포트리스게임 개선

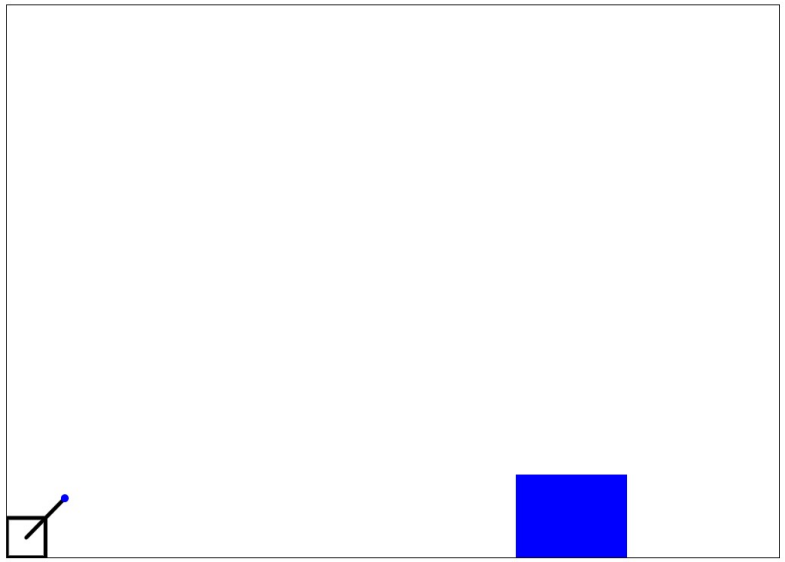
소프트웨어학과

2018675060 정성호

목차

- 게임 설명
- 원본 소스 분석
- 게임 보완 및 업그레이드할 점
- 업그레이드 소스 분석

게임 설명



- 탱크를 좌우로 조작하여 목표물을 맞추는 게임
- 목표물을 맞출 때까지 계속 시도
- 좌우조작 = 방향키
- 미사일 발사 = 스페이스바

원본 소스 분석 : <canvas>, setInterval

```
<canvas id="fortress" width="1100px" height="700px"></canvas>
```

- <canvas> 태그는 자바스크립트를 통해 다양한 그림을 그릴 수 있는 공간을 제공
- canvas의 폭과 높이 설정

```
let start = setInterval(draw, 10);
```

- setInterval로 draw()를 10ms마다 반복 호출
- setInterval를 이용하여 애니메이션 효과 연출

원본 소스 분석 : 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();
  }
  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();
  }
};
```

- <canvas>에 그림을 그리는 메인 함수
- 탱크, 미사일 등 객체는 따로 구현 후 호출
- 탱크의 좌우 움직임 구현(if문)
- 게이지 충전중(isCharging)일 때,
  게이지(gauge) 계속 충전(gaugeDIF)
- 발사 전(!isFired), 미사일은 캐논 끝부분에 위치
- 발사 후, 중력가속도(GRAVITY_ACCELERATION)
  의해 미사일 점차 하강

➢ clearRect() = 사각 영역 지우는 함수

➢ tankX = 탱크 위치, tankDx = 탱크 속도

➢ tankLeftPressed = 왼쪽 방향키가 눌렸는지 여부

➢ tankRightPressed = 오른쪽 방향키가 눌렸는지 여부

원본 소스 분석 : checkMissile()

```
const checkMissile = () => {
  if (missileX <= 0 || missileX >= width || missileY >= height) {
    isFired = false;
  }
  if (
    missileX >= targetX &&
    missileX <= targetX + targetWidth &&
    missileY >= targetY
  ) {
    isHitted = true;
    clearInterval(start);
    if (confirm("명중입니다. 다시 하시겠습니까?")) {
      location.reload();
    }
  }
};
```

- 목표물 명중 체크하는 함수
- 미사일이 canvas 왼쪽, 오른쪽, 아래 벽에 닿으면 발사 전 상태로 복귀(false)
- 미사일이 목표물에 명중 시 함수 반복 중단(clearInterval) 및 confirm창 출력 후 새로고침(location.reload)

➢ missileX = 미사일 x좌표, missileY = 미사일 y좌표

➢ targetX = 목표물 x좌표, targetY = 목표물 y좌표

➢ isFired = 공이 발사되었는지 여부

➢ isHitted = 명중했는지 여부

원본 소스 분석 : drawMissile()

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

- 미사일 객체 그리는 함수
- beginPath() ~ closePath() 사이에 코드를 넣어 도형 그리기
- 미사일은 원형(arc)
- 미사일의 색 지정(fillStyle)
- 미사일 생성(fill)

➢ missileRadius = 미사일의 반지름

arc(x,y,반지름,시작각도,끝각도)

원본 소스 분석 : drawGausing()

```
const drawGausing = () => {
  ctx.beginPath();
  ctx.arc(
    tankCenterX,
    tankCenterY - cannonLength,
    gaugeBarRadius,
    Math.PI,
    gauge,
    false
  );
  ctx.stroke();
};
```

- 게이지 그리는 함수
- 게이지는 반원(gaugeBarRadius)모양
- 시계방향으로(false)으로 그리기
- 경로에 있는 도형들을 모두 그림(stroke)

➢ gauge = 게이지

원본 소스 분석 : drawTank()

```
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();
};
```

- 탱크 그리는 함수
- 선 두께 지정(lineWidth)
- 선 끝 부분은 둥글게(lineCap)
- 탱크를 그릴 선이 시작될 좌표(moveTo),
  선이 끝나는 좌표 지정(lineTo)
- 캐논 또한 동일하게 좌표 지정
- 마지막으로 선을 그려서(stroke) 탱크를 그림

➢ cannonAngle = 캐논 각도

원본 소스 분석 : drawTarget()

```
const drawTarget = () => {
  ctx.fillRect(targetX, targetY, targetWidth, targetHeight);
  ctx.fillStyle = "red";
};
```

- 목표물 그리는 함수
- 색이 채워진 사각형을 그림(fillRect)
- 사각형 색 지정(fillStyle)
  - targetWidth = 목표물 폭
  - targetHeight = 목표물 높이

원본 소스 분석 : keydownHandler()

```
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;
  }
};
```

- 키보드가 눌렸을 때 실행되는 핸들러
- 왼쪽방향키(keyCode=37)를 눌렀다면 tankLeftPressed = true
- 위쪽방향키(keycode=38)을 누르고, 캐논각도가 범위 내라면 각도 상승
- 스페이스바(keycode=32)를 누르고, 발사하지 않았다면 게이지 충전(isCharging)

원본 소스 분석 : keyupHandler()

```
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;
  }
};
```

- 키보드에서 손을 떼었을 때 실행되는 핸들러
- 왼쪽방향키에서 손을 뗐다면
  tankLeftPressed = false
- 스페이스바를 떼고 발사하지 않은 상태라면
  게이지 충전 중단(false) 및 발사(isFired),
  미사일 속도 지정 및 미사일 위치 변화

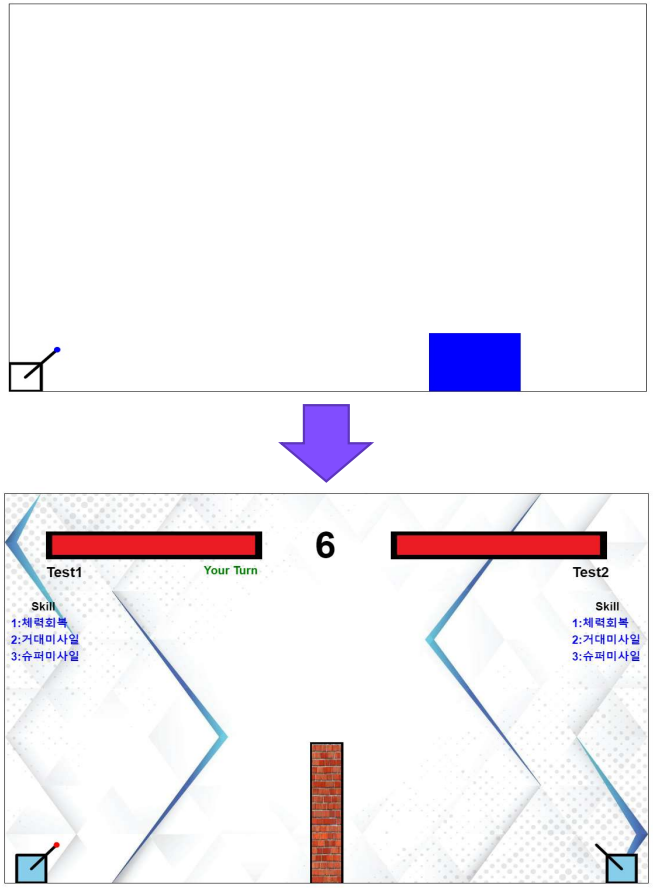
➢ missilePower = 미사일 파워

➢ missileDx = 미사일의 x방향 속도

➢ missileDy = 미사일의 y방향 속도

게임 보완 및 업그레이드할 점

- canvas, 최대 이동 거리 등 잘못된 수치 수정
- confirm창 문제, 발사 이후 움직임 등 버그 수정
- 1:1 모드(턴제) 추가
- 배경이미지 추가
- 방해물 추가
- HP 추가 및 그래픽화
- 플레이어명 추가
- 효과음(발사, 명중, 스킬) 추가
- 스킬 3종 추가(skillHandler())
- 타이머 추가



- 잘못된 수치 수정

```
<canvas id="fortress" width="1100px" height="700px"></canvas>
```

- 미사일 최대비거리 반영하여 canvas 크기 수정

drawTankA()

```
const drawTankA = () => {
  ctx.lineWidth = 5;
  ctx.lineCap = "round";
  ctx.beginPath();
  ctx.moveTo(tankA, height - tankHeight);
  ctx.lineTo(tankA + tankWidth, height - tankHeight);
  ctx.lineTo(tankA + tankWidth, height);
  ctx.lineTo(tankA, height);
  ctx.lineTo(tankA, height - tankHeight);
  ctx.moveTo(tankACenterX, tankACenterY);
  ctx.fillStyle = "skyblue";
  ctx.fillRect(tankACenterX - 25, tankACenterY - 25, tankWidth, tankHeight);
  ctx.lineTo(tankACenterX + cannonLength * Math.cos(cannonAAngle), tankACenterY - cannonLength * Math.sin(cannonAAngle));
  ctx.stroke();
  ctx.closePath();
};
```

- 탱크 내부가 투명한 문제 수정
  (fillStyle, FillRect)

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

- 미사일 색이 정상 적용되지 않았던 문제 수정

draw()

```
if (tankLeftPressed && tankA > 0) {
  tankA -= tankDa;
}
if (tankRightPressed && tankA + tankWidth < width / 2.5) {
  tankA += tankDa;
}
```

- 탱크가 끝에서 끝까지 가게 되면 게임이
  바로 끝나게 되는 문제가 생기므로
  탱크의 최대이동거리 제한

- 버그 수정

keydownHandler()

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

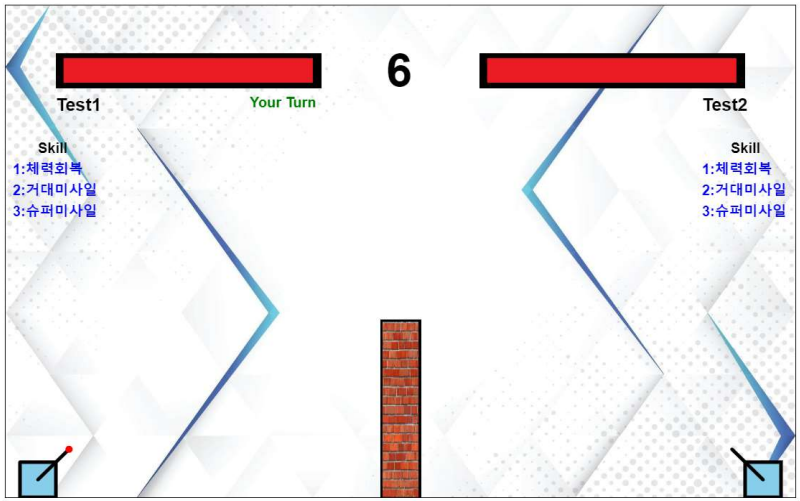
- 게이지 충전 이후에도 탱크가 움직일 수 있는 버그 존재
- 충전 이후에는 탱크가 움직이지 못하게 수정 (!isFired && !isCharging)

draw()

```
if (AHP <= 0) {
  clearInterval(start);
  if (confirm(Bname+"님 승리! 다시 하시겠습니까?")) {
    location.reload();
  }
  else window.close();
}
else if (BHP <= 0) {
  clearInterval(start);
  if (confirm(Aname+"님 승리! 다시 하시겠습니까?")) {
    location.reload();
  }
  else window.close();
}
```

- 게임 결과를 confirm창에서 출력한 후, 취소버튼 누를 시 브라우저가 멈추는 버그 존재
- 취소버튼 누를 시 브라우저가 꺼지도록 수정 (window.close())

- 1:1 모드 추가(턴제)



- 기존의 1인용에서 2인용으로 변경
- 상대 탱크를 명중시켜 먼저 HP를 0이하로 만들면 승리
- 공격권을 주고받으며 턴제로 게임 운영
- 공격권=control, 공격권 전환=swap()으로 턴제 구현
- 공격권이 있을 때만 탱크 이동, 스킬사용, 미사일 발사 가능
- control=false면 Player1의 턴, true면 Player2의 턴

```
let control = false;
```

swap()

```
const swap = () => {
  control = !control;
  isFired = false;
  isCharging = false;
  missileRadius = 5;

  count = 9;
}
```

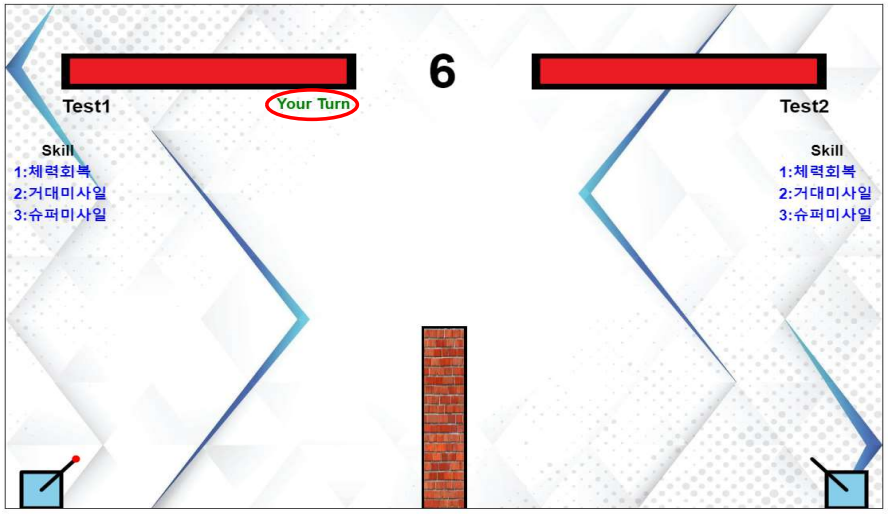
예시(공격권에 따른 탱크 이동)

```
if (control == false) {
  if (tankLeftPressed && tankA > 0) {
    tankA -= tankDa;
  }
  if (tankRightPressed && tankA + tankWidth < width / 2.5) {
    tankA += tankDa;
  }
} else if (control == true) {
  if (tankLeftPressed && tankB + tankWidth > width / 1.5 - 20) {
    tankB -= tankDb;
  }
  if (tankRightPressed && tankB + tankWidth < width) {
    tankB += tankDb;
  }
}
```

- 1:1 모드 추가(턴제)

drawUI()

```
ctx.fillStyle = 'green';
if(control == false) { ctx.fillText("Your Turn", height / 2 - 10, 145); }
else if (control == true) { ctx.fillText("Your Turn", height - 35, 145); }
```



- 체력회복 스킬 사용, 명중/오중, 제한시간 오버 시 swap()
- swap() 호출시 공격권 전환을 위해 변수 및 타이머 초기화
- control을 이용하여 누구의 턴인지 직관적으로 확인 (Your Turn)
- Your Turn은 공격권 전환할때마다 해당 Player의 위치로 이동하여 출력

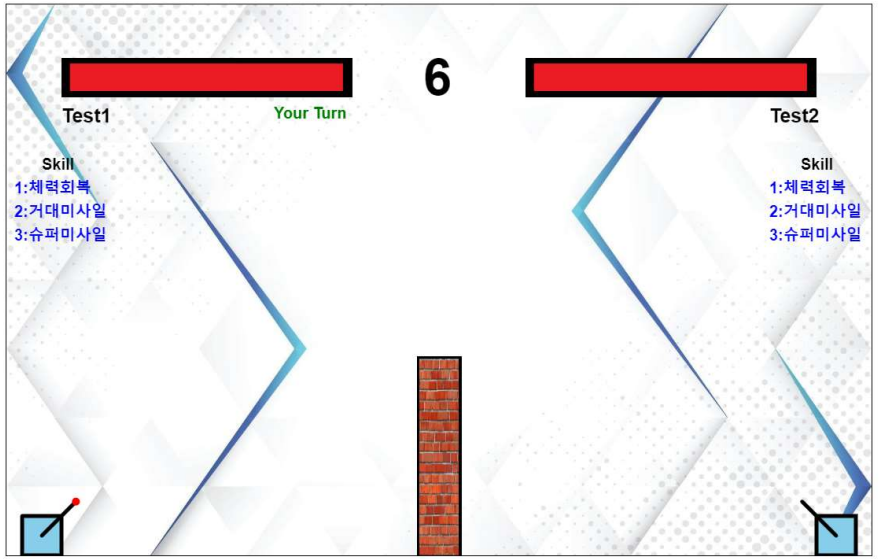
- 배경이미지 추가

```
<div style="display: none"><img id="background" src="./image/background.jpg" /></div>
```

draw()

```
ctx.drawImage(document.getElementById("background"), 0, 0, width, height);
```

- 원본에 없던 배경이미지 추가
- drawImage로 이미지 출력



- 방해물 추가

```
<div style="display: none"><img id="wall" src="./image/wall.jpg" /></div>
```

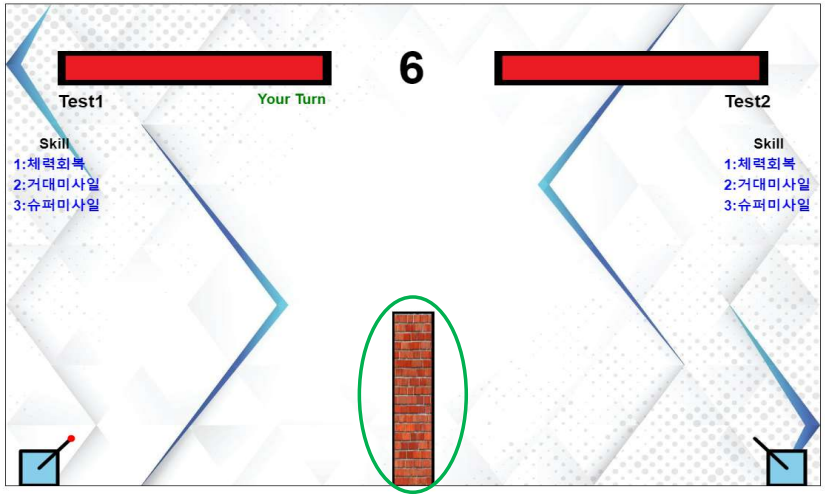
draw()

```
ctx.drawImage(document.getElementById("wall"), blockX, blockY, blockWidth, blockHeight);
```

checkMissile()

```
if (missileX >= blockX && missileX <= blockX + blockWidth && missileY >= blockY) {
    missileX = width + 10;
    missileY = height - 10;
}
```

- canvas 중앙에 방해물 설치
- 미사일은 방해물을 통과할 수 없게 설정
- 미사일이 방해물에 닿을 시 공격권 전환



- HP 추가 및 그래픽화

```
let AHP = 5;
let BHP = 5;
```

checkMissile()

```
if (control == true) {
  if (missileX >= tankACenterX - 0.5 * tankWidth && missileX <= tankACenterX + 0.5 * tankWidth && missileY >= tankACenterY - 0.5 * tankHeight) {
    if (checktSkill == 1) { AHP = AHP - 1; checktSkill = 0; }
    AHP = AHP - 1;
    boom.play();
    missileX = width + 10;
    missileY = height - 10;
  }
} else {
  if (missileX >= tankBCenterX - 0.5 * tankWidth && missileX <= tankBCenterX + 0.5 * tankWidth && missileY >= tankBCenterY - 0.5 * tankHeight) {
    if (checktSkill == 1) { BHP = BHP - 1; checktSkill = 0; }
    BHP = BHP - 1;
    boom.play();
    missileX = width + 10;
    missileY = height - 10;
  }
}
```

draw()

```
if (AHP <= 0) {
   clearInterval(start);
   if (confirm(Bname+"님 승리! 다시 하시겠습니까?")) {
     location.reload();
   }
   else window.close();
}
else if (BHP <= 0) {
   clearInterval(start);
   if (confirm(Aname+"님 승리! 다시 하시겠습니까?")) {
     location.reload();
   }
   else window.close();
}
```

- 각 Player HP 초기값은 5
- 명중 시 피격 당한 Player HP 1 감소
- 슈퍼미사일 스킬 사용 시 HP 추가 감소
- 체력회복 스킬 사용 시 HP 1 증가
- HP가 0이하면 게임 종료 및 confirm창 출력

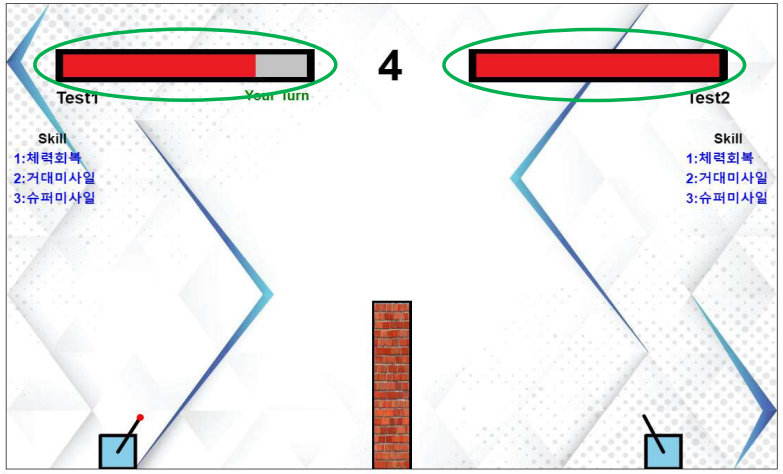
- HP 추가 및 그래픽화

```
<div style="display: none"><img id="hpbar1" src="./image/hpbar1.png"/></div>
<div style="display: none"><img id="hpbar2" src="./image/hpbar2.png"/></div>
<div style="display: none"><img id="hpbar3" src="./image/hpbar3.png"/></div>
<div style="display: none"><img id="hpbar4" src="./image/hpbar4.png"/></div>
<div style="display: none"><img id="hpbar5" src="./image/hpbar5.png"/></div>
<div style="display: none"><img id="bhpbar1" src="./image/Bhpbar1.png"/></div>
<div style="display: none"><img id="bhpbar2" src="./image/Bhpbar2.png"/></div>
<div style="display: none"><img id="bhpbar3" src="./image/Bhpbar3.png"/></div>
<div style="display: none"><img id="bhpbar4" src="./image/Bhpbar4.png"/></div>
```

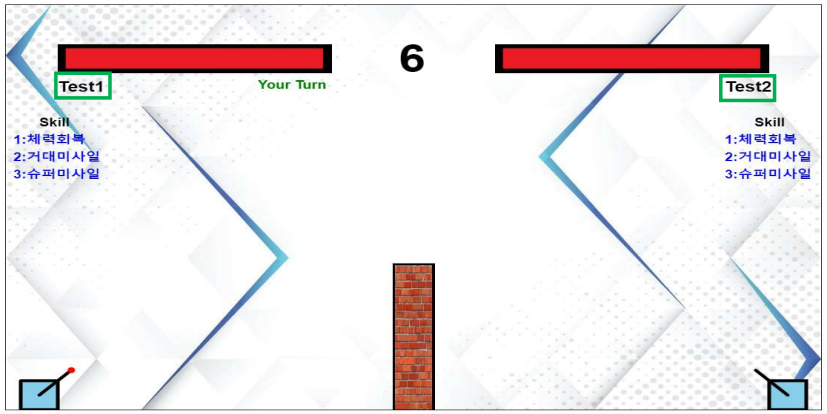
drawUI()

```
if(AHP == 4) { ctx.drawImage(document.getElementById("hpbar4"), height / 10, 68, 370, 50); }
else if(AHP == 3) { ctx.drawImage(document.getElementById("hpbar3"), height / 10, 68, 370, 50); }
else if(AHP == 2) { ctx.drawImage(document.getElementById("hpbar2"), height / 10, 68, 370, 50); }
else if(AHP == 1) { ctx.drawImage(document.getElementById("hpbar1"), height / 10, 68, 370, 50); }
if(BHP == 4) { ctx.drawImage(document.getElementById("bhpbar4"), height - 40, 68, 370, 50); }
else if(BHP == 3) { ctx.drawImage(document.getElementById("bhpbar3"), height - 40, 68, 370, 50); }
else if(BHP == 2) { ctx.drawImage(document.getElementById("bhpbar2"), height - 40, 68, 370, 50); }
else if(BHP == 1) { ctx.drawImage(document.getElementById("bhpbar1"), height - 40, 68, 370, 50); }
```

- HP 표현을 텍스트가 아닌 이미지로 표시하여 직관성 강화
- drawImage()로 각 Player의 HP에 맞게 출력



- 플레이어명 추가



- 최초 페이지 진입 시 prompt를 통해 플레이어명 입력
- Player1과 Player2간 이름이 겹치지 않게끔 설정
- 메인화면에서 각자의 플레이어명이 HP바 아래에 출력
- 경기 종료 시 승리한 Player의 플레이어명 출력(draw())

name()

```
const name = () => {
  Aname = prompt("Player1의 이름을 입력해주세요.");
  while(Aname == null || Aname == "") {Aname = prompt("Player1의 이름을 다시 입력해주세요.")};
  Bname = prompt("Player2의 이름을 입력해주세요.(Player1과 이름 중복 불가)");
  while(Bname == null || Bname == "" || Bname == Aname) {Bname = prompt("Player2의 이름을 다시 입력해주세요.(Player1과 이름 중복 불가)")};
}
```

drawUI()

```
ctx.font = "bold 25px Arial, sans-serif";
ctx.fillText(Aname, height / 10, 150);
ctx.fillText(Bname, height + 270, 150);
```

- 효과음(발사, 명중, 스킬) 추가

```
bang.src = "./sound/bang.mp3";
bang.volume =  0.5;
bang.playbackRate = 1.5;
let boom = new Audio();
boom.src = "./sound/boom.mp3";
boom.volume = 0.5;
boom.playbackRate = 2.0;
let useSkill = new Audio();
useSkill.src = "./sound/skill.mp3";
```

- 미사일 발사 시, 미사일 명중 시, 스킬 사용 시 효과음 발생
  - ➢ volume = 소리 세기
  - ➢ playbackRate = 배속

```
if (missileX >= tankACenterX - 0.5 * tankWidth && missileX <= tankACenterX + 0.5 * tankWidth && missileY >= tankACenterY - 0.5 * tankHeight) {
  if (checktSkill == 1) { AHP = AHP - 1; checktSkill = 0; }
  AHP = AHP - 1;
  boom.play();
  missileX = width + 10;
  missileY = height - 10;
}
```

미사일 명중
(boom)

```
if (control == false && fSkillA == true && !isCharging && !isFired && AHP != 5) {
  useSkill.play();
  AHP += 1;
  fSkillA = false;
  checktSkill = 0;
  swap();
}
```

스킬 사용(useSkill)

```
if (!isFired) {
  if (control == false) {
    ...
    ...
  }
} else {
  bang.play();
  missileDy -= GRAVITY_ACCELERATION;
  if (control == false) {
    ...
    ...
  }
}
```

미사일 발사
(bang)

- 스킬 추가(체력 회복)

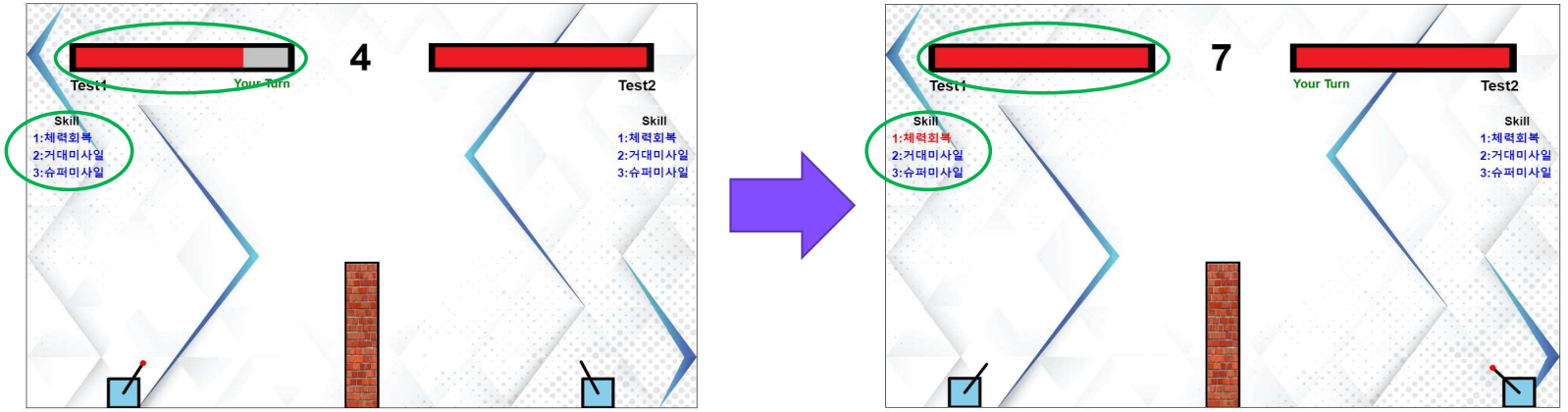
skillHandler()

```
if (event.keyCode == 49) {
  if (control == false && fSkillA == true && !isCharging && !isFired && AHP != 5) {
    useSkill.play();
    AHP += 1;
    fSkillA = false;
    checktSkill = 0;
    swap();
  }
  else if (control == true && fSkillB == true && !isCharging && !isFired && BHP != 5) {
    useSkill.play();
    BHP += 1;
    fSkillB = false;
    checktSkill = 0;
    swap();
  }
}
```

drawUI()

```
if (fSkillA == true) { ctx.fillStyle = 'blue'; }
else { ctx.fillStyle = 'red'; }
ctx.fillText("1:체력회복", 10, 240);
```

- 키보드 1번(keyCode=49) 누를 시 HP 1 증가
- 게이지 충전 및 발사 전 사용 가능
- HP가 최대(HP==5)면 스킬 사용 불가
- 스킬 사용 후 즉시 공격권 전환(swap())
- 스킬 재사용 불가능(fSkillA=false)
- 스킬 사용 여부 UI 추가(drawUI())

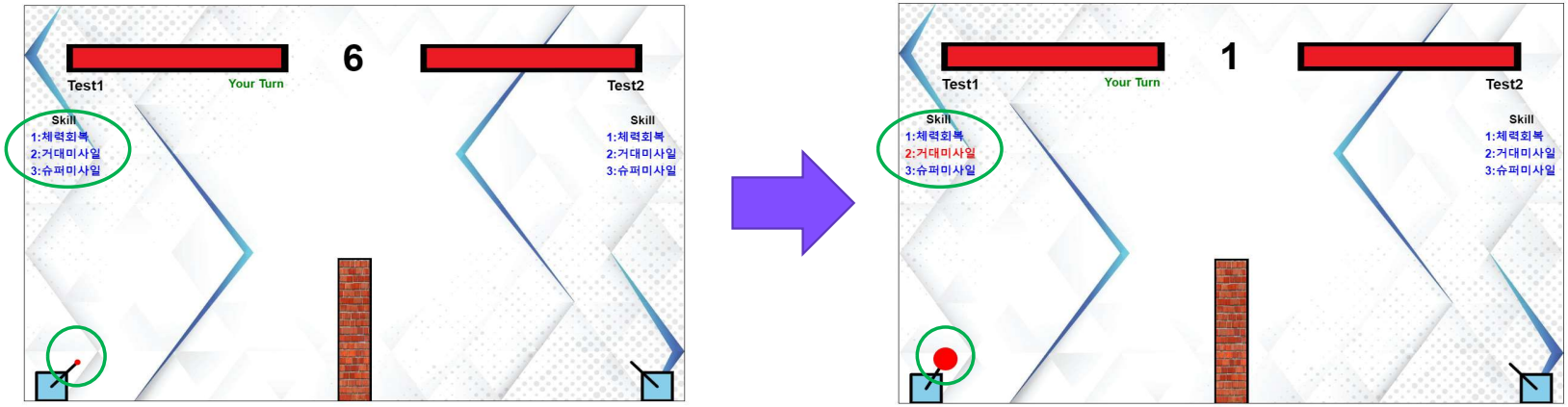


- 스킬 추가(거대미사일)

skillHandler()

```
else if (event.keyCode == 50) {
  if (control == false && sSkillA == true && !isCharging && !isFired) {
    useSkill.play();
    missileRadius = 20;
    sSkillA = false;
  }
  else if (control == true && sSkillB == true && !isCharging && !isFired) {
    useSkill.play();
    missileRadius = 20;
    sSkillB = false;
  }
}
```

- 키보드 2번(keyCode=50) 누를 시 거대미사일
- 게이지 충전 및 발사 전 사용 가능
- 미사일 반지름(missileRadius) 4배 증가
- 스킬 재사용 불가능(sSkillA=false)
- 공격권 전환 시 missileRadius값 초기화(swap())



- 스킬 추가(슈퍼미사일)

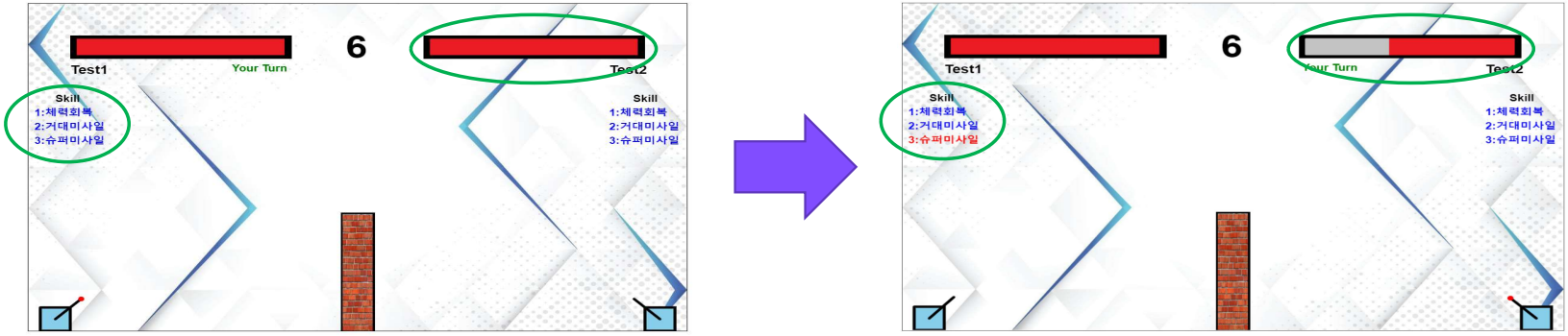
skillHandler()

```
  else if (event.keyCode == 51) {
    if (control == false && tSkillA == true && !isCharging && !isFired) {
      useSkill.play();
      checktSkill = 1;
      tSkillA = false;
    }
    else if (control == true && tSkillB == true && !isCharging && !isFired) {
      useSkill.play();
      checktSkill = 1;
      tSkillB = false;
    }
  }
```

checkMissile()

```
if (missileX >= tankACenterX - 0.5 * tankWidth && missileX <= tankACenterX + 0.5 * tankWidth && missileY >= tankACenterY - 0.5 * tankHeight) {
  if (checktSkill == 1) { AHP = AHP - 1; checktSkill = 0; }
  AHP = AHP - 1;
  boom.play();
  missileX = width + 10;
  missileY = height - 10;
}
```

- 키보드 3번(keyCode=51) 누를 시 슈퍼미사일
- 게이지 충전 및 발사 전 사용 가능
- 스킬 사용 시 checktSkill=1
- checktSkill=1일 때 명중하면 추가데미지
- 스킬 재사용 불가능(tSkillA=false)
- 공격권 전환 시 checkSkill=0



- 타이머 추가

counter()

```
const counter = () => {
  if(!isCharging && !isFired) {
    count = count - 1;
    if (count <= 0) {
      swap();
      checktSkill = 0;
    }
  }
}
```

swap()

```
const swap = () => {
  control = !control;
  isFired = false;
  isCharging = false;
  missileRadius = 5;

  count = 9;
}
```

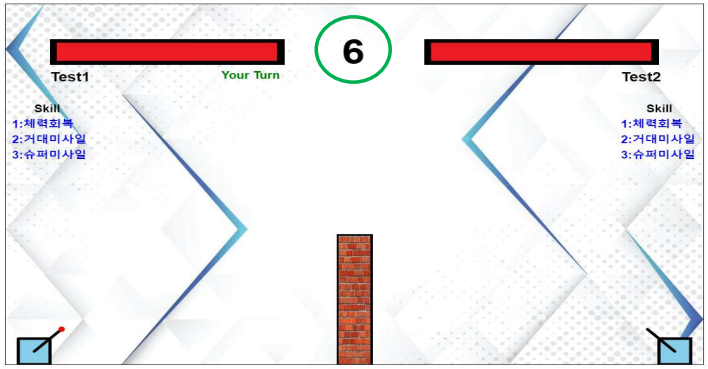
setInterval()

```
setInterval(counter, 1000);
```

drawUI()

```
ctx.font = "bold 65px Arial, sans-serif";
ctx.fillStyle = 'black';
ctx.fillText(count, blockX + 5, 115);
```

- 제한시간 내에 발사하지 않으면 공격권 강제 전환
- 게이지 충전중이지 않고(!isCharging), 발사하지 않았을때만(!isFired) 타이머 동작
- setInterval()로 1000ms(1초)마다 count 감소
- count<=0이면 공격권 강제 전환(swap()) 및 변수들 원래 상태로 초기화
- 명중/오중 시에도 swap() 호출
- 다시 count=9부터 시작
- 화면 중앙 상단에 타이머 표시



출처

- 원본 : https://codingbroker.tistory.com/76
- 사운드 : https://www.mewpot.com/
- 배경화면 및 이미지 : https://www.vecteezy.com/vector-art/3161225-abstract-white-and-blue-background/
- keyCode 정리 : http://superkts.pe.kr/upload/helper/file1/keyCode.html
- addEventListener : https://kyounghwan01.github.io/blog/JS/JSbasic/addEventListener/
- 타이머 : https://www.codemahal.com/javascript-and-html5-canvas-game-tutorial-code/
- canvas 기초 : https://www.habonyphp.com/2021/01/blog-post_47.html
- canvas 기초 : https://malonmiming.tistory.com/87
- canvas 기초 : https://curryyou.tistory.com/328

깃허브 주소

- https://github.com/seong2517/Game_Programming/tree/main/html_game(1116)

감사합니다.