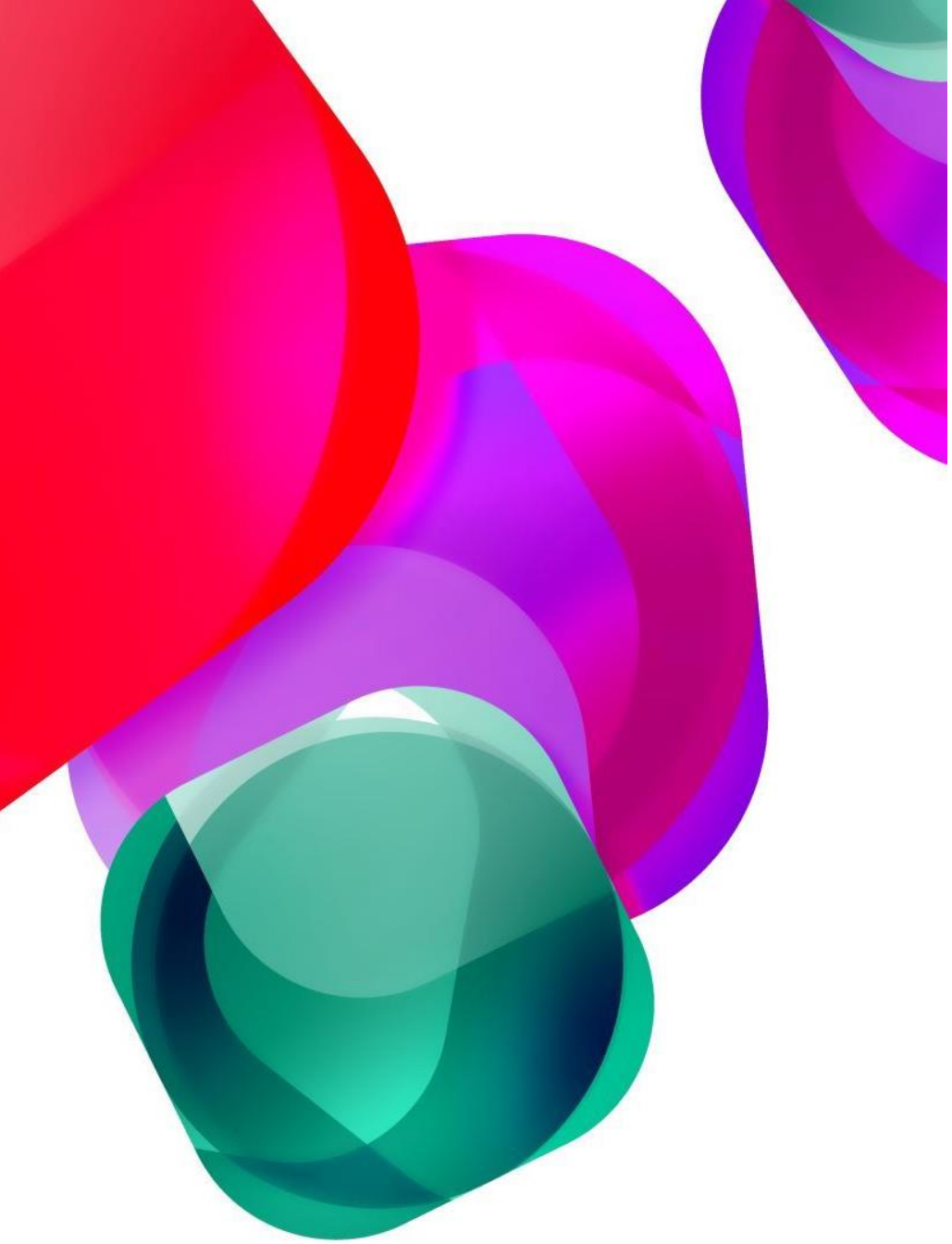
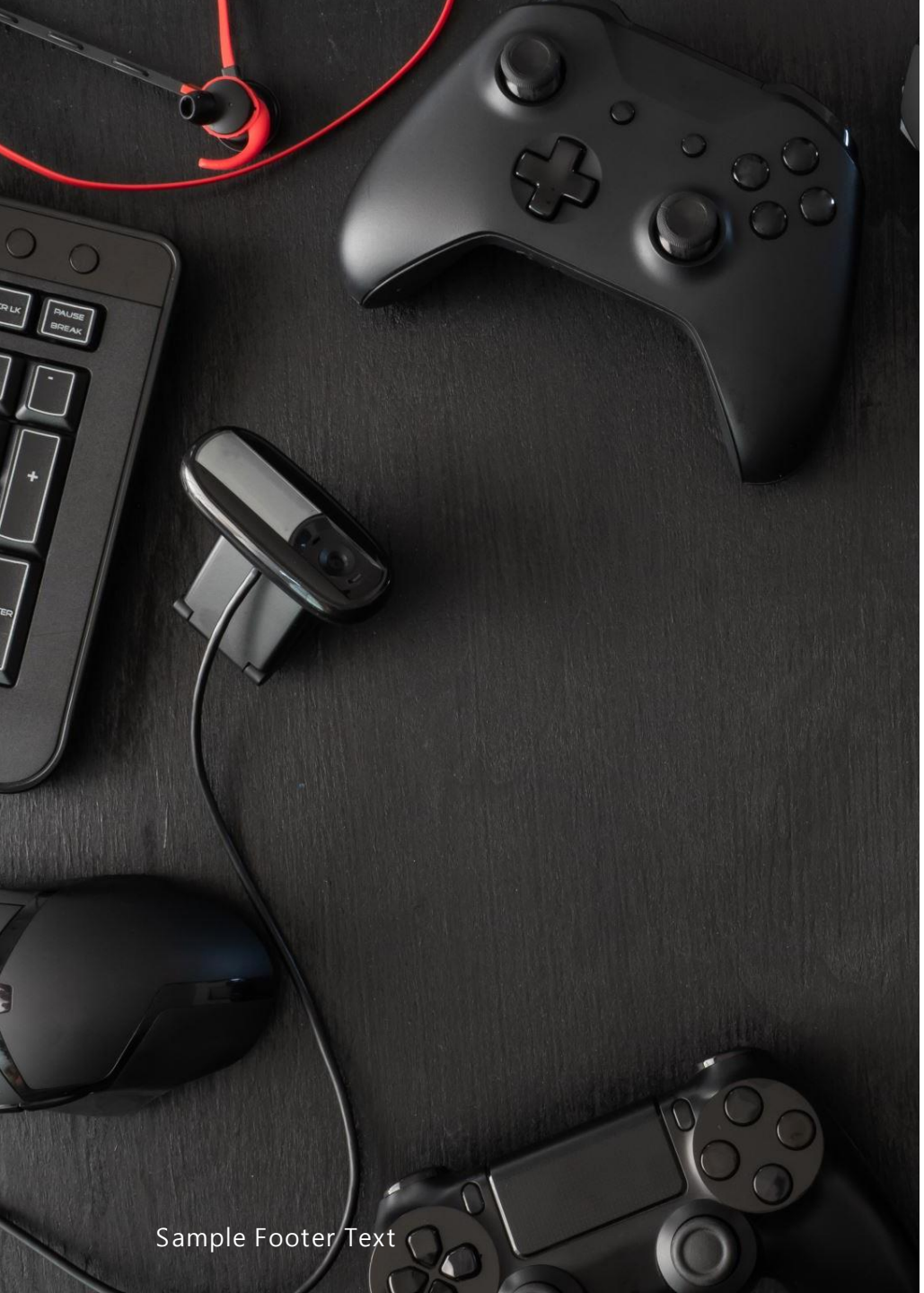


HTML 게임 분석

학번 : 2019975070

학과 : 소프트웨어학과

이름 : 한재훈

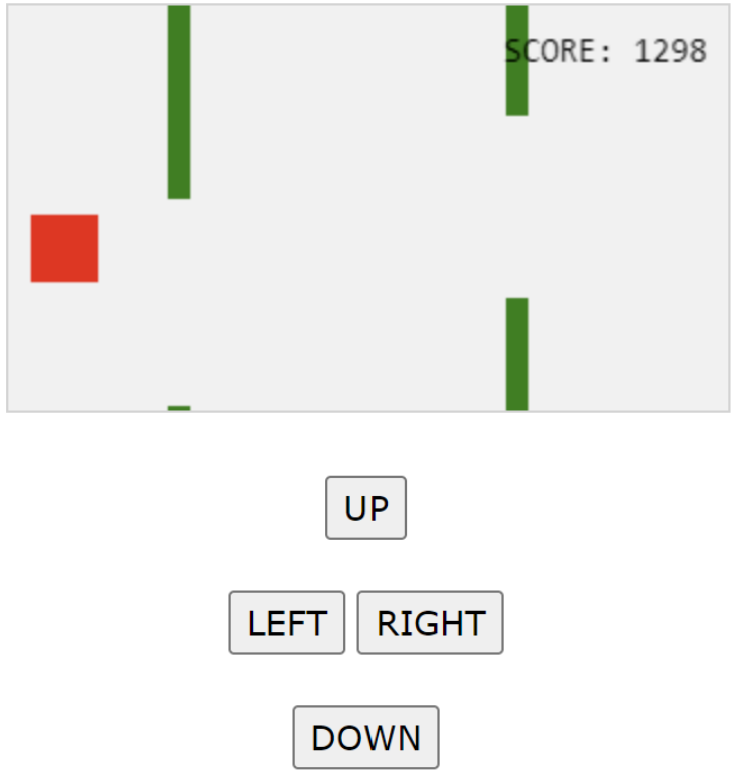


목차

- **게임 개요**
- **게임 구성 요소**
  - Canvas (캔버스)
  - Components (컴포넌트)
  - Controllers (컨트롤러)
  - Obstacles (장애물)
  - Score (점수)
  - Images (이미지 텍스처)
  - Sound (사운드드)
- **게임의 추가요소**
  - Bouncing (바운스) , Rotation(회전), Movement(이동)

게임 개요

- 이 게임은 초록색의 막대가 랜덤으로 생성되고 그러한 장애물을 컨트롤러(UP,LEFT,RIGHT,DOWN) 버튼을 통해 피하는 게임입니다. 점수는



게임 구성 요소 <Canvas> -개요-

- 먼저 게임의 화면을 보여주기 위해서는 <Canvas>라는 요소가 필요합니다. 우리는 이 <Canvas>요소를 이용하여 텍스트,도형,이미지등 그래픽적인 부분들을 표현할 수 있습니다. 그렇기에 게임이 시작할 때 우리는 먼저 <Canvas>를 생성하여 여러 개체들을 표현할 수 있게 준비를 해줍니다. 우리는 이 과정을 게임 화면을 초기화 하는 과정이라고 표현할 수 있습니다. 또한 이 부분에서 우리는 Start()라는 함수를 구현하는데 이는 Unity에서도 게임을 초기화하는 함수로 주로 쓰입니다.

게임 구성 요소 <Canvas> -code-

```
function startGame() {
  myGameArea.start();
}

var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
  }
}
```

게임 구성 요소 <Canvas> -분석-

1. StartGame() 함수 :

- `myGameArea.start()`를 호출하여 게임 엔진을 초기화합니다. 이 함수를 호출하면 게임 영역을 설정하고 캔버스를 생성하게 됩니다.

2. MyGame Area 객체 :

- **canvas** 속성: HTML의 **canvas** 엘리먼트를 동적으로 생성합니다.
- **start** 메서드: 게임 영역을 초기화하고 캔버스의 너비와 높이를 설정하며, 2D 그래픽 컨텍스트를 얻습니다. 그리고 이 캔버스를 HTML 문서의 맨 처음 자식 노드로 삽입합니다.

게임 구성 요소 <Components> -개요-

- <Components>란 말 그대로 게임의 구성요소라고 볼 수 있으며 속성이라고도 할 수 있습니다. 우리는 이 <Components>라는 요소를 통해 오브젝트의 위치나 색상 그리고 움직임등을 제어할 수 있습니다. 그리고 이러한 속성을 실시간으로 바꿔주기 위해서 우리는 update()라는 함수를 만들어줄겁니다. 매 프레임 맏마다 캔버스를 지우고 다시 그려주는 작업을 해주는 update()함수를 이용해 우리는 오브젝트의 움직임과 변화를 구현할 수 있습니다. Update()라는 함수는 우리가 unity를 통해 게임을 만들때도 자주 쓰이는 이벤트 메서드 이기에 이 예제를 통해 어떤 개념인지를 확실히 알고 가는게 좋을 것 같습니다.

게임 구성 요소 <Components> -code-

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.interval = setInterval(updateGameArea, 20);
  },
  clear : function() {
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

```
function component(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.x = x;
  this.y = y;
  this.update = function(){
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
}

function updateGameArea() {
  myGameArea.clear();
  myGamePiece.update();
}
```

게임 구성 요소 <Components> -분석-

1. myGameArea 객체:

- **canvas** 속성: HTML의 **canvas** 엘리먼트를 동적으로 생성합니다.
- **start** 메서드: 게임 영역을 초기화하고 캔버스의 너비와 높이를 설정하며, 2D 그래픽 컨텍스트를 얻습니다. 그리고 이 캔버스를 HTML 문서의 맨 처음 자식 노드로 삽입합니다.
- **interval** 속성: **setInterval**을 사용하여 **updateGameArea** 함수를 20 밀리초 간격으로 호출하는 것을 나타내며, 이로써 게임 루프를 시작합니다.

2. component 생성자 함수:

- 게임 객체를 나타내는데 사용됩니다. **width**, **height**, **color**, **x**, **y**를 인수로 받아 게임 객체의 속성을 초기화합니다.
- **update** 메서드: 게임 객체를 그리는데 사용되며, 현재 **myGameArea**의 2D 그래픽 컨텍스트를 얻고, 지정된 색상으로 사각형을 그립니다.

3. updateGameArea 함수:

- **myGameArea.clear()**를 호출하여 이전 프레임의 그래픽 내용을 모두 지우고, **myGamePiece.update()**를 호출하여 게임 객체를 새 위치에 그립니다.

게임 구성 요소 <Controllers> -개요-

- <Controllers> 요소 역시 게임의 중요한 요소 중 하나입니다. 플레이어는 지정된 <Controllers>의 키 값을 통해 오브젝트를 제어하거나 기능을 수행 할 수 있습니다. 이 예제에서는 <Controllers>를 통해 빨간색 사각형 오브젝트를 이동시킬 수 있습니다. <Controllers>의 key 속성은 키보드, 화면 터치, 마우스 커서, 캔버스 자체 버튼등 다양하게 표현될 수 있습니다. 또한 동시 입력을 통해 대각선을 이동하는 경우 역시 생각해 대각성 이동 부분 역시 구현될 수 있습니다.

게임 구성 요소 <Controllers> -code-

```
<script>
function component(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.speedX = 0;
  this.speedY = 0;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
  this.newPos = function() {
    this.x += this.speedX;
    this.y += this.speedY;
  }
}

function updateGameArea() {
  myGameArea.clear();
  myGamePiece.newPos();
  myGamePiece.update();
}
```

```
function updateGameArea() {
  myGameArea.clear();
  myGamePiece.newPos();
  myGamePiece.update();
}

function moveup() {
  myGamePiece.speedY -= 1;
}

function movedown() {
  myGamePiece.speedY += 1;
}

function moveleft() {
  myGamePiece.speedX -= 1;
}

function moveright() {
  myGamePiece.speedX += 1;
}
</script>

<button onclick="moveup()">UP</button>
<button onclick="movedown()">DOWN</button>
<button onclick="moveleft()">LEFT</button>
<button onclick="moveright()">RIGHT</button>
```

게임 구성 요소 <Controllers> -분석-

- **component** 생성자 함수:
  - **width**, **height**, **color**, **x**, **y**를 매개변수로 받아 게임 객체를 생성합니다.
  - **speedX** 및 **speedY** 속성은 가로 및 세로 방향의 속도를 나타냅니다.
  - **update** 메서드: 게임 객체를 화면에 그리기 위해 사용됩니다. Canvas 컨텍스트를 얻어 객체를 해당 위치와 크기에 그립니다.
  - **newPos** 메서드: 객체의 위치를 업데이트합니다. 현재 위치에 **speedX** 및 **speedY**를 더하여 객체를 이동시킵니다.
- **updateGameArea** 함수:
  - **myGameArea.clear()**를 호출하여 이전 프레임의 그래픽 내용을 지우고, **myGamePiece.newPos()**를 호출하여 게임 객체의 위치를 업데이트하고, **myGamePiece.update()**를 호출하여 객체를 그립니다.
- **moveup**, **movedown**, **moveleft**, **moveright** 함수:
  - 각각 "UP", "DOWN", "LEFT", "RIGHT" 버튼을 클릭했을 때 호출되는 함수로, **myGamePiece** 객체의 **speedX** 또는 **speedY** 값을 조정하여 객체를 위, 아래, 왼쪽, 오른쪽으로 움직입니다.
- HTML 버튼:
  - "UP", "DOWN", "LEFT", "RIGHT" 네 개의 버튼이 제공되며, 각 버튼을 클릭하면 해당 방향으로 **myGamePiece** 객체가 움직입니다.

게임 구성 요소 <Obstacles> -개요-

- 이제 게임에 <Obstacles> 즉 장애물 요소를 추가합니다. 장애물 역시 update()함수를 통해 매 프레임마다 위치를 이동시키며 플레이어 오브젝트와 충돌하면 게임이 멈추게끔 로직을 구성합니다. 이때 충돌 여부는 플레이어 컴포넌트에 추가합니다. 만약 장애물과 플레이어 오브젝트가 충돌하면 게임은 정지됩니다. 또한 정지 되있고 똑같은 장애물만 있다면 지루한 게임이 되기 때문에 장애물 오브젝트의 위치와 크기를 업데이트 될 때마다 바꿔줍니다.

게임 구성 요소 <Obstacles> -code-(충돌여부)

```
stop : function() {
  clearInterval(this.interval);
}
```

```
function updateGameArea() {
  if (myGamePiece.crashWith(myObstacle)) {
    myGameArea.stop();
  } else {
    myGameArea.clear();
    myObstacle.x += -1;
    myObstacle.update();
    myGamePiece.newPos();
    myGamePiece.update();
  }
}
```

```
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;
}
```

게임 구성 요소 <Obstacles> -분석-(충돌여부)

- **Stop 메서드 :**
  - 이 함수는 인자로 타이머 ID를 받아서 해당 타이머를 중지시킵니다. 장애물에 부딪혔을때 게임이 정지 되어야야함으로 myGameArea에 추가하여 줍니다.
- **CrashWith의 기능 :**
  - **this** 객체의 위치와 크기 정보를 변수에 저장합니다. **myleft**, **myright**, **mytop**, **mybottom** 변수는 **this** 객체의 왼쪽, 오른쪽, 위, 아래 경계를 나타냅니다.
  - **otherobj** 객체의 위치와 크기 정보를 변수에 저장합니다. **otherleft**, **otherright**, **othertop**, **otherbottom** 변수는 **otherobj** 객체의 왼쪽, 오른쪽, 위, 아래 경계를 나타냅니다.
  - **crash** 변수를 **true**로 초기화합니다. 이 변수는 두 객체가 충돌했는지를 나타내는 플래그입니다.
  - **if** 조건문을 사용하여 두 객체의 경계를 비교합니다. 만약 두 객체가 서로 겹치지 않으면 **crash**를 **false**로 설정합니다.
  - 최종적으로 **crash** 변수를 반환하여 두 객체가 충돌한 경우 **true**를, 그렇지 않은 경우 **false**를 반환합니다.
- **Update() 함수에 장애물 이동기능 추가 :**
  - 매 프레임마다 장애물이 정지되있는게 아닌 왼쪽으로 이동하여야 하기 때문에 Update 부문에 x좌표가 1씩 감소하게 만들어줍니다.

게임 구성 요소 <Obstacles> -code-(여러장애물)

```
var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.frameNo = 0;
    this.interval = setInterval(updateGameArea, 20);
  },
  clear : function() {
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  },
  stop : function() {
    clearInterval(this.interval);
  }
}

function everyinterval(n) {
  if ((myGameArea.frameNo / n) % 1 == 0) {return true;}
  return false;
}
```

```
var myGamePiece;
var myObstacles = [];

function updateGameArea() {
  var x, y;
  for (i = 0; i < myObstacles.length; i += 1) {
    if (myGamePiece.crashWith(myObstacles[i])) {
      myGameArea.stop();
      return;
    }
  }
  myGameArea.clear();
  myGameArea.frameNo += 1;
  if (myGameArea.frameNo == 1 || everyinterval(150)) {
    x = myGameArea.canvas.width;
    y = myGameArea.canvas.height - 200
    myObstacles.push(new component(10, 200, "green", x, y));
  }
  for (i = 0; i < myObstacles.length; i += 1) {
    myObstacles[i].x += -1;
    myObstacles[i].update();
  }
  myGamePiece.newPos();
  myGamePiece.update();
}
```

게임 구성 요소 <Obstacles> -분석-(여러장애물)

- `everyinterval(n)` 함수:
  - `myGameArea.frameNo`를 `n`으로 나눴을 때 나머지가 `0`이면 `true`를 반환하고, 그렇지 않으면 `false`를 반환하는 함수입니다. 이 함수는 일정한 간격으로 무언가를 수행하려는 경우 유용하게 사용될 수 있습니다. `myGameArea.frameNo`는 현재 프레임 번호를 나타내며, 일반적으로 게임 루프에서 사용됩니다.
- `myGamePiece` 및 `myObstacles` 변수:
  - `myGamePiece`: 게임에서 제어할 주요 캐릭터 또는 요소를 나타내는 변수입니다. 이 변수는 게임에서 주로 사용됩니다.
  - `myObstacles`: 게임에서 생성된 장애물을 저장하는 배열입니다. 게임에 표시되는 여러 장애물을 관리하는 데 사용됩니다.
- `updateGameArea` 함수:
  - `for` 루프를 사용하여 `myObstacles` 배열을 반복하며, `myGamePiece`가 어떤 장애물과 충돌했는지를 검사합니다. 만약 충돌이 감지되면 `myGameArea.stop()`을 호출하여 게임을 중지하고 함수를 빠져나갑니다.
  - `myGameArea.clear()`: 이전 프레임의 그래픽 내용을 지워서 새 프레임을 그리기 전에 화면을 지웁니다.
  - `myGameArea.frameNo`를 증가시킵니다. 이 변수는 현재 프레임 번호를 나타내며 게임 루프에서 시간 경과를 추적하는 데 사용됩니다.
  - `myGameArea.frameNo`이 `1`이거나 일정한 간격 (여기서는 `150`)으로 호출되는 `everyinterval(150)`이 `true`를 반환하는 경우, 새 장애물을 생성하고 `myObstacles` 배열에 추가합니다.
  - 장애물 배열을 반복하며, 각 장애물의 `x` 좌표를 감소시키고 `myObstacles[i].update()`를 호출하여 장애물을 새 위치에 그립니다.
  - `myGamePiece.newPos()`를 호출하여 `myGamePiece`의 위치를 업데이트하고, 그 후 `myGamePiece.update()`를 호출하여 새 위치에 캐릭터를 그립니다.

게임 구성 요소 <Obstacles> -code-(무작위크기)

```
if (myGameArea.frameNo == 1 || everyinterval(150)) {
  x = myGameArea.canvas.width;
  minHeight = 20;
  maxHeight = 200;
  height = Math.floor(Math.random()*(maxHeight-minHeight+1)+minHeight);
  minGap = 50;
  maxGap = 200;
  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));
}
```

게임 구성 요소 <Obstacles> -분석-(무작위크기)

- 이 코드 조각은 게임 루프 내에서 새로운 장애물을 생성하고 **myObstacles** 배열에 추가하는 부분입니다. 아래는 코드의 주요 부분에 대한 설명입니다:
- **if (myGameArea.frameNo == 1 || everyinterval(150))**: 이 조건문은 프레임 번호가 1이거나 일정한 간격(여기서는 150)으로 **everyinterval(150)**이 **true**를 반환할 때 새로운 장애물을 생성하도록 하는 조건입니다. 이렇게하면 게임에서 일정한 시간 간격으로 새로운 장애물이 생성됩니다.
- 변수 설정:
  - **x**: 새로운 장애물의 x 좌표를 설정합니다. **myGameArea.canvas.width**는 캔버스의 가로 너비입니다. 따라서 새 장애물은 오른쪽에서 시작하게 됩니다.
  - **minHeight** 및 **maxHeight**: 새로운 장애물의 높이 범위를 정의합니다. **minHeight**는 장애물의 최소 높이이고, **maxHeight**는 최대 높이입니다.
  - **height**: 장애물의 높이를 무작위로 생성합니다. **Math.random()**을 사용하여 **minHeight**와 **maxHeight** 사이의 임의의 값으로 **height**를 설정합니다.
  - **minGap** 및 **maxGap**: 장애물 사이의 간격을 정의합니다. **minGap**은 최소 간격이고, **maxGap**은 최대 간격입니다.
  - **gap**: 장애물 사이의 간격을 무작위로 생성합니다. **Math.random()**을 사용하여 **minGap**과 **maxGap** 사이의 임의의 값으로 **gap**을 설정합니다.
- **myObstacles.push(new component(10, height, "green", x, 0));** 및 **myObstacles.push(new component(10, x - height - gap, "green", x, height + gap));**: 두 개의 장애물을 **myObstacles** 배열에 추가합니다. 각각의 장애물은 **new component(10, height, "green", x, 0)**와 **new component(10, x - height - gap, "green", x, height + gap)**으로 생성됩니다. 각 장애물은 높이가 **height**이고, 폭(두께)가 10인 초록색의 컴포넌트입니다. 첫 번째 장애물은 캔버스의 맨 위에서 시작하고, 두 번째 장애물은 첫 번째 장애물 아래의 간격 **gap**만큼 떨어진 위치에서 시작합니다.

게임 구성 요소 <Score> -개요-

- 장애물을 피하는 게임에는 <Score>라는 점수 시스템 요소가 필요합니다. 점수를 올리는 방식은 여러가지가 있습니다. 특정 아이템을 먹거나 시간 혹은 거리에 따라 점수를 올리는 방식등이 있고 이 게임에는 시간에 따라 점수를 계산합니다. 점수 표시는 <Canvas> 요소의 텍스트를 이용하여 나타냅니다. 그렇기에 생성자 함수에서 "텍스트" 유형인지 확인하는 메소드인 fillRect 함수를 이용합니다. 또한 실시간으로 점수가 올라가야 함으로 update()함수에 점수를 실시간으로 계산하는 기능 역시 추가하여 줍니다.

게임 구성 요소 <Score> -code-

```
function component(width, height, color, x, y, type) {
  this.type = type;
  this.width = width;
  this.height = height;
  this.speedX = 0;
  this.speedY = 0;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    if (this.type == "text") {
      ctx.font = this.width + " " + this.height;
      ctx.fillStyle = color;
      ctx.fillText(this.text, this.x, this.y);
    } else {
      ctx.fillStyle = color;
      ctx.fillRect(this.x, this.y, this.width, this.height);
    }
  }
...
}
```

```
myGameArea.clear();
myGameArea.frameNo += 1;
if (myGameArea.frameNo == 1 || everyinterval(150)) {
  x = myGameArea.canvas.width;
  minHeight = 20;
  maxHeight = 200;
  height = Math.floor(Math.random()*(maxHeight-minHeight+1)+minHeight);
  minGap = 50;
  maxGap = 200;
  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));
}
for (i = 0; i < myObstacles.length; i += 1) {
  myObstacles[i].speedX = -1;
  myObstacles[i].newPos();
  myObstacles[i].update();
}
myScore.text = "SCORE: " + myGameArea.frameNo;
myScore.update();
myGamePiece.newPos();
myGamePiece.update();
```

게임 구성 요소 <Score> -분석-(1)

- **매개변수 type 추가** :
  - 이 객체가 Text type인지 아닌지를 판별하기 위해 컴포넌트 생성자 함수의 매개변수에 type인자를 추가합니다.
- **this.update** 함수: 이 객체의 **update** 메서드는 객체를 렌더링하는 역할을 합니다.
  - **ctx = myGameArea.context**: **ctx** 변수를 사용하여 2D 그래픽 컨텍스트를 가져옵니다. 이 컨텍스트를 사용하여 객체를 화면에 그립니다.
  - **if (this.type == "text")**: 객체의 유형이 "text"인 경우 텍스트를 렌더링합니다.
    - **ctx.font**: 텍스트의 폰트 크기와 스타일을 설정합니다.
    - **ctx.fillStyle**: 텍스트의 색상을 설정합니다.
    - **ctx.fillText**: 지정된 텍스트를 해당 좌표에 그립니다.
  - 그렇지 않은 경우, 객체의 유형이 "text"가 아니라면, 사각형을 그립니다.
    - **ctx.fillStyle**: 사각형의 색상을 설정합니다.
    - **ctx.fillRect**: 지정된 좌표와 크기로 사각형을 그립니다.

게임 구성 요소 <Score> -분석-(2)

- **게임루프 객체에 스코어 점수** :
  - **MyGameArea.frameNo의 값을 score에 축적하고 Update하는 코드를 추가합니다. 이렇게 되면 매 프레임마다 점수가 실시간으로 변화되는것을 확인할 수 있습니다.**

게임 구성 요소 <Images> -개요-

- 단순히 도형으로 플레이어 오브젝트를 표현하기에는 심심한 부분이 있습니다. 이때 getContext("2d") 개체의 이미지 속성과 메소드를 통해 플레이어 오브젝트를 이미지로 표현할 수 있습니다. 오브젝트를 생성할 때 색상 참조 대신 이미지의 URL을 참조하고 이 구성요소가 "이미지" 유형임을 알립니다. 이미지 유형임을 알면 "drawImage"함수를 통해 이미지를 그려줍니다.

게임 구성 요소 <Images> -code-

```
function component(width, height, color, x, y, type) {
  this.type = type;
  if (type == "image") {
    this.image = new Image();
    this.image.src = color;
  }
  this.width = width;
  this.height = height;
  this.speedX = 0;
  this.speedY = 0;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    if (type == "image") {
      ctx.drawImage(this.image,
        this.x,
        this.y,
        this.width, this.height);
    } else {
      ctx.fillStyle = color;
      ctx.fillRect(this.x, this.y, this.width, this.height);
    }
  }
}
```

게임 구성 요소 <Images> -분석-

- **`if (type == "image")`**: 객체의 유형이 "image"일 때 이미지를 사용할 것이라고 판별합니다.
  - **`this.image = new Image()`**: 이미지 객체를 생성합니다.
  - **`this.image.src = color`**: **`color`** 매개변수에 지정된 이미지 파일의 경로를 설정합니다.
- **`this.update`** 함수: 이 객체의 **`update`** 메서드는 객체를 렌더링하는 역할을 합니다.
  - **`ctx = myGameArea.context`**: **`ctx`** 변수를 사용하여 2D 그래픽 컨텍스트를 가져옵니다. 이 컨텍스트를 사용하여 객체를 화면에 그립니다.
  - **`if (type == "image")`**: 객체의 유형이 "image"인 경우 이미지를 그립니다.
    - **`ctx.drawImage(this.image, this.x, this.y, this.width, this.height)`**: **`drawImage`** 메서드를 사용하여 이미지를 지정된 좌표와 크기로 그립니다.
  - 그렇지 않은 경우, 객체의 유형이 "image"가 아니라면, 사각형을 그립니다.
    - **`ctx.fillStyle = color`**: 사각형의 색상을 설정합니다.
    - **`ctx.fillRect(this.x, this.y, this.width, this.height)`**: 지정된 좌표와 크기로 사각형을 그립니다.

게임 구성 요소 <Sound> -개요-

- 게임에서의 <Sound> 요소 역시 게임의 중요한 요소중 하나입니다. <Sound>는 특정 액션을 했을때 음향 효과와 배경음악등으로 구성될 수 있고 이 게임에서 우리는 배경음악과 플레이어 오브젝트가 장애물과 부딪혔을 때 나오는 음향효과를 추가할 수 있습니다.

게임 구성 요소 <Sound> -code-

```
function sound(src) {
  this.sound = document.createElement("audio");
  this.sound.src = src;
  this.sound.setAttribute("preload", "auto");
  this.sound.setAttribute("controls", "none");
  this.sound.style.display = "none";
  document.body.appendChild(this.sound);
  this.play = function(){
    this.sound.play();
  }
  this.stop = function(){
    this.sound.pause();
  }
}
```

게임 구성 요소 <Sound> -분석-

- **function sound(src)**: 이 함수는 **src** 매개변수로 전달된 오디오 파일 경로를 사용하여 소리 객체를 생성합니다.
- **this.sound = document.createElement("audio");**: **audio** 엘리먼트를 동적으로 생성하여 소리를 표현할 객체를 만듭니다.
- **this.sound.src = src;**: 소리 객체의 **src** 속성을 설정하여 소리 파일의 경로를 지정합니다.
- **this.sound.setAttribute("preload", "auto");**와 **this.sound.setAttribute("controls", "none");**: **preload**와 **controls** 속성을 설정하여 소리를 자동으로 미리 로드하도록 하고, 웹 페이지에 기본적인 오디오 컨트롤 버튼을 표시하지 않도록 설정합니다.
- **this.sound.style.display = "none";**: 소리 객체의 **display** 스타일을 "none"으로 설정하여 화면에서 숨깁니다.
- **document.body.appendChild(this.sound);**: 소리 객체를 현재 웹 페이지의 **body** 엘리먼트에 추가합니다. 이렇게하면 소리 객체가 화면에서 실제로 표시되지는 않지만 사용 가능한 상태가 됩니다.
- **this.play = function(){ this.sound.play(); }**: **play** 메서드를 정의하여 소리를 재생합니다. 이 메서드를 호출하면 소리가 재생됩니다.
- **this.stop = function(){ this.sound.pause(); }**: **stop** 메서드를 정의하여 소리를 일시 중지합니다. 이 메서드를 호출하면 소리 재생이 중지됩니다.
- 이 **sound** 함수를 사용하면 웹 페이지에서 소리 파일을 동적으로 추가하고 재생, 일시 중지할 수 있으며, 웹 페이지에 오디오 컨트롤을 표시하지 않고 숨길 수 있습니다. 이는 웹 애플리케이션에서 오디오 효과를 통제하거나 게임에서 효과음을 다루는 데 유용할 수 있습니다.

게임의 추가요소

- 중력(Gravity)
- 바운싱(bouncing)
- 회전(Rotation)
- 움직임설정(Movement)

게임 구성 요소 <Gravity> -개요-

- 대부분의 횡스크롤형 2D 게임에는 "중력"이라는 요소가 들어갑니다. 이러한 중력이라는는 요소를 추가함으로써 우리는 점프 액션 역시 구현할 수 있고, 이러한 중력을 구현하기 위해서는 컴포넌트 생성자 부문에서 중력의 세기를 설정해 줄 수 있습니다.

게임 구성 요소 <Gravity> -code-

```
function component(width, height, color, x, y, type) {
  this.type = type;
  this.width = width;
  this.height = height;
  this.x = x;
  this.y = y;
  this.speedX = 0;
  this.speedY = 0;
  this.gravity = 0.05;
  this.gravitySpeed = 0;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
  this.newPos = function() {
    this.gravitySpeed += this.gravity;
    this.x += this.speedX;
    this.y += this.speedY + this.gravitySpeed;
  }
}
```

게임 구성 요소 <Gravity> -분석-

- **this.speedX**와 **this.speedY**: 객체의 가로 및 세로 방향 속도를 나타내는 속성입니다.
- **this.gravity**와 **this.gravitySpeed**: 중력을 나타내는 속성입니다. **gravity**는 중력의 세기를 나타내며, **gravitySpeed**는 중력에 따른 속도의 누적을 나타냅니다.
- **this.newPos** 함수: 이 객체의 **newPos** 메서드는 객체의 새 위치를 계산하는 역할을 합니다.
- **this.gravitySpeed**를 중력에 따른 속도의 누적값으로 업데이트합니다.
- **this.x**를 현재 **x** 좌표와 가로 방향 속도 **this.speedX**를 합하여 업데이트합니다.
- **this.y**를 현재 **y** 좌표와 세로 방향 속도 **this.speedY**와 중력에 따른 속도 **this.gravitySpeed**를 합하여 업데이트합니다.

게임 구성 요소 <Bouncing> -개요-

- 중력으로 플레이어 오브젝트가 떨어졌을때 아무 반응 없이 떨어지면 무언가 심심합니다. 그렇기에 우리는 여기서 "바운스"라는 속성을을 추가할 수 있습니다. 바운스를 구현하기 위해서는 플레이어가 바닥에 부딪혔는지를 판단하는 기능과 바닥에 부딪혔을때 튕기는 효과 두가지를 구현하여야 합니다. 만약 점프 게임을 구현할 때 어떠한 물체와 부딧혔을때 튕기는 효과를 이를 이용하여 구현가능합니다.

게임 구성 요소 <Bouncing> -code-

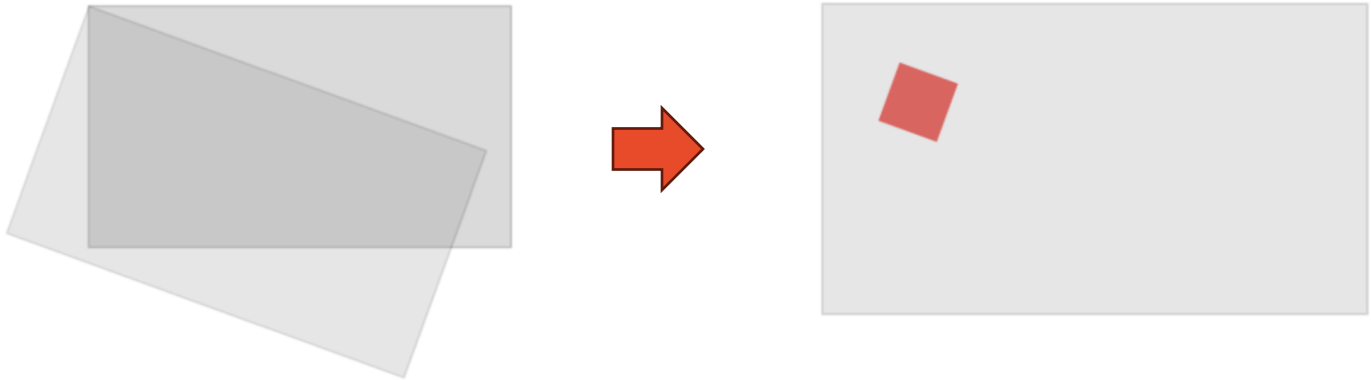
```
function component(width, height, color, x, y, type) {
  this.type = type;
  this.width = width;
  this.height = height;
  this.x = x;
  this.y = y;
  this.speedX = 0;
  this.speedY = 0;
  this.gravity = 0.1;
  this.gravitySpeed = 0;
  this.bounce = 0.6;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
  }
  this.newPos = function() {
    this.gravitySpeed += this.gravity;
    this.x += this.speedX;
    this.y += this.speedY + this.gravitySpeed;
    this.hitBottom();
  }
  this.hitBottom = function() {
    var rockbottom = this.gamearea.canvas.height - this.height;
    if (this.y > rockbottom) {
      this.y = rockbottom;
      this.gravitySpeed = -(this.gravitySpeed * this.bounce);
    }
  }
}
```

게임 구성 요소 <Bouncing> -분석-

- **this.bounce**: 바운스(튕김) 효과를 나타내는 속성으로, 물체가 바닥에 닿았을 때 튕겨올라가는 양을 조절합니다.
- **this.newPos** 함수: 이 객체의 **newPos** 메서드는 객체의 새 위치를 계산하는 역할을 합니다.
  - **this.gravitySpeed**를 중력에 따른 속도의 누적값으로 업데이트합니다.
  - **this.x**를 현재 **x** 좌표와 가로 방향 속도 **this.speedX**를 합하여 업데이트합니다.
  - **this.y**를 현재 **y** 좌표와 세로 방향 속도 **this.speedY**와 중력에 따른 속도 **this.gravitySpeed**를 합하여 업데이트합니다.
  - **this.hitBottom** 메서드를 호출하여 객체가 바닥에 도달하면 튕김 효과를 적용합니다.
- **this.hitBottom** 함수: 이 함수는 객체가 화면 하단에 닿았을 때 튕김 효과를 적용합니다.
  - **rockbottom** 변수에는 화면 하단의 y 좌표가 저장됩니다.
  - 만약 객체의 **y** 좌표가 **rockbottom**을 초과하면, **this.y**를 **rockbottom**으로 설정하여 객체가 바닥에 닿았을 때의 위치를 정의합니다.
  - 또한, **this.gravitySpeed**를 현재 속도의 반대 방향으로 설정하고, **this.bounce**를 곱하여 튕김 효과를 적용합니다.

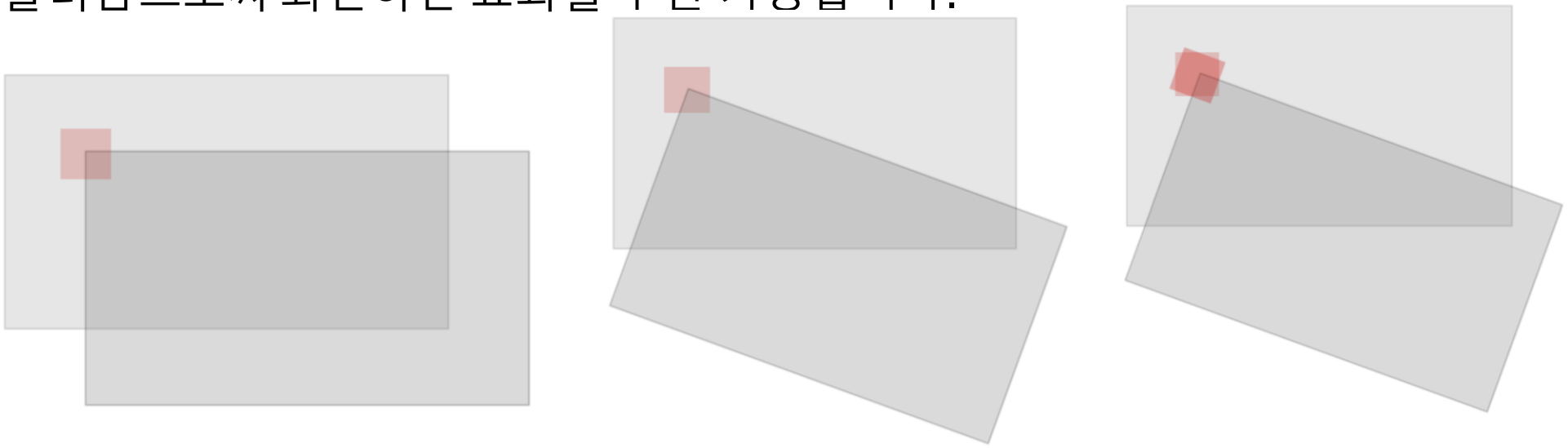
게임 구성 요소 <Rotation> -분석(1)-

- 게임에서의 플레이어 오브젝트는 항상 멈춰있는 모습만 보이기에 우리는 여기서 회전이라는 속성을 추가할 수 있습니다. 회전 요소를 넣기위해서는 우리는 캔버스를 그리는 방식을 바꾸어주어야 합니다.



게임 구성 요소 <Rotation> -분석(2)-

- 우선 현재의 캔버스 컨텍스트를 저장하고 위치를 옮겨준뒤 회전을 시켜줍니다. 그런 다음 플레이어 객체 즉 빨간색 네모를 다시 그려주고 그래픽 컨텍스트를 다시 이전 상태로 되돌려줌으로써 회전하는 효과를 구현 가능합니다.



게임 구성 요소 <Rotation> -code-

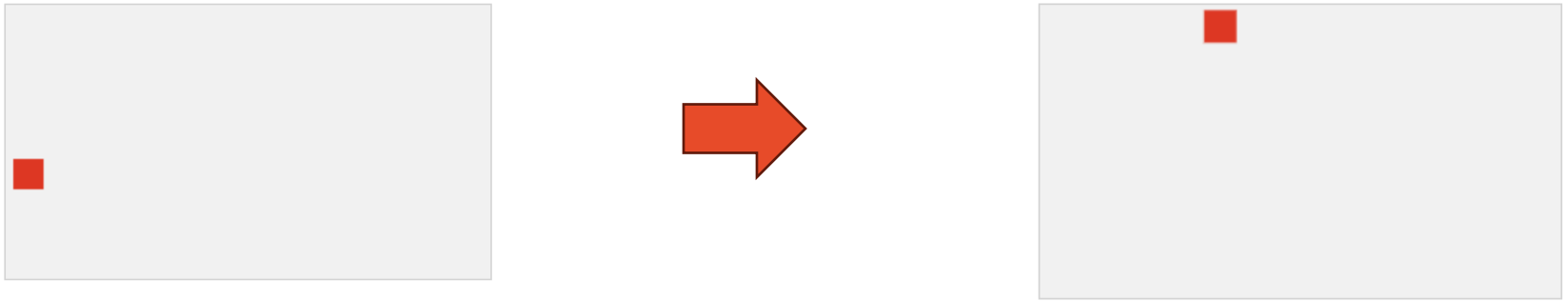
```
function component(width, height, color, x, y) {
  this.width = width;
  this.height = height;
  this.angle = 0;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.save();
    ctx.translate(this.x, this.y);
    ctx.rotate(this.angle);
    ctx.fillStyle = color;
    ctx.fillRect(this.width / -2, this.height / -2, this.width, this.height);
    ctx.restore();
  }
}

function updateGameArea() {
  myGameArea.clear();
  myGamePiece.angle += 1 * Math.PI / 180;
  myGamePiece.update();
}
```

게임 구성 요소 <Rotation> -분석-

- **`this.angle = 0;`**: 객체의 회전 각도를 나타내는 속성으로, 초기값은 0입니다.
- **`this.update`** 함수: 이 객체의 **`update`** 메서드는 객체를 렌더링하고 회전하는 역할을 합니다.
  - **`ctx = myGameArea.context`**: 2D 그래픽 컨텍스트를 가져옵니다.
  - **`ctx.save()`**: 그래픽 컨텍스트의 현재 상태를 저장합니다.
  - **`ctx.translate(this.x, this.y)`**: 객체의 중심으로 이동합니다.
  - **`ctx.rotate(this.angle)`**: **`this.angle`**에 저장된 각도에 따라 객체를 회전시킵니다.
  - **`ctx.fillStyle = color`**: 객체의 색상을 설정합니다.
  - **`ctx.fillRect(this.width / -2, this.height / -2, this.width, this.height)`**: 회전된 객체를 그립니다. 중심으로 이동한 후 사각형을 그리므로 중심을 중심으로 회전됩니다.
  - **`ctx.restore()`**: 그래픽 컨텍스트의 이전 상태로 돌아갑니다.
- **`function updateGameArea()`**: 이 함수는 게임 루프 내에서 호출되며 게임 객체를 업데이트하고 렌더링합니다.
  - **`myGameArea.clear()`**: 이전 프레임을 지우고 새로운 프레임을 그리기 위해 캔버스를 지웁니다.
  - **`myGamePiece.angle += 1 * Math.PI / 180;`**: **`myGamePiece`** 객체의 **`angle`** 값을 1도만큼 증가시켜 객체를 회전시킵니다. 이 부분은 게임 루프마다 객체가 조금씩 회전하도록 합니다.
  - **`myGamePiece.update()`**: **`myGamePiece`** 객체의 **`update`** 메서드를 호출하여 회전된 객체를 그립니다.

게임 구성 요소 <Movement> -분석-

- 플레이어의 Movement는 게임에서 중요한 요소중 하나 입니다. Speed값을 하나 추가하여 스피드 값과 각도의 값을 이용하여 컴포넌트를를 움직일 수 있습니다. 이는 게임의 특수효과나 시뮬레이션에 이용할 수 있고 다양한 동적 요소를 생성가능합니다.



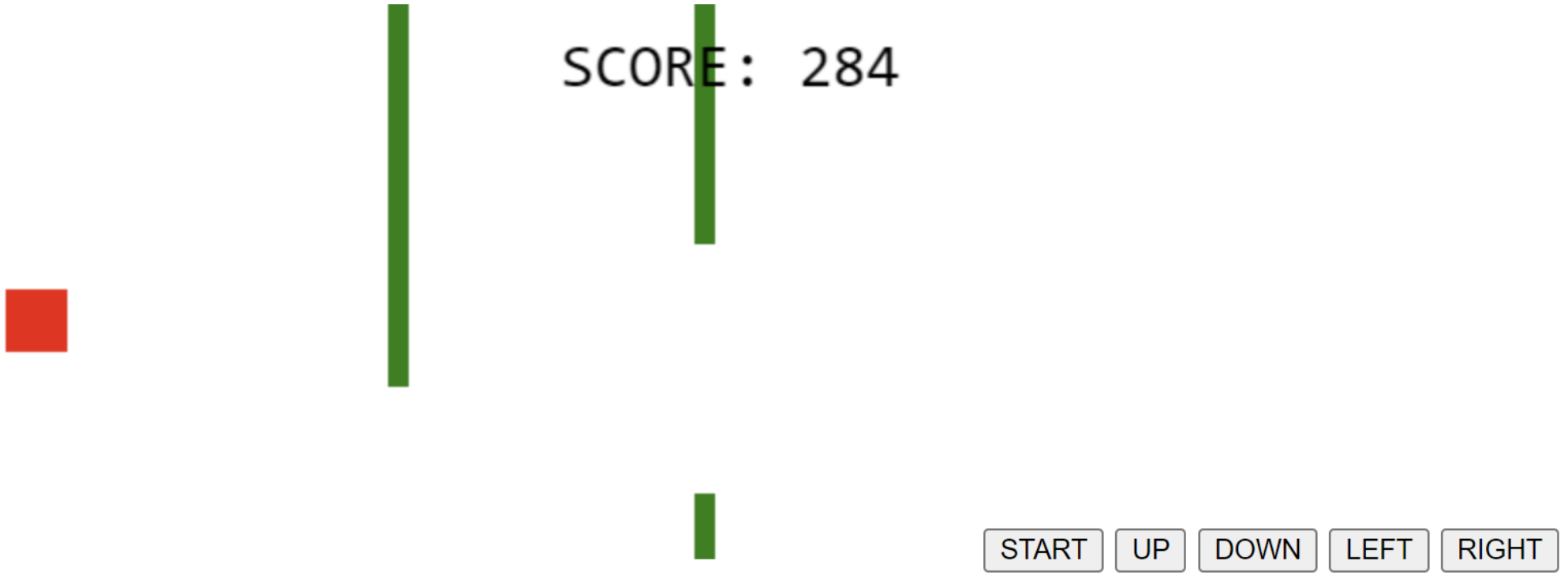
게임 구성 요소 <Movement> -code-

```
function component(width, height, color, x, y) {
  this.gamearea = gamearea;
  this.width = width;
  this.height = height;
  this.angle = 0;
  this.speed = 1;
  this.x = x;
  this.y = y;
  this.update = function() {
    ctx = myGameArea.context;
    ctx.save();
    ctx.translate(this.x, this.y);
    ctx.rotate(this.angle);
    ctx.fillStyle = color;
    ctx.fillRect(this.width / -2, this.height / -2, this.width, this.height);
    ctx.restore();
  }
  this.newPos = function() {
    this.x += this.speed * Math.sin(this.angle);
    this.y -= this.speed * Math.cos(this.angle);
  }
}
```

게임 구성 요소 <Movement> -분석-

- **this.angle = 0;**: 객체의 회전 각도를 나타내는 속성으로, 초기값은 0도입니다.
- **this.speed = 1;**: 객체의 이동 속도를 나타내는 속성으로, 초기값은 1입니다.
- **this.update** 함수: 이 객체의 **update** 메서드는 객체를 렌더링하고 회전하는 역할을 합니다. 코드 내용은 이전 설명과 동일합니다.
  - **ctx = myGameArea.context**: 2D 그래픽 컨텍스트를 가져옵니다.
  - **ctx.save()**: 그래픽 컨텍스트의 현재 상태를 저장합니다.
  - **ctx.translate(this.x, this.y)**: 객체의 중심으로 이동합니다.
  - **ctx.rotate(this.angle)**: **this.angle**에 저장된 각도에 따라 객체를 회전시킵니다.
  - **ctx.fillStyle = color**: 객체의 색상을 설정합니다.
  - **ctx.fillRect(this.width / -2, this.height / -2, this.width, this height)**: 회전된 객체를 그립니다. 중심으로 이동한 후 사각형을 그리므로 중심을 중심으로 회전됩니다.
  - **ctx.restore()**: 그래픽 컨텍스트의 이전 상태로 돌아갑니다.
- **this.newPos** 함수: 이 객체의 **newPos** 메서드는 객체의 새 위치를 계산하는 역할을 합니다.
  - **this.x**와 **this.y** 값을 이동 방향과 속도에 따라 업데이트합니다.
  - **Math.sin(this.angle)** 및 **Math.cos(this.angle)**를 사용하여 현재 각도에 따라 이동합니다. 이를 통해 객체는 현재 각도를 고려하여 회전 및 이동합니다.

구성요소를 종합하여 만든 최종본



추가 구성 요소를 통해 업그레이드 할 수 있는부분

- Gravity와 Bouncing 기능을 통해 점프 요소를 추가할 수 있습니다.
- Movement와 Rotation의 기능을 통해 더욱 더 어려운 장애물들을 추가 할 수 있습니다.
- 또한 먹으면 점수가 오르는 아이템 오브젝트 역시 구현 가능합니다.
- Image 기능을 통해 Player 오브젝트의 Material을 변경 가능합니다.
- 종합적으로 충돌의 기능과 여러 요소를 조합하면 간단한 2D 횡스크롤 게임을 구현 가능합니다.

감사합니다.