웹프로그래밍 2조 보고서

조명 : 웹프로그래밍 2조

팀원 : 202411299 배정환, 202211393 추현규, 202211263 김도현 202211262 김경수

제출일 : 2025년 6월 8일

게임 시나리오

인트로

💡 옛날 옛적부터 지옥의 검을 지키는 가업을 수행하는 형제가 있었다.

지옥의 검은 막강한 힘을 자랑하지만 그와 동시에 사람을 난폭하게 만드는 아주 위험한 검이다.

형과 동생은 현재는 봉인된 검의 힘에 대해서 느껴본 바는 없었다만 서로 생각하는 부분은 달랐다.

형은 오로지 검을 개방해서는 안된다는 생각을 가지고 있었고 반면 동생은 검을 사용하고자 갈망하고 있었다.

검에 대한 호기심과 강력한 힘에 대한 갈망에 형이 자리를 비운 사이 동생이 검 강탈을 시도하는데...

검을 훔친 동생은 막강한 힘을 누리는 듯 싶더니...

결국 힘을 주체하지 못하고 밖으로 뛰쳐나가버린다.

뒤늦게 이를 알아차린 형은 동생을 지키기 위해 그리고 가업을 마저 수행하기 위해 검을 찾으러 나선다.

"동생 성격이라면 분명 검의 성능을 테스트하러 우리 마을을 수호하는 기사들이 모여있는 성으로 갔을거야!"

에필로그

💡 동생은 힘을 잃고 그만 주저앉고 말았다.

이내 원래 모습으로 되돌아 왔으며 형과 눈물의 재회를 하였다.

"못된 짓을 한 나를 지켜줘서 고마워 형...”

"내가 해야 했을 일인데 뭐..ㅎㅎ 너가 무사해서 다행이지.”

결국 검은 제자리에 돌아왔으며 더욱 깊은 공간에 봉인될 수 있었다.

게임 동작 방법

1. 최초 시작 시
   - 맨 처음에 press anywhere to start라는 창이 뜨고 여기서 아무 곳이나 클릭하면 인트로 시나리오가 전개되는데 이때 space 키를 누르면 빠른 전개가 가능하고 skip 버튼을 누르면 메인 화면으로 넘어갈 수 있다. 이는 에필로그 시나리오 전개 이후에도 동일하게 적용된다.
2. 메인 화면
   - Play 버튼

     게임을 플레이 하기 위해서 눌러야 하는 버튼이다.
     - level 선택

       level 1부터 level 3까지 있으며 레벨이 높아질수록 공의 속도기 증가하며 등장하는 보스도 레벨에 따라 달라진다.
   - Setting 버튼
     - 사운드 설정

        - 게임에 사용하는 효과음, 배경 브금들의 사운드를 조절할 수 있다.
    - 벽돌, 공 색깔 설정
        - 벽돌과 공 색깔은 따로따로 설정이 가능하다.
    - Apply 버튼
        - 적용 버튼으로 해당 설정을 적용시키는 버튼이다. 이 버튼을 누르지 않고 Back 버튼을 누르게 된다면 변경한 설정이 적용되지 않는다.
    - Reset 버튼
        - 게임 초기 실행 시 설정해둔 색상, 볼륨으로 변경된다.
    - Back 버튼
        - 메인 화면으로 돌아가는 버튼이다.
- Quit 버튼

  게임을 종료하는 버튼이며 실행할 경우 창이 닫히며 종료된다.

3. 인게임
    - 기본적인 조작

      마우스를 통해 좌우로 이동이 가능하며 공이 하단부에 떨어지지 않게 패들(주인공 입장에서는 방패)을 이용하여 튕긴다. 튕긴 공이 벽돌에 맞으면 벽돌이 사라짐과 동시에 점수가 추가되고 보스에 맞으면 보스 체력이 감소한다.
    - 아이템

      아이템이 각각 1개 이상일 때, A키를 눌러 공격, S키를 눌러 점수 버프, D키를 눌러 공 오브젝트 튕김 무시를 사용할 수 있다.
    - 일시정지

      스페이스 키를 눌러 일시정지가 가능하며 일시정지 시에는 게임 내에서 흘러가는 시간도 멈춘다. 다시 스페이스 키를 눌러 일시정지 해제가 가능하다.
    - 메인 메뉴 버튼

      게임 도중 그만하기를 원하면 메인 메뉴 버튼을 눌러 메인 화면으로 돌아갈 수 있다. 이때 점수는 기록되지 않는다.
    - 즉시 다시 시작

      게임 오버 시 try again? 형식으로 물어보며 즉시 기존 단계를 처음부터 실행할 수 있다.
    - 다음 레벨

      게임 클리어 시 next level 형식으로 등장하며 다음 레벨로 넘어간다. 3단계 클리어 시에는 에필로그가 재생된다.

주요 구현 기능

시나리오

1. 타이핑 구현

- `showNextParagraph()` 함수에서 `document.quertSelectorAll()`로 불러진 인트로 혹은 아웃트로의 p태그를 typeText()라는 함수로 넘긴다.

- `typeText()`에서 `charAt()`을 통해 p태그를 문자 하나하나씩 받아오며 실제로 보여지는 textContent 영역에 글자를 받아오는대로 하나씩 추가한다.
- space바를 누를 경우에는 하나하나씩 추가하던 작업을 멈추고 한꺼번에 원래 있던 문장을 보여준다.

2. 이미지 넘김 구현
   - `showimageNext()` 함수에서 텍스트 입력이 되는 시점과 동시에 image 번호값을 넘기면 `showImage()` 함수에서 `document.getElementById()`로 현재 띄워져 있는 이미지와 다음에 띄울 이미지를 가져와서 현재 띄워져 있는 이미지를 페이드 아웃 시킨 후 2초 대기했다가 다음에 띄울 이미지를 페이드 인 시켜서 구현하였다.
   - 모든 이미지가 다 보여지면 메인 메뉴로 자동 접근하게 구현하였다.

인터페이스

1. 기본 구성
   - 영어 폰트는 VT323, 검은 배경에 흰색 글씨로 통일했다.
   - 브라우저 정책 상 음악 자동실행이 불가하므로 press anywhere to start 화면으로 시작하여 클릭하면 음악을 실행하도록 하였다.
   - 메인 화면, 레벨 선택 화면, 게임 화면, 설정 화면, 게임 오버 및 클리어 화면을 구성하였고 `pageChange()` 함수를 통해 각각의 화면으로 이동할 수 있도록 구현했다.
   - Setting을 통해 음악 볼륨, 벽돌 색상, 공 색상을 사용자가 설정할 수 있게 하였다. 이때 Apply 버튼을 눌러야 변경된 값이 저장되고, Reset 버튼을 누르면 초기 값으로 변경, Back 버튼을 누르면 변경사항이 저장되지 않도록 구현하였다.
2. 인게임
   - 스페이스바를 통해 게임을 일시 정지할 수 있도록 하였고, 이때 창을 띄워 일시 정지된 상태임을 알리고 space를 다시 눌러 해제할 수 있음을 알려주도록 하였다. 이때 전역 변수 paused로 정지 상태를 관리하며, paused=true일 경우 정지 상태이므로 모든 Interval 구문에서 if (!paused) 조건을 사용하여 일시 정지를 구현하였다.
   - 인게임에서 메인 메뉴를 누른 경우에도 자동으로 일시 정지가 되도록 하였다.
   - 게임 화면은 게임 영역과 정보 영역으로 구분된다. 이때 게임 영역의 캔버스에서 게임이 진행되며, 우측의 정보 영역에는 남은 시간, 라이프, 벽돌 제거 점수, 최고 기록, 아이템을 표시해준다.
3. 클리어
   - 단계를 클리어한 경우에 남은 시간, 라이프 등을 합산하여 점수를 최고 기록과 함께 표시해준다.
     - 점수 계산 공식: (남은 시간 * 10) + (라이프 * 3) + 벽돌 제거 점수 + 보스 클리어 점수(5000)
   - 사용자는 Next Level을 눌러 다음 단계로 넘어가거나 Main Menu를 눌러 메인 메뉴로 돌아갈 수 있으나, 3단계를 클리어한 경우 Next 버튼만을 두어 에필로그로 넘어가도록 하였다.

인게임 기본 구현

인게임 구현 방식의 간단한 흐름도는 다음과 같다. `draw()` 함수는 `requestAnimationFrame(draw)`를 통해 매 프레임 호출된다. 이 사이클이 반복되면서 공이 튕기고, 보스가 움직이고, 플레이어 입력(마우스/키보드)도 실시간으로 반영된다.

```
gameStart() ──▶ initBricks() 등 초기 세팅 ──▶
|
```

```
▼
requestAnimationFrame(draw)
─ draw 시작 ──────────────────────────────────────────────────────
 │ 1) 캔버스 지우기
 │ 2) 벽돌·보스·공·아이템·패들 상태 그리기
 │ 3) 물리 작용 업데이트(충돌 검사, 점수, 속도 반전 등)
 │ 4) x, y 같은 변수 갱신
 │
 └─ requestAnimationFrame(draw) ──────────────────────▶ draw 재진입 (다음 프레임)
```

게임은 1단계, 2단계, 3단계로 이루어져 있으며 선택해서 플레이 할 수 있다. 단계가 높아질 수록 공의 속도가 빨라지며, 단계별로 해당하는 보스 그리고 플레이 할 수 있는 시간 또한 달라진다.

1. 공의 판정 및 움직임
   - 게임 시작 후 공은 자동적으로 우측 상단으로 발사가 된다. 공의 속도는 레벨이 높아질수록 `dx`, `dy`의 절댓값을 늘림으로써 공이 더 빠르게 움직이도록 하였다.
   - 이 때 공은 벽돌, 벽(canvas의 좌,우,상단), 패들과 부딪혔을 때 튕겨나가면서 반사가 된다. 반사가 되는 원리는 공이 장애물이 부딪혔을 때 공 x의 변화량(`dx`) 혹은 y의 변화량(`dy`)의 부호를 바꿔줌으로써 구현을 하였다.
   - 하지만 공이 바닥(canvas의 하단)에 떨어졌을 때, 다시 말해 하단 벽과 닿았을 때는 라이프를 잃게 된다. 이 때 떨어졌던 공은 패들 위에 부착이 되면서 클릭 할 시 재발사가 되어 게임이 이어진다.
   - 하지만 이 때 라이프가 더 이상 남아있지 않았다면 게임 오버가 된다.
2. 점수 시스템
   - 게임을 플레이 하면서 사용자는 벽돌을 깨면서 점수를 쌓을 수 있다. 이 때 해당 벽돌이 아이템 벽돌인지에 대한 여부와 현재 사용자가 아이템 버프(점수 증가 아이템)를 얻은 상태인지에 따라서 벽돌을 통해 얻을 수 있는 점수가 달라진다.
   - 정리를 해본다면, 각 상황에선 다음과 같은 점수가 주어진다.
     300점 : 점수 증가 버프 상태에서 아이템 벽돌을 깼을 때
     200점 : 아이템 벽돌을 깼을 때 혹은 점수 증가 버프 상태에서 일반 벽돌을 깼을 때
     100점 : 일반 벽돌을 깼을 때
   - 단, 단계를 클리어 했을 때 최종 점수는 남은 시간, 남은 라이프(목숨), 보스 클리어에 따라서 합산된다.
3. 패들 판정 및 움직임
   - 마우스의 움직임에 따라 패들이 좌우로 부드럽게 따라 움직인다.

```
canvas.addEventListener("mousemove", e => {
  const rect = canvas.getBoundingClientRect();
  const mx = e.clientX - rect.left;
  paddleX = mx - paddleWidth / 2;
  if (paddleX < 0) paddleX = 0;
  if (paddleX > canvas.width - paddleWidth) paddleX = canvas.width - paddleWidth;
});
```

- 다음 코드와 같이 해당 조건에 충족이 된다면 패들을 통해서 아래로 내려오던 공의 속도(dy)의 부호를 반전 시킴으로써 다시 위로 반사 시킨다.

```
if (
    dy > 0 &&                          // 공이 아래로 내려오는 중일 때만
    y + ballRadius <= paddleTop &&      // 현재 프레임 공이 패들 상단 위에 있고
    nextY + ballRadius >= paddleTop &&  // 다음 프레임에 패들 상단을 넘으면
    nextX > paddleX &&                  // 공의 X좌표가 패들 왼쪽 경계보다 크고
    nextX < paddleX + paddleWidth       // 공의 X좌표가 패들 오른쪽 경계보다 작다면
) {
    dy = -dy;                          // 충돌 시 y축 속도 부호 반전 → 위로 튕김
    y = paddleTop - ballRadius;         // 위치 보정: 공이 패들 상단 바로 위에 놓이도록
}
```

4. 목숨(라이프) 시스템
   - 게임 시작 후, 사용자는 3개의 라이프가 주어진다. 사용자의 라이프는
     (1) 공이 바닥으로 떨어졌을 때,
     (2) 보스 몬스터의 공격을 맞았을 때
     한개씩 소모된다.
   - 라이프가 모두 소진되어 0이 되었을 때는 게임 오버가 되며 다시 플레이 하거나 메인 메뉴로 돌아가는 버튼이 주어지는 화면으로 넘어간다.
   - 라이프가 하나씩 줄어들수록 패들 크기도 조금씩 작아지며 반대로 라이프가 하나씩 늘어날수록 패들 크기도 조금씩 늘어난다. 단, 라이프가 5로 늘어나는 순간부터는 패들 크기가 늘어나지 않으며 마찬가지로 라이프가 3으로 줄어들기 전까지 패들 크기도 줄어들지 않는다.
5. 벽돌 시스템 등에 대한 설명 작성
   - 벽돌 초기화

     우선 벽돌(bricks)는 2차원 배열로 구상을 하였고 배열 안에는 status 값을 할당하여 벽돌의 존재 여부를 판단하였다. 이 status 값에는 추후에 벽돌이 파괴되었을 때 0이 부여되고 파괴되지 않았을 때는 1인 상태가 된다.

```
// 벽돌 배열 초기화
function initBricks() {
  bricks = []; brickWidth = canvas.width / brickColumnCount;
  for (let c = 0; c < brickColumnCount; c++) {
    bricks[c] = [];
    for (let r = 0; r < brickRowCount; r++) {
      bricks[c][r] = { status: 1 };
    }
  }
}
```

   - 게임 진행 중 벽돌 한 줄 씩 추가

     게임이 진행 되면서 일정 시간 간격으로 벽돌이 한 줄(한 행) 생성이 된다. 이는 보스 미출현 시 해당 사항이다. 보스가 출현하고 나서는 벽돌이 추가 되지 않는다.

```
// 벽돌 위에서 한줄 추가
function addBrickRow() {
  brickRowCount++;
  for (let c = 0; c < brickColumnCount; c++) {
    bricks[c].unshift({ status: 1 });
  }
}
```

- 벽돌 충돌 여부 검사

  해당 코드는 공이 벽돌과 부딪혔는지 매 프레임마다 검사해서 부딪혔으면(공의 경계가 벽돌 사각형의 영역과 겹쳤으면) 공의 속도를 반사 방향으로 뒤집는다(dy 또는 dx 반전). 이 때 해당 벽돌의 status를 0으로 바꿔 깨진 상태로 처리한다.

```
// draw() 내부, 공 이동·충돌 로직 중

    // 충돌 여부 검사:
    if (
      b.status &&                          // 아직 깨지지 않은 벽돌일 때만
      nextX + ballRadius > bx &&           // 공의 오른쪽이 벽돌 왼쪽 넘어섰고
      nextX - ballRadius < bx + brickWidth &&   // 공의 왼쪽이 벽돌 오른쪽 안쪽에 있고
      nextY + ballRadius > by &&           // 공의 아래쪽이 벽돌 위쪽 넘어섰고
      nextY - ballRadius < by + brickHeight     // 공의 위쪽이 벽돌 아래쪽 안쪽에 있을 때
) {

  // 이전 위치로 충돌 방향 판정
  if (!invEnable) {
    const prevX = x - dx;
    const prevY = y - dy;

    if (prevY <= by || prevY >= by + bh) {  // 위아래에서 충돌
      dy = -dy;
       } else {    // 좌우에서 충돌
      dx = -dx;
    }
  }

  // status 를 0 으로 바꿔 깨짐 처리
  b.status = 0;
}
```

아이템

게임 시작 5초 후 처음 등장하며 이후 15초 간격으로 등장하는 아이템 벽돌을 공으로 맞춰 사용할 수 있다.

아이템 벽돌은 현재 존재하는 일반적인 벽돌 중에서 무작위로 하나를 선택해 아이템 벽돌로 변경시킨다.

1. 기본 아이템 시스템 구현

- 아이템 벽돌 구현

  아이템 벽돌 생성을 시도하는 함수가 현재 존재하는 벽돌들을 모두 수집하여 그 중에서 하나를 랜덤으로 선택하여 아이템 속성을 부여하도록 하였다.

- 키보드에 따른 작동 방식

  키보드를 눌렀을 때 쿨타임이 없는 상태에서 아이템이 존재할 때에만 (`addEventListener`)의 (`keydown`) 이벤트를 통해서 키 입력 신호가 전달됨에 따라 키에 설정된 아이템 발동 함수가 작동된다.

2. 패시브 아이템

   패시브 아이템은 아이템 벽돌을 맞추는 순간 바로 작동되는 아이템이다. 테두리가 시안색으로 둘러져 있으며 각각에 맞는 이미지가 그려져 있다.

   - 시간 추가 아이템
     - 게임 진행 가능 시간을 늘려주는 아이템이다.
     - 게임의 남은 시간은 (`left`)라는 전역 변수가 관리하며 해당 값을 단순히 10을 추가해줌으로써 구현하였다.

       ```
       function timeAdd() {
         startClockSfx(); // 효과음
         left += 10; // 전역 변수 값 추가
       }
       ```

   - 생명 추가 아이템
     - 생명을 1만큼 늘려주는 아이템이다.
     - 생명이 4 이하이면 패들의 길이도 늘어나며 반대로 4 이상이라면 패들의 길이는 더이상 늘어나지 않고 생명만 증가한다.
     - dom 모델에서 (`.current-life`) class에 있는 요소를 불러와 직접 값을 늘려 구현하였다.

       ```
       function lifeAdd() {
         const info = document.getElementById(`level${level}`);
         const lifeEl = info.querySelector(".current-life"); // dom 모델로 받아오기
         let currentLife = parseInt(lifeEl.textContent); // 문자에서 숫자로
         if (currentLife < 4) {
           paddleWidth += 40; // 생명 추가에 따른 패들 추가
         }
         currentLife++; // 생명 추가
         lifeEl.textContent = currentLife; // 반영
         startLifeSfx(); // 효과음
       }
       ```

3. 액티브 아이템

   액티브 아이템은 아이템 벽돌을 맞추면 저장이 되며 원할 때 사용할 수 있는 아이템이다.

   액티브 아이템은 서로 중첩이 가능하다.

   - 공격 아이템
     - A키를 눌러 사용 가능, 쿨타임 10초

- 키를 누름과 동시에 반원 모양의 투사체 2개를 발사한다.
- 투사체는 현재 패들의 너비와 동일한 너비를 가진다.
- 점수, 보스 피격 데미지는 공으로 맞췄을때와 동일하다.
- 투사체는 벽돌, 보스에 맞으면 사라지며 오브젝트 튕김 무시 아이템을 활성화하면 화면 끝까지 발사된다.
- 투사체는 아이템이 있는 상태에서 A키를 누르면 투사체 배열을 생성한다.

  공처럼 매 순간순간마다 배열에 들어온 값이 있는지 체크하고 들어온 값이 있다면 (`setInterval()`)을 통해 반복해서 그려지며 (`drawProjectiles()`)로 반원 투사체를 그린 후 블록의 x, y값을 받아와 충돌 판정, 보스의 x, y값을 받아와 보스 판정을 수행하였다.

```
if (e.key === "a" || e.key === "A") {
  if (availableAttack > 0 && attackCool == false) {
    attack(-1); // dom에 접근하여 숫자 1 감소시킴
    attackCool = true;
    attackTime(5); // setInterval을 통한 바로 쿨타임 활성화
    startAtkSfx();

    const projY = canvas.height - paddleHeight - imgH;
    const projXCenter = paddleX + paddleWidth / 2;

       // 키보드 누를 때만 투사체 배열에 투사체 생성
    projectiles.push(createProjectile(projXCenter, projY));
    projectiles.push(createProjectile(projXCenter, projY - 30));
  }
}
```

```
function createProjectile(x, y) { // 투사체 만들기
  return {
    x,
    y, // 투사체 좌표
    radius: paddleWidth / 2, // 투사체 반지름
    speed: 8, // 투사체 속도
    projLife: 120, // 투사체 지속 시간
  };
}

function drawProjectiles() { // 투사체 그리기
  for (let i = projectiles.length - 1; i >= 0; i--) {
    const p = projectiles[i];

    // 반원 위쪽 방향 그리기
    ctx.beginPath();
    ctx.arc(p.x, p.y, p.radius, Math.PI, 0); // 반원
    ctx.strokeStyle = ballColor;
    ctx.lineWidth = 2;
```

```
    ctx.stroke();
    ctx.closePath();

    // 위로 이동
    p.y -= p.speed;
    p.projLife--;

    // 벽돌 충돌 처리
    (로직 생략, 공에 대한 벽돌 충돌과 원리 동일)

      // 보스 충돌 처리
    if (checkBossCollision(p.x, p.y, p.radius)) {
      if (!invEnable) {
        projectiles.splice(i, 1);
      }
    }

       // 투사체가 밖으로 나갔을 때
    if (p.y < -10 || p.projLife <= 0) {
      projectiles.splice(i, 1);
    }
  }
}
```

- 점수 & 보스 피격 데미지 증가 아이템
  - S키를 눌러 사용 가능, 활성화 시간 30초, 쿨타임 10초
  - 벽돌을 맞췄을 때 추가 점수 100점, 보스에게 입히는 공격력 1 증가
  - 벽돌 점수 추가 옵션의 경우 bool형 변수 (`damageEnable`)로 설정하였으며 해당 설정이 true일 때 스코어 추가 방식을 조건문 분기를 통해 다르게 적용하여 구현하였다.
  - 보스 공격의 경우에도 해당 옵션이 켜져있으면 HP 감소 방식을 조건문으로 분기하여 추가 데미지를 입히도록 구현하였다.

```
if (b.isItem) { // 벽돌 충돌 판정 점수, isItem은 아이템 벽돌 여부
  if (damageEnable == true) {
    score += 300;
  } else {
    score += 200;
  }
  applyItemEffect(b.itemType);
} else {
  if (damageEnable == true) {
    score += 200;
  } else {
    score += 100;
  }
}
```

```
if (damageEnable) { // 보스 충돌 판정 데미지
  boss.hp -= 2;
} else {
  boss.hp--;
}
```

- 오브젝트 튕김 무시 아이템
  - D키를 눌러 사용 가능, 활성화 시간 15초 쿨타임 15초
  - 공이 벽돌, 보스를 맞춰도 튕겨나가지 않고 보스의 경우에는 그대로 통과한다. 단 패들은 이를 무시하고 해당 상태에서도 계속 튕겨낼 수 있다.
  - 보스를 통과할 때 공과 공격 투사체가 보스 히트박스 내에 존재할 경우 1초간 지속 데미지를 입힌다. (지속 데미지는 피격 데미지와 동일)
  - 해당 방식의 경우 bool형 변수 (`invEnable`)로 설정하였으며 해당 설정이 true일 경우 공에 대한 벽돌 판정 옵션 코드가 수행이 되지 않도록 그리고 공에 대한 보스 판정 옵션 코드가 수행히 되지 않도록 구현하였다.

```
if (!invEnable) { // 벽돌에 부딪혔을 때 조건 안에서 해당 아이템이
  const prevX = x - dx; // 켜져 있으면 충돌 판정 무시
  const prevY = y - dy;

  if (prevY <= by || prevY >= by + bh) {
    dy = -dy;
  } else {
    dx = -dx;
  }
}

if (!invEnable) { // 공격 투사체도 아이템이 꺼져있을 때만 충돌 시 사라짐
    projectiles.splice(i, 1);
}
```

보스

1. 보스 등장 시점 및 조건

   게임은 3개의 레벨(Level 1~3)로 구성되어 있으며, 각 레벨마다 보스가 등장한다. 보스는 일정 시간이 지난 뒤 자동으로 등장하며, 그 전까지는 일반 벽돌깨기 형식으로 진행된다.

   - **등장 조건**
     - `step`(게임 시간, 초 단위)이 65 이상일 경우 경고 표시
     - `step`이 70 이상이면 보스 등장 (`spawnBoss()` 호출)
   - **연출**
     - 경고 사운드(`startDgrSfx`)와 시각적 강조(`dangerInfo()`)를 통해 등장 예고
     - 이후 `dangerClear()` 호출로 시각 효과 해제

- **배경 음악 변경**
  - 보스 등장과 함께 각 레벨 전용 보스 BGM으로 전환 (`playBgm(7~9)`)

2. 보스 기본 구조

   보스는 게임 내부 전역 객체 `boss`로 정의되며, 다음과 같은 속성을 갖는다:

   - 위치 (`x` , `y`)
   - 크기 (`width` , `height`)
   - 체력 (`hp`)
   - 활성화 여부 (`active`)
   - 프레임 기반 애니메이션 정보 (프레임 배열, 현재 프레임 등)

   보스의 초기화는 `initBoss()` 함수에서 이루어지며, 레벨별로 보스의 체력과 이미지 경로, 프레임 수가 설정된다.

3. 보스 행동 패턴

   보스는 등장 이후 자동으로 움직이고 공격한다. 움직임과 공격은 다음과 같은 로직으로 구성되어 있다:

   - 3.1 이동
     - 보스는 고정된 X좌표를 따라 좌우로 이동
     - `moveBoss()` 함수에 의해 매 프레임 실행
   - 3.2 공격 (`bossAttack()`)
     - **레벨 1:**
       - 정면으로 돌진하여 캐릭터와 충돌 시 피해
     - **레벨 2:**
       - 정면으로 1개의 투사체 발사
     - **레벨 3:**
       - 정면과 양쪽에서 3개의 투사체를 동시에 발사

     보스의 투사체는 배열 `bossProjectiles`에 저장되어 매 프레임 `draw()` 내에서 이동, 그리기, 충돌 판정이 수행됨.

4. 보스 피격 및 체력 관리
   - 플레이어의 공 또는 `attack` 스킬에 의해 보스는 피해를 입는다.
   - 피격 위치에 따라 이펙트가 다르게 표현될 수 있도록 분기 처리되어 있음.
   - 보스의 체력이 0이 되면 제거되며 게임 클리어 상태로 전환됨.

5. 보스와 플레이어의 상호작용
   - 보스의 투사체가 패들에 닿으면 목숨이 감소 (`gameOver()` 내에서 처리)
   - 보스를 공격하기 위해서는 다음과 같은 수단이 존재:
     - **공:** 일반적인 벽돌깨기처럼 반사되는 공
     - **스킬 A (공격):** `A` 키를 누르면 참격 형태의 공격이 발사되어 보스에게 피해를 줌
     - **스킬 S (점수 버프, 보스가 입는 데미지 증가):** 공격 시 점수 상승 및 피해 증가
     - **스킬 D (공의 보스 통과):** 잠시 무적 상태로 보스 투사체에 피격되지 않음

6. 보스 전용 효과 및 시각 요소
    - 등장 시 시각 강조 효과 (붉은 테두리 및 `dangerInfo()` 호출)
    - 각 보스는 고유한 이미지와 애니메이션 프레임을 가진다.
    - 체력바는 보스 아래에 위치하며 시각적으로 표시된다.
    - 공격 이펙트 및 충돌 효과는 오디오(`sound/damage.mp3` 등)와 연동됨.

토의 사항

1. 개선점
    - 코드 중복 제거 및 모듈화 필요
        - `goLv1()`, `goLv2()`, `goLv3()` 와 같은 함수들은 로직이 거의 동일하므로 하나의 함수로 통합해야 더 효율적이었을 것이라고 생각한다.
        - 페이지 전환 관련 함수들이 많은데, `changePage(i)` 만으로 대부분 처리되므로 이벤트 핸들러만 잘 정리해도 코드가 간결해질 수 있었을 것이다.
    - 불필요한 글로벌 변수 다수 존재
        - ex: `projectiles`, `step`, `left`, `paused` 등 수십 개의 전역 변수는 추후 유지보수에 치명적임.
        - `GameState` 객체 등으로 구조화하면 관리와 테스트가 수월해지지 않았을까 생각한다.
    - Accessibility 및 모바일 대응 부족
        - 키보드와 마우스 외 입력 수단에 대한 고려가 부족했다.
        - CSS가 PC 기준으로만 짜여 있다. (예: `width: 920px` 고정값 사용 등).
2. 소감
    - 시나리오와 게임을 연결하는 과정이 인상적이었다.
        - 단순한 벽돌깨기 게임이 아니라, 인트로와 에필로그를 통해 몰입감을 주는 연출이 가능하다는 점을 새롭게 느꼈다.
        - 특히 `typeText()` 와 이미지 페이드 효과는 생각보다 몰입도가 높아서 UI 연출에 대한 흥미가 생겼다.
    - 효과음과 BGM 처리의 중요성을 체감했다.
        - 처음에는 사소해보였던 배경음악과 효과음이, 실제로 구현하면서 사용자의 긴장감과 몰입도를 크게 좌우한다는 걸 느꼈다.
        - 이를 위해 preload, play, pause를 세심하게 처리해야 한다는 걸 배웠다.
    - 코드를 작성하는 것보다 '조직'하는 것이 더 중요했다.
        - 단순히 동작하는 것에 그치지 않고, 화면 전환, 상태 유지, 스코어 관리 등을 통합적으로 관리하려다 보니 전역 상태나 이벤트 흐름이 꼬이기 쉬웠다.
        - 다음에는 게임의 흐름이나 상태 전이를 더 명확하게 설계한 뒤에 구현을 시작할 계획이다.
    - 함께 만든다는 협업의 어려움과 보람을 느낄 수 있었다.
        - 파일이 많아지고 역할이 분담되면서 충돌이 발생하거나 누가 어느 코드를 담당했는지 모호해지는 경우가 종종 있었다.

기여도

김경수 : 25% : 보스 몹, 보스 몹 이미지 구현

김도현 : 25% : 벽돌과 공의 움직임 구현

배정환 : 25% : 아이템, 시나리오, 전반적인 이미지 구현

추현규 : 25% : 인터페이스, 배경 음악, 환경 설정, 기본 틀 구현