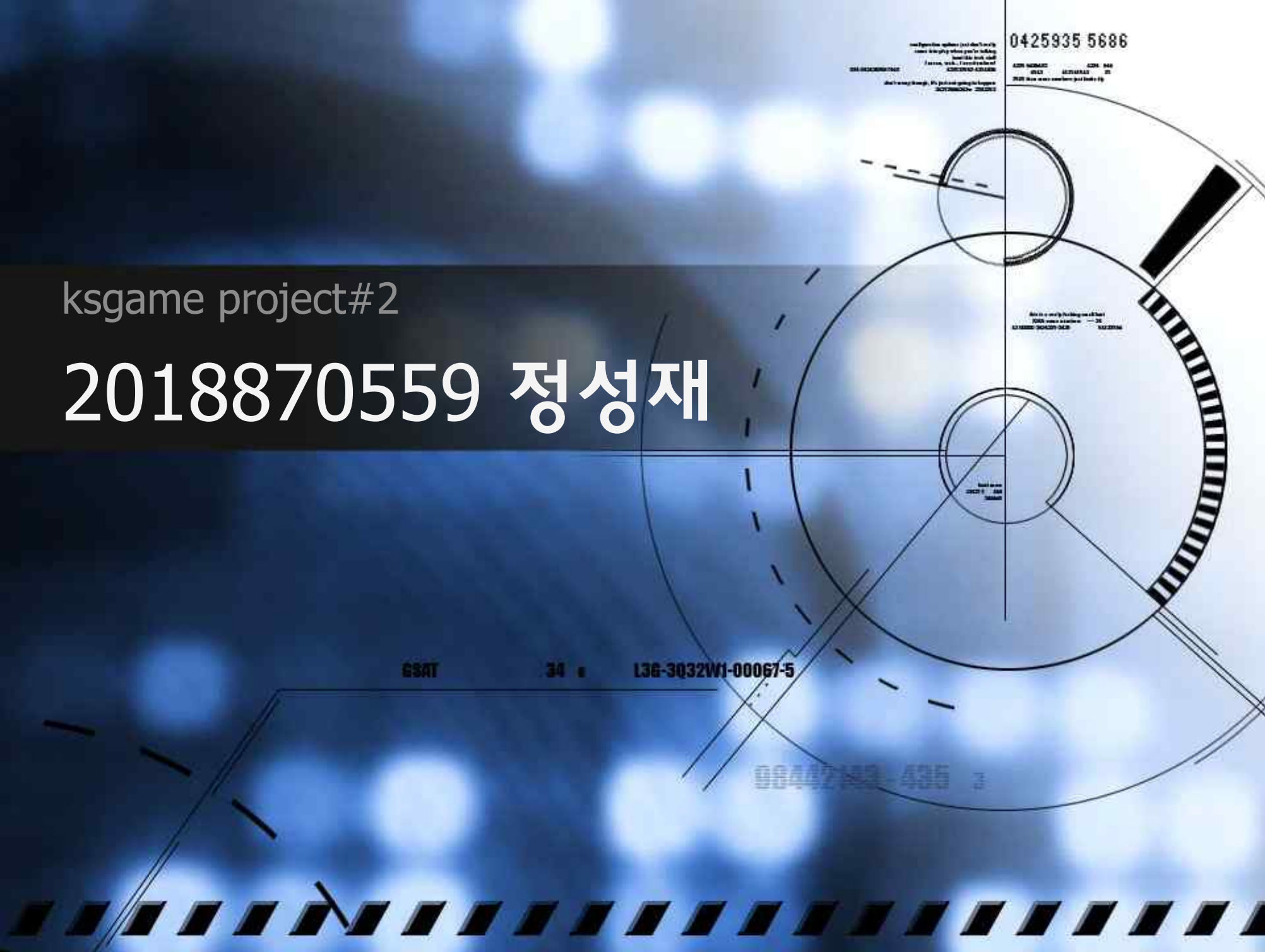

ksgame project#2

2018870559 정성재

선택한 source code와 코드 수정 방법

- canvas game-fortress
- 기능 추가후 해당 기능 업그레이드 방식 사용

초기 code

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>fortress</title>
    <style>
      body {
        margin: 0;
        height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
      }
      #fortress {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <canvas id="fortress" width="1000px" height="700px"></canvas>
    <script>
      const canvas = document.getElementById("fortress");
      const ctx = canvas.getContext("2d");
      const width = canvas.width;
      const height = canvas.height;
      const tankWidth = 50;
      const tankHeight = 50;
      let tankX = 0;
      const tankDx = 3;
      let tankLeftPressed = false;
      let tankRightPressed = false;
      let tankCenterX;
      let tankCenterY;
      let cannonAngle = Math.PI / 4;
      const cannonAngleDIF = Math.PI / 60;
      const cannonLength = tankWidth * Math.sqrt(2);
      const targetWidth = Math.floor(Math.random() * 100 + 30);
      const targetHeight = Math.floor(Math.random() * 100 + 10);
      const targetX = Math.floor(Math.random() * (500 - targetWidth) + 500);
      const targetY = height - targetHeight;
      const draw = () => {
        ctx.clearRect(0, 0, width, height);
        tankCenterX = tankX + 0.5 * tankWidth;
        tankCenterY = height - 0.5 * tankHeight;
        if (tankLeftPressed && tankX > 0) {
          tankX -= tankDx;
        }
        if (tankRightPressed && tankX + tankWidth < width) {
          tankX += tankDx;
        }
        drawTank();
        drawTarget();
        drawMissile();
      };
      const drawTank = () => {
        ctx.lineWidth = 5;
        ctx.lineCap = "round";
        ctx.beginPath();
        ctx.moveTo(tankX, height - tankHeight);
        ctx.lineTo(tankX + tankWidth, height - tankHeight);
        ctx.lineTo(tankX + tankWidth, height);
        ctx.lineTo(tankX, height);
        ctx.lineTo(tankX, height - tankHeight);
        ctx.moveTo(tankCenterX, tankCenterY);
        ctx.lineTo(
          tankCenterX + cannonLength * Math.cos(cannonAngle),
          tankCenterY - cannonLength * Math.sin(cannonAngle)
        );
        ctx.stroke();
        ctx.closePath();
      };
      const drawTarget = () => {
        ctx.fillRect(targetX, targetY, targetWidth, targetHeight);
        ctx.fillStyle = "red";
      };
      const drawMissile = () => {};
      draw();
      const keydownHandler = event => {
        if (event.keyCode === 37) {
          tankLeftPressed = true;
        } else if (event.keyCode === 39) {
          tankRightPressed = true;
        } else if (event.keyCode === 38 && cannonAngle <= Math.PI) {
          cannonAngle += cannonAngleDIF;
        } else if (event.keyCode === 40 && cannonAngle >= 0) {
          cannonAngle -= cannonAngleDIF;
        }
      };
      const keyupHandler = event => {
        if (event.keyCode === 37) {
          tankLeftPressed = false;
        } else if (event.keyCode === 39) {
          tankRightPressed = false;
        }
      };
      const start = setInterval(draw, 10);
      document.addEventListener("keydown", keydownHandler, false);
      document.addEventListener("keyup", keyupHandler, false);
    </script>
  </body>
</html>
```

기능(0)

```
<script>
  const canvas = document.getElementById("fortress");
  const ctx = canvas.getContext("2d");
  const width = canvas.width;
  const height = canvas.height;
  const tankWidth = 50;
  const tankHeight = 50;
  let tankX = 0;
  const tankDx = 3;
  let tankLeftPressed = false;
  let tankRightPressed = false;
  let tankCenterX;
  let tankCenterY;
  let cannonAngle = Math.PI / 4;
  const cannonAngleDIF = Math.PI / 60;
  const cannonLength = tankWidth * Math.sqrt(2);
  const targetWidth = Math.floor(Math.random() * 100 + 30);
  const targetHeight = Math.floor(Math.random() * 100 + 10);
  const targetX = Math.floor(Math.random() * (500 - targetWidth) + 500);
  const targetY = height - targetHeight;
```

```
const draw = () => {
  ctx.clearRect(0, 0, width, height);
  tankCenterX = tankX + 0.5 * tankWidth;
  tankCenterY = height - 0.5 * tankHeight;
  if (tankLeftPressed && tankX > 0) {
    tankX -= tankDx;
  }
  if (tankRightPressed && tankX + tankWidth < width) {
    tankX += tankDx;
  }
  drawTank();
  drawTarget();
  drawMissile();
};
```

```
let missileRadius = 5;
let missileX;
let missileY;
let isCharging = false;
let isFired = false;
let isHitted = false;
let gauge = Math.PI;
const gaugeDIF = Math.PI / 60;
const gaugeBarRadius = 30;
let missilePower;
let missileDx;
let missileDy;
const GRAVITY_ACCELERATION = 0.098;
```

기능 추가를 위해 필요한 변수를 선언

기능(1-1) 파워 게이지

```
const keydownHandler = event => {
    if (event.keyCode === 37) {
        tankLeftPressed = true;
    } else if (event.keyCode === 39) {
        tankRightPressed = true;
    } else if (event.keyCode === 38 && cannonAngle <= Math.PI / 2) {
        cannonAngle += cannonAngleDIF;
    } else if (event.keyCode === 40 && cannonAngle >= 0) {
        cannonAngle -= cannonAngleDIF;
    } else if (event.keyCode === 32 && !isFired) {
        isCharging = true;
    }
    };
    const keyupHandler = event => {
    if (event.keyCode === 37) {
        tankLeftPressed = false;
    } else if (event.keyCode === 39) {
        tankRightPressed = false;
    } else if (event.keyCode === 32 && !isFired) {
        isCharging = false;
        isFired = true;
    }
    };
```

- keydownHandler와 keyupHandler 아래쪽에 추가
- 스페이스바(keyCode로 32)가 키다운상태이면 게이지 충전을 시작하고(isCharging = true) 상태가 헤제되면 게이지 충전이 끝나면서(isCharging = false) 미사일이 발사(isFired = true)

기능(1-2)-파워게이지

```
const draw = () => {
    ctx.clearRect(0, 0, width, height);
    tankCenterX = tankX + 0.5 * tankWidth;
    tankCenterY = height - 0.5 * tankHeight;
    if (tankLeftPressed && tankX > 0) {
        tankX -= tankDx;
    }
    if (tankRightPressed && tankX + tankWidth < width) {
        tankX += tankDx;
    }
    if (isCharging && !isFired) {
        if (gauge < Math.PI * 2) {
        gauge += gaugeDIF;
        }
    }
    drawTank();
    drawTarget();
    drawMissile();
};
```

- draw() 함수 내부에서 isCharging에 따라서 게이지가 충전
- isCharing이 true이고 아직 발사되지 않았을 때 gauge는 충전
- gauge는 초기값인 Math.PI 에서 시작해서 Math.PI * 2 까지 충전

기능(1-3) 파워게이지

```
const draw = () => {
    const draw = () => {
        ctx.clearRect(0, 0, width, height);
        tankCenterX = tankX + 0.5 * tankWidth;
        tankCenterY = height - 0.5 * tankHeight;
        if (tankLeftPressed && tankX > 0) {
            tankX -= tankDx;
        }
        if (tankRightPressed && tankX + tankWidth < width) {
            tankX += tankDx;
        }
        if (isCharging && !isFired) {
            if (gauge < Math.PI * 2) {
            gauge += gaugeDIF;
            }
            drawGausing();
        }
        drawTank();
        drawTarget();
        drawMissile();
        };
        const drawGausing = () => {
        ctx.beginPath();
        ctx.arc(
            tankCenterX,
            tankCenterY - cannonLength,
            gaugeBarRadius,
            Math.PI,
            gauge,
            false
        );
        ctx.stroke();
        };
};
```

- 충전된 게이지를 그리는 drawGauge() 함수를 작성하고 충전될 때마다 <canvas>에 그려지도록 조건문 안에서 호출

기능(1-4) 파워 게이지

```
const keyupHandler = event => {
if (event.keyCode === 37) {
    tankLeftPressed = false;
} else if (event.keyCode === 39) {
    tankRightPressed = false;
} else if (event.keyCode === 32 && !isFired) {
    isCharging = false;
    isFired = true;
    gauge = Math.PI;
}
};
```

- 키다운 헤제시 게이지 초기화를 위해 gauae=Math.PI문장 삽입

기능(2-1) 미사일 발사

```
const draw = () => {
    ctx.clearRect(0, 0, width, height);
    tankCenterX = tankX + 0.5 * tankWidth;
    tankCenterY = height - 0.5 * tankHeight;
    if (tankLeftPressed && tankX > 0) {
        tankX -= tankDx;
    }
    if (tankRightPressed && tankX + tankWidth < width) {
        tankX += tankDx;
    }
    if (isCharging && !isFired) {
        if (gauge < Math.PI * 2) {
        gauge += gaugeDIF;
        }
        drawGausing();
    }
    if (!isFired) {
        missileX = tankCenterX + cannonLength * Math.cos(cannonAngle);
        missileY = tankCenterY - cannonLength * Math.sin(cannonAngle);
    }
    drawTank();
    drawTarget();
    drawMissile();
    };

    const drawMissile = () => {
    ctx.beginPath();
    ctx.arc(missileX, missileY, missileRadius, 0, Math.PI * 2);
    ctx.fillStyle = "blue";
    ctx.fill();
    ctx.closePath();
};
```

- 캐논의 끝부분에 미사일이 보여지도록 미사일을 그리는 함수 선언
- if 조건으로 미사일이 발사중이 아닐 때 캐논의 끝부분에 위치

기능(2-2) 미사일 발사

```
if (!isFired) {
missileX = tankCenterX + cannonLength * Math.cos(cannonAngle);
missileY = tankCenterY - cannonLength * Math.sin(cannonAngle);
} else {
missileDy -= GRAVITY_ACCELERATION;
missileX = missileX + missileDx;
missileY = missileY - missileDy;
}
```

- 발사되었을 때의 움직이는 미사일을 그리기 위해 if 조건문에 아래 코드를 추가
- 더 알아보기:미사일이 날아가는 동안에는 x방향으로는 missileDx라는 동일한 힘을(등속도) 받고, y방향으로는 missileDy(중력가속도)의 영향으로 아래쪽으로 점점 큰 힘(등가속도)을 받도록 설계

기능(2-3) 미사일 발사

```
const keyupHandler = event => {
    if (event.keyCode === 37) {
        tankLeftPressed = false;
    } else if (event.keyCode === 39) {
        tankRightPressed = false;
    } else if (event.keyCode === 32 && !isFired) {
        isCharging = false;
        isFired = true;
        missilePower = gauge * 1.6;
        missileDx = missilePower * Math.cos(cannonAngle);
        missileDy = missilePower * Math.sin(cannonAngle);
        gauge = Math.PI;
    }
};
```

- missileDx와 missileDy가 각도와 파워게이지에 따라서 정해지도록 keyupHandler에 아래 코드를 추가
- 키다운 상태인 스페이스바가 상태 헤제 될때 isCharging이 false로 되면서 파워게이지의 충전이 끝나고 isFired가 true로 되면서 미사일의 발사가 감지됩니다. 충전된 gauge에 의해서 미사일의 파워와, 그에 따른 x방향 속도, y방향 속도가 결정됩니다. gauge는 다시 처음값으로 초기화

기능(3-1) 판정

```
const draw = () => {

    // ... 생략

    checkMissile();
    drawTank();
    drawTarget();
    drawMissile();
};
const checkMissile = () => {};
```

- checkMissile() 함수를 작성하고 draw() 함수 내부에서 호출

기능(3-2) 판정-명중하지 않았을때

```
const checkMissile = () => {
    // canvas 왼쪽, 오른쪽, 아래 벽에 닿으면
    if (missileX <= 0 || missileX >= width || missileY >= height) {
        isFired = false;
    }
};
```

- 미사일이 명중되지 않은 경우는 미사일이 왼쪽, 오른쪽, 아래쪽 canvas 테두리에 닿았을 때가 되도록 구현
- 명중되지 않았을 경우 다시 발사 할수 있도록 초기화 해준다

기능(3-3) 판정-명중했을때

```
const checkMissile = () => {
    // canvas 왼쪽, 오른쪽, 아래 벽에 닿으면
    if (missileX <= 0 || missileX >= width || missileY >= height) {
        isFired = false;
    }
    // target 명중
    if (
        missileX >= targetX &&
        missileX <= targetX + targetWidth &&
        missileY >= targetY
    ) {
        isHitted = true;
        clearInterval(start);
        if (confirm("명중입니다. 다시 하시겠습니까?")) {
        location.reload();
        }
    }
};
```

- 미사일이 명중된 경우는 미사일의 x좌표와 y좌표가 목표물의 테두리 안에 닿았을 때가 되도록 구현
- 미사일이 명중되면 isHitted가 true가 되면서 clearInterval()로 인해 draw() 함수의 실행이 중지

기능(3-4) 판정

```
const draw = () => {

    // ...생략

    checkMissile();
    drawTank();
    if (!isHitted) {
        drawTarget();
        drawMissile();
    }
};
```

- draw() 함수 내부에서 isHitted에 따라 미사일과 목표물이 그려지도록 수정
- 미사일이 목표물에 명중되어 isHitted가 true가 되면 목표물과 미사일이 더이상 보이지 않게된다

수정 후 코드

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>fortress</title>
    <style>
      body {
        margin: 0;
        height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
      }
      #fortress {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <canvas id="fortress" width="1000px" height="700px"></canvas>
    <script>
      const canvas = document.getElementById("fortress");
      const ctx = canvas.getContext("2d");
      const width = canvas.width;
      const height = canvas.height;
      const tankWidth = 50;
      const tankHeight = 50;
      let tankX = 0;
      const tankDx = 3;
      let tankLeftPressed = false;
      let tankRightPressed = false;
      let tankCenterX;
      let tankCenterY;
```

```
let cannonAngle = Math.PI / 4;
const cannonAngleDIF = Math.PI / 60;
const cannonLength = tankWidth * Math.sqrt(2);
const targetWidth = Math.floor(Math.random() * 150 + 30);
const targetHeight = Math.floor(Math.random() * 100 + 10);
const targetX = Math.floor(Math.random() * (500 - targetWidth) + 500);
const targetY = height - targetHeight;
let missileRadius = 5;
let missileX;
let missileY;
let isCharging = false;
let isFired = false;
let isHitted = false;
let gauge = Math.PI;
const gaugeDIF = Math.PI / 60;
const gaugeBarRadius = 30;
let missilePower;
let missileDx;
let missileDy;
const GRAVITY_ACCELERATION = 0.098;
const draw = () => {
  ctx.clearRect(0, 0, width, height);
  tankCenterX = tankX + 0.5 * tankWidth;
  tankCenterY = height - 0.5 * tankHeight;
  if (tankLeftPressed && tankX > 0) {
    tankX -= tankDx;
  }
  if (tankRightPressed && tankX + tankWidth < width) {
    tankX += tankDx;
  }
  if (isCharging && !isFired) {
    if (gauge < Math.PI * 2) {
      gauge += gaugeDIF;
    }
    drawGausing();
```

```
    }
    if (!isFired) {
      missileX = tankCenterX + cannonLength * Math.cos(cannonAngle);
      missileY = tankCenterY - cannonLength * Math.sin(cannonAngle);
    } else {
      missileDy -= GRAVITY_ACCELERATION;
      missileX = missileX + missileDx;
      missileY = missileY - missileDy;
    }
    checkMissile();
    drawTank();
    if (!isHitted) {
      drawTarget();
      drawMissile();
    }
  };
  const checkMissile = () => {
    // canvas 왼쪽, 오른쪽, 아래 벽에 닿으면
    if (missileX <= 0 || missileX >= width || missileY >= height) {
      isFired = false;
    }
    // target 명중
    if (
      missileX >= targetX &&
      missileX <= targetX + targetWidth &&
      missileY >= targetY
    ) {
      isHitted = true;
      clearInterval(start);
      if (confirm("명중입니다. 다시 하시겠습니까?")) {
        location.reload();
      }
    }
  };
  const drawMissile = () => {
```

```
      ctx.beginPath();
      ctx.arc(missileX, missileY, missileRadius, 0, Math.PI * 2);
      ctx.fillStyle = "blue";
      ctx.fill();
      ctx.closePath();
    };
    const drawGausing = () => {
      ctx.beginPath();
      ctx.arc(
        tankCenterX,
        tankCenterY - cannonLength,
        gaugeBarRadius,
        Math.PI,
        gauge,
        false
      );
      ctx.stroke();
    };
    const drawTank = () => {
      ctx.lineWidth = 5;
      ctx.lineCap = "round";
      ctx.beginPath();
      ctx.moveTo(tankX, height - tankHeight);
      ctx.lineTo(tankX + tankWidth, height - tankHeight);
      ctx.lineTo(tankX + tankWidth, height);
      ctx.lineTo(tankX, height);
      ctx.lineTo(tankX, height - tankHeight);
      ctx.moveTo(tankCenterX, tankCenterY);
      ctx.lineTo(
        tankCenterX + cannonLength * Math.cos(cannonAngle),
        tankCenterY - cannonLength * Math.sin(cannonAngle)
      );
      ctx.stroke();
      ctx.closePath();
    };
```

```
    const drawTarget = () => {
      ctx.fillRect(targetX, targetY, targetWidth, targetHeight);
      ctx.fillStyle = "red";
    };
    draw();
    const keydownHandler = event => {
      if (event.keyCode === 37) {
        tankLeftPressed = true;
      } else if (event.keyCode === 39) {
        tankRightPressed = true;
      } else if (event.keyCode === 38 && cannonAngle <= Math.PI / 2) {
        cannonAngle += cannonAngleDIF;
      } else if (event.keyCode === 40 && cannonAngle >= 0) {
        cannonAngle -= cannonAngleDIF;
      } else if (event.keyCode === 32 && !isFired) {
        isCharging = true;
      }
    };
    const keyupHandler = event => {
      if (event.keyCode === 37) {
        tankLeftPressed = false;
      } else if (event.keyCode === 39) {
        tankRightPressed = false;
      } else if (event.keyCode === 32 && !isFired) {
        isCharging = false;
        isFired = true;
        missilePower = gauge * 1.6;
        missileDx = missilePower * Math.cos(cannonAngle);
        missileDy = missilePower * Math.sin(cannonAngle);
        gauge = Math.PI;
      }
    };
    const start = setInterval(draw, 10);
    document.addEventListener("keydown", keydownHandler, false);
    document.addEventListener("keyup", keyupHandler, false);
  </script>
</body>
</html>
```

업그레이드(1)-각도 확장

```
} else if (event.keyCode === 38 && cannonAngle <= Math.PI / 2) {
  cannonAngle += cannonAngleDIF;
```

```
} else if (event.keyCode === 38 && cannonAngle <= Math.PI ) {
  cannonAngle += cannonAngleDIF;
```

- Pi/2를 Pi로 변경하여 각도 확장

업그레이드(2)-키다운 후에 출력

- 키다운 횟수를 지칭하는 변수 선언 후 조건문에 해당 변수가 1 이상 일때 메시지를 출력하도록 구현하고 벽에 맞았을 경우 그 횟수를 초기화
- 이때 키다운 상태에서 변수 값을 올릴경우 키다운 상태일동안 계속 올라가게 된다. 이경우 타겟 근접 시 스페이스바를 키다운 중인 상태일때 메시지가 출력되므로 반드시 키다운을 헤제한 상태로 넘어갔을 때 변수 값을 올려주도록 한다

```
let missilePower;
let missileDx;
let missileDy;
const GRAVITY_ACCELERATION = 0.098;
let presscount = 0;
const draw = () => {
```

```
if (missileX <= 0 || missileX >= width || missileY >= height) {
  isFired = false;
  presscount = 0;
}
```

```
const keyupHandler = event => {
  if (event.keyCode === 37) {
    tankLeftPressed = false;
  } else if (event.keyCode === 39) {
    tankRightPressed = false;
  } else if (event.keyCode === 32 && !isFired) {
    isCharging = false;
    isFired = true;
    presscount+=1;
    missilePower = gauge * 1.6;
    missileDx = missilePower * Math.cos(cannonAngle);
    missileDy = missilePower * Math.sin(cannonAngle);
    gauge = Math.PI;
  }
};
```

업그레이드 최종 코드

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>fortress</title>
    <style>
      body {
        margin: 0;
        height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
      }
      #fortress {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <canvas id="fortress" width="1000px" height="700px"></canvas>
    <script>
      const canvas = document.getElementById("fortress");
      const ctx = canvas.getContext("2d");
      const width = canvas.width;
      const height = canvas.height;
      const tankWidth = 50;
      const tankHeight = 50;
      let tankX = 0;
      const tankDx = 3;
      let tankLeftPressed = false;
      let tankRightPressed = false;
      let tankCenterX;
      let tankCenterY;
      let cannonAngle = Math.PI / 4;
      const cannonAngleDIF = Math.PI / 60;
      const cannonLength = tankWidth * Math.sqrt(2);
```

```
const targetWidth = Math.floor(Math.random() * 150 + 30);
const targetHeight = Math.floor(Math.random() * 100 + 10);
const targetX = Math.floor(Math.random() * (500 - targetWidth) + 500);
const targetY = height - targetHeight;
let missileRadius = 5;
let missileX;
let missileY;
let isCharging = false;
let isFired = false;
let isHitted = false;
let gauge = Math.PI;
const gaugeDIF = Math.PI / 60;
const gaugeBarRadius = 30;
let missilePower;
let missileDx;
let missileDy;
const GRAVITY_ACCELERATION = 0.098;
let presscount = 0;
const draw = () => {
  ctx.clearRect(0, 0, width, height);
  tankCenterX = tankX + 0.5 * tankWidth;
  tankCenterY = height - 0.5 * tankHeight;
  if (tankLeftPressed && tankX > 0) {
    tankX -= tankDx;
  }
  if (tankRightPressed && tankX + tankWidth < width) {
    tankX += tankDx;
  }
  if (isCharging && !isFired) {
    if (gauge < Math.PI * 2) {
      gauge += gaugeDIF;
    }
    drawGausing();
  }
  if (!isFired) {
    missileX = tankCenterX + cannonLength * Math.cos(cannonAngle);
    missileY = tankCenterY - cannonLength * Math.sin(cannonAngle);
  } else {
```

```
    missileDy -= GRAVITY_ACCELERATION;
    missileX = missileX + missileDx;
    missileY = missileY - missileDy;
  }
  checkMissile();
  drawTank();
  if (!isHitted) {
    drawTarget();
    drawMissile();
  }
};
const checkMissile = () => {
  // canvas 왼쪽, 오른쪽, 아래 벽에 닿으면
  if (missileX <= 0 || missileX >= width || missileY >= height) {
    isFired = false;
    presscount = 0;
  }
  // target 명중
  if (
    missileX >= targetX &&
    missileX <= targetX + targetWidth &&
    missileY >= targetY &&
    presscount >= 1
  ) {
    isHitted = true;
    clearInterval(start);
    if (confirm("명중입니다. 다시 하시겠습니까?")) {
      location.reload();
    }
  }
};
const drawMissile = () => {
  ctx.beginPath();
  ctx.arc(missileX, missileY, missileRadius, 0, Math.PI * 2);
  ctx.fillStyle = "blue";
  ctx.fill();
  ctx.closePath();
};
```

```
const drawGausing = () => {
  ctx.beginPath();
  ctx.arc(
    tankCenterX,
    tankCenterY - cannonLength,
    gaugeBarRadius,
    Math.PI,
    gauge,
    false
  );
  ctx.stroke();
};
const drawTank = () => {
  ctx.lineWidth = 5;
  ctx.lineCap = "round";
  ctx.beginPath();
  ctx.moveTo(tankX, height - tankHeight);
  ctx.lineTo(tankX + tankWidth, height - tankHeight);
  ctx.lineTo(tankX + tankWidth, height);
  ctx.lineTo(tankX, height);
  ctx.lineTo(tankX, height - tankHeight);
  ctx.moveTo(tankCenterX, tankCenterY);
  ctx.lineTo(
    tankCenterX + cannonLength * Math.cos(cannonAngle),
    tankCenterY - cannonLength * Math.sin(cannonAngle)
  );
  ctx.stroke();
  ctx.closePath();
};
const drawTarget = () => {
  ctx.fillRect(targetX, targetY, targetWidth, targetHeight);
  ctx.fillStyle = "red";
};
draw();
const keydownHandler = event => {
  if (event.keyCode === 37) {
    tankLeftPressed = true;
  } else if (event.keyCode === 39) {
```

```
        tankRightPressed = true;
      } else if (event.keyCode === 38 && cannonAngle <= Math.PI ) {
        cannonAngle += cannonAngleDIF;
      } else if (event.keyCode === 40 && cannonAngle >= 0) {
        cannonAngle -= cannonAngleDIF;
      } else if (event.keyCode === 32 && !isFired) {
        isCharging = true;
      }
    };
    const keyupHandler = event => {
      if (event.keyCode === 37) {
        tankLeftPressed = false;
      } else if (event.keyCode === 39) {
        tankRightPressed = false;
      } else if (event.keyCode === 32 && !isFired) {
        isCharging = false;
        isFired = true;
        presscount+=1;
        missilePower = gauge * 1.6;
        missileDx = missilePower * Math.cos(cannonAngle);
        missileDy = missilePower * Math.sin(cannonAngle);
        gauge = Math.PI;
      }
    };
    const start = setInterval(draw, 10);
    document.addEventListener("keydown", keydownHandler, false);
    document.addEventListener("keyup", keyupHandler, false);
  </script>
</body>
</html>
```

이상입니다.

참조 사이트: https://codingbroker.tistory.com/76

index: https://relaxed-rolypoly-95eed8.netlify.app/

github: https://github.com/immotals/ksgame/tree/main/canvas