamePiece.gravity = 0.05;

➔ 중력 값을 설정하여 플레이어에 **중력 효과**를 부여함.

➔ 플레이어는 아래로 떨어질 때 **중력의 영향**을 받음.

코드 1

4. myScore = new component("30px", "Consolas", "black", 280, 40, "text");

➔ 텍스트로 표시되는 **점수 요소를 생성**.

➔ 글꼴 크기는 **30px**이며, 글꼴은 **"Consolas"**로 지정.

5. myVerticalPlayer = new component(30, 30, "blue", 50, 50);

➔ 파란색으로 플레이어 2를 **생성**하고, 초기 위치는 (50, 50)으로 설정.

6. myGameArea.start();

-> **게임 영역을 시작**. 이는 Canvas를 **생성하고 초기화**하는 역할.

코드 2 - 1

```
var myGameArea = {
canvas: document.createElement("canvas"),
```

➔ myGameArea 객체의 canvas 속성은 HTML <canvas> 엘리먼트를 동적으로 생성

```
start: function() {
this.canvas.width = 480;
this.canvas.height = 270;
this.context = this.canvas.getContext("2d");
```

➔ start 메서드는 Canvas의 **너비와 높이**를 설정하고 2D 컨텍스트를 가져옴.
➔ 이는 게임 영역의 **크기를 결정**하고 그림을 그릴 수 있는 Canvas 컨텍스트를 얻는 단계.

코드 2 - 2

```
document.body.insertBefore(this.canvas, document.body.childNodes[0]);
this.frameNo = 0;
this.interval = setInterval(updateGameArea, 20);
window.addEventListener('keydown', function(e) {
myGameArea.keys = (myGameArea.keys || []);
myGameArea.keys[e.keyCode] = true;
});
window.addEventListener('keyup', function(e) {
myGameArea.keys[e.keyCode] = false;
});
},
clear: function() {
this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
}
}
```

➔ 1. Canvas **삽입 및 초기화**

➔ 2. 프레임 번호 및 **업데이트 간격**

➔ 3. 키보드 이벤트 **리스너 등록**

➔ 3. 키보드 이벤트 **리스너 등록**

➔ 4. Canvas **지우기**

코드 2 - 2

1. Canvas 삽입 및 초기화 =

```
document.body.insertBefore(this.canvas, document.body.childNodes[0]);
```

➔this.canvas는 HTML **<canvas> 엘리먼트**를 **동적**으로 생성한 것.

➔**insertBefore**를 사용하여 **this.canvas**를 HTML 문서의 **<body>에 삽입**함.

2. 프레임 번호 및 업데이트 간격 =

```
this.frameNo = 0;
this.interval = setInterval(updateGameArea, 20);
```

➔**frameNo**는 **현재 게임 프레임 번호**를 나타냄.

➔**setInterval 함수**를 사용하여 **updateGameArea 함수**를 **20초 간격**으로 호출.

➔이는 게임의 **주기적인 업데이트**를 담당함.

코드 2 - 2

3. 키보드 이벤트 리스너 등록 =

```
window.addEventListener('keydown', function(e) {
myGameArea.keys = (myGameArea.keys || []);
myGameArea.keys[e.keyCode] = true;});
```

➔ **keydown 이벤트**가 발생하면 해당 **키의 상태를 true**로 설정

```
window.addEventListener('keyup', function(e) {
myGameArea.keys[e.keyCode] = false;});
```

➔ **keyup 이벤트**가 발생하면 false로 설정.

➔ 이를 통해 **어떤 키가 눌렸는지**에 대한 정보를 저장할 수 있습니다.

4. Canvas 지우기 =

```
 clear: function() {
this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
```

➔ **clear 메서드**는 Canvas**를 지우는 역할**. clearRect **메서드**를 사용하여 Canvas**의 전체 영역을 지움.**

➔ 이는 각 **프레임**마다 **이전의 그림을 지우고 새로운 그림**을 그리기 위해 사용.

코드 3

```
function component(width, height, color, x, y, type) {
this.width = width;
this.height = height;
this.color = color;
this.x = x;
this.y = y;
this.type = type;
this.gravity = 0;
this.gravitySpeed = 0;
this.speedX = 0;
this.speedY = 0;
```

1. 각 구성 요소를 나타내는 component **클래스**를 정의.
2. 클래스의 생성자 함수로써, 새로운 게임 요소를 만들 때 호출

– 주요 속성은 다음과 같습니다:

- Width → 게임 요소의 **너비 (가로 길이)**.
- Height → 게임 요소의 **높이 (세로 길이)**.
- Color → 게임 요소의 **색상**.
- X → 게임 요소의 x **좌표 (가로 위치)**.
- Y → 게임 요소의 y **좌표 (세로 위치)**.
- Type → 게임 요소의 **종류 또는 타입**을 **식별하는 문자열** 또는 값을 나타냄.
- Gravity → **중력의 세기**를 나타내는 값. 기본값은 0으로 중력을 나타내지 않음.
- GravitySpeed → **중력에 의한 속도**. 기본값은 0으로 **중력의 영향을 받지 않음**을 나타냄.
- SpeedX → **x 방향**으로의 속도.
- SpeedY → **y 방향**으로의 속도.

코드 4

```
this.update = function() {
ctx = myGameArea.context;
if (this.type === "text") {
ctx.font = this.width + " " + this.height;
ctx.fillStyle = this.color;
ctx.fillText(this.text, this.x, this.y);
} else {
ctx.fillStyle = this.color;
ctx.fillRect(this.x, this.y, this.width, this.height);
}
};
```

1. Update 함수

```
this.newPos = function() {
this.gravitySpeed += this.gravity;
this.x += this.speedX;
this.y += this.speedY + this.gravitySpeed;
this.hitBottom();
};
```

2. newPos 함수

```
this.hitBottom = function() {
var rockbottom = myGameArea.canvas.height -
this.height;
if (this.y > rockbottom) {
this.y = rockbottom;
this.gravitySpeed = 0;
}
};
```

3. HitBottom 함수

➔ 코드 전체에서는 **myGameArea 및 객체의 초기 설정**이 필요하며,
➔ 게임 루프가 있어서 **주기적으로 update 및 newPos 함수**가 호출될 것입니다.

코드 4

1. Update 함수

- 게임 영역을 **업데이트**하고, 객체를 렌더링하는 역할.
- Ctx = Canvas 2D **컨텍스트**를 나타내며, myGameArea.context에서 가져옴.
- 객체의 타입이 "text"인 경우, 텍스트를 그림. 그렇지 않은 경우에는 **사각형**을 그립니다.
- **텍스트인 경우 = 폰트, 색상 및 위치**를 설정하고, fillText **메서드**를 사용하여 **텍스트를 그림**.
- 그 외의 경우 = **사각형의 위치와 크기**를 설정하고, fillRect **메서드**를 사용하여 **사각형을 그림**.

2. newPos 함수

- 객체의 새로운 위치를 계산.
- **중력 효과**를 시뮬레이션하기 위해 gravity와 gravitySpeed를 사용하여 **세로 방향으로의 속도를 조절**.
- speedX**와** speedY = 객체의 **가로 및 세로 방향 속도**이며, 이를 이용하여 **새로운 위치를 업데이트**.
- hitBottom **함수**를 호출하여 **객체가 화면 하단에 도달했는지** 확인함.

3. hitBottom 함수

- 객체가 화면 하단에 도달했는지 검사.
- 만약 객체가 화면 하단에 도달하면, rockbottom 변수를 **기준으로 객체를 다시 위**로 이동시킴.
- gravitySpeed를 **0으로 설정하여 중력 효과**를 초기화.

코드 5

```
this.crashWith = function(otherobj) {
var myleft = this.x;
var myright = this.x + (this.width);
var mytop = this.y;
var mybottom = this.y + (this.height);
var otherleft = otherobj.x;
var otherright = otherobj.x + (otherobj.width);
var othertop = otherobj.y;
var otherbottom = otherobj.y + (otherobj.height);
var crash = true;
if ((mybottom < othertop) ||
(mytop > otherbottom) ||
(myright < otherleft) ||
(myleft > otherright)) {
crash = false;
}
return crash;
};
}
```

➔ 이 조건문에서는 현재 객체의 **아래가 다른 객체의 위**에 있거나,
현재 객체의 **위가 다른 객체의 아래**에 있거나,
현재 객체의 **오른쪽이 다른 객체의 왼쪽**에 있거나,
현재 객체의 **왼쪽이 다른 객체의 오른쪽**에 있으면 충돌이 발생하지 않는 것으로 판단

➔ 이 때 crash 변수는 **false**로 설정.

코드 5

- myleft, myright, mytop, mybottom:

➔현재 객체의 좌표를 기반으로 **한 좌표 값**들로, 해당 객체의 **왼쪽, 오른쪽, 위, 아래**의 좌표를 나타냄.

- otherleft, otherright, othertop, otherbottom:

➔다른 객체(other object)의 좌표를 기반으로 **한 좌표 값**들로 해당 객체의 **왼쪽, 오른쪽, 위, 아래**의 좌표를 나타냄.

- Crash -> 충돌 여부를 **저장하는 변수**로, 초기값은 true로 설정됩니다.

코드 6 - 1

```
function updateGameArea() {
for (var i = 0; i < myObstacles.length; i++) {
if (myGamePiece.crashWith(myObstacles[i])) {
clearInterval(myGameArea.interval);
alert("Game Over!");
return;
}
}
myGameArea.clear();
myGameArea.frameNo += 1;

// Add vertical player movement
if (myGameArea.keys && myGameArea.keys[38]) {
myVerticalPlayer.y -= 2;
}
if (myGameArea.keys && myGameArea.keys[40]) {
myVerticalPlayer.y += 2;
}

myVerticalPlayer.update();
myGamePiece.newPos();
myGamePiece.update();
myScore.text = "Score: " +
myGameArea.frameNo;
myScore.update();
```

➔ 1. 장애물 충돌 검사

➔ 2. 게임 영역 지우기

➔ 3. 프레임 번호 증가

➔ 4. 수직 플레이어 (Player 2) 검사

➔ 5. 플레이어와 수직 플레이어 업데이트

➔ 6. 점수 업데이트

코드 6 - 1

1. 장애물 충돌 검사: 현재 플레이어 **객체(myGamePiece)**가 각 장애물(myObstacles)과 **충돌하는지 검사**. 만약 충돌이 감지되면, clearInterval을 사용하여 게임의 주기적인 업데이트를 중단하고, "Game Over!" 알림을 띄우고 함수를 종료.
2. 게임 영역 지우기: 이전 프레임에서 그려진 **게임 영역을 지움**.
3. 프레임 번호 증가: 현재의 프레임 번호를 **1 증가시킵니다.**
4. 수직 플레이어 (Player 2) 이동: 만약 위쪽 화살표 키가 눌렸으면 Player 2(myVerticalPlayer)를 **위로 이동**시키고, 아래쪽 화살표 키가 눌렸으면 **아래로 이동**시킵니다.
5. 플레이어와 수직 플레이어 업데이트: Player 2 와 Player 1 위치를 업데이트하고, 그에 따라 그림을 그립니다.
6. 점수 업데이트: 현재의 점수를 업데이트하고, 이를 화면에 표시합니다.

➔ 이러한 단계들을 통해 updateGameArea 함수는 게임의 각 프레임에서 필요한 업데이트와 동작을 수행함.

코드 6 - 2

➔ 1. 일정 주기마다 장애물 생성

```
if (myGameArea.frameNo % 100 === 0) {
var x = myGameArea.canvas.width;
var minHeight = 20;
var maxHeight = 200;
var height = Math.floor(Math.random() * (maxHeight - minHeight + 1) + minHeight);
var minGap = 50;
var maxGap = 200;
var gap = Math.floor(Math.random() * (maxGap - minGap + 1) + minGap);
myObstacles.push(new component(10, height, "green", x, 0));
myObstacles.push(new component(10, x - height - gap, "green", x, height + gap));
}
```

➔ 2. 장애물 이동 및 화면에 그리기

```
for (var i = 0; i < myObstacles.length; i++) {
myObstacles[i].x -= 1;
myObstacles[i].update();
}
}
```

코드 6 - 2

1. 일정 주기마다 장애물 생성:

➔myGameArea.frameNo가 100의 배수일 때마다(100프레임마다), **새로운 장애물을 생성함**.

➔생성 위치와 크기는 **무작위로 설정**되며, myObstacles 배열에 **두 개의 장애물**을 추가함.

➔하나는 **위쪽에 생성**되고, 다른 하나는 **그 위에서 일정한 간격(gap)만큼 떨어진 위치**에 생성됨.

2. 장애물 이동 및 화면 그리기:

➔myObstacles 배열에 있는 **모든 장애물을 왼쪽**으로 이동시킴 **(myObstacles[i].x -= 1)**

➔그리고 각 장애물을 update 메서드를 **사용하여 화면에 그림**.

게임 스타트 버튼

```
<button onclick="startGame()">Start Game</button>
```

➔HTML에서 버튼 엘리먼트를 생성하고, 버튼이 클릭될 때 실행될 함수를 onclick 이벤트 핸들러로 설정.

➔여기서는 "Start Game" 텍스트를 가진 **버튼을 생성**하고, 클릭되면 **startGame() 함수를 호출**하도록 설정되어 있습니다.

출처

- https://chat.openai.com/c/f60b639c-e097-4b40-baf6-f308112f46de