Bridge of Fire

20200298 컴퓨터학과 장고은

프로젝트명

초기 기획안의 명칭을 유지하여, 무너지는 다리와 불덩이라는 핵심 요소를 포함하는 **Bridge of Fire**로 최종 확정하였다.

게임 개요

• Bridge of Fire는 2D 횡스크롤 플랫폼 액션 게임이다.
• 제한 시간(60초) 내에 무너지는 다리를 건너며, 사방에서 날아오는 불덩이를 피해 목표 지점(GoldKey)에 도달하는 것을 목표로 한다.
• 실제 구현: SceneManager를 통해 GameScene, GameOver, GameClear 씬 간의 유기적인 전환을 구현하였으며, BGMPlayer를 통해 씬 전환 시에도 음악이 끊기지 않는 환경을 조성하였다.

재미 요소

1. 날아오는 불덩이를 피해야 한다. 불덩이와 캐릭터가 충돌할 경우 HP가 10씩 감소한다. (HP는 100부터 시작한다.)
2. 다리는 징검다리 형식으로 되어 있고, 특정 다리에 사용자가 올라간지 n초가 지나면 특정 다리에 중력이
적용되어 무너지는 느낌을 준다.
3. 종점에 가까워질수록 불덩이가 날아오는 속도가 빨라지거나 다리의 간격을 넓혀 난이도를 올린다.
4. 플레이어 이동에 애니메이션을 적용해 생동감 있는 움직임을 표현한다.
5. 점프/불덩이가 날아오는 효과음.. 등 사운드를 추가하여 사용자로 하여금 게임 진행에 긴장감을 갖도록 한다.

상세 결과 (최종 시스템 설계)

1단계: 화면에 놓인 오브젝트 리스트

cat (Player)	Rigidbody2D 기반의 조작 캐릭터 애니메이터를 통해 이동/점프 모션 표현
FireBall	Prefab으로 관리됨 플레이어와 충돌 시 데미지를 입히고 소멸하는 장애물
Bridge	평소에는 고정되어 있으나 플레이어가 밟으면 일정 시간 후 낙하하는 발판
Goal	게임 클리어 판정을 위한 목적지 트리거 오브젝트
DeathZone	다리에서 낙하 시 충돌되는 바닥(사망 처리용)
UI (Time/HP)	TextMeshProUGUI를 사용하여 남은 시간과 현재 체력을 실시간 표시
TIME UI	TIME을 표시하는 UI

2단계: 오브젝트를 움직일 수 있는 컨트롤러 스크립트

PlayerController	• AddForce를 이용한 좌우 이동 및 점프 로직 구현 • localScale을 이용해 이동 방향에 따른 캐릭터 좌우 반전 처리 • 추락 시(-10f 이하) 또는 클리어 지점 도달 시 코루틴을 통한 결과 씬 전환 제어
FireBallController	• Translate를 이용해 불덩이를 왼쪽으로 이동시키며, 화면 밖 이탈 시 메모리 관리를 위해 자동 삭제 • 플레이어와 충돌 시 데미지 계산 및 효과음 재생 후 오브젝트 파괴
BridgeController	• OnCollisionEnter2D로 플레이어 접촉을 감지 • IEnumerator를 활용해 3초의 대기 시간 후 bodyType을 Dynamic으로 변경하여 낙하 구현

```
// BridgeController.cs 핵심 로직
private void OnCollisionEnter2D(Collision2D collision) {
    if (collision.gameObject.CompareTag("Player")) {
        StartCoroutine(EnableGravityAfterDelay(3f)); // 3초 대기 후 낙하
    }
}
IEnumerator EnableGravityAfterDelay(float delay) {
    yield return new WaitForSeconds(delay);
    rigid2D.bodyType = RigidbodyType2D.Dynamic; // 물리 타입 변경
}
```

CameraController	• 플레이어(cat)의 X축 좌표가 특정 범위(0~53) 내에 있을 때만 카메라가 플레이어를 추적하도록 제한하여 맵 밖 노출 방지

3단계: 오브젝트를 자동으로 생성하는 제너레이터 스크립트

FireBallGenerator	• delta 시간을 누적하여 설정된 주기(span)마다 불덩이 프리팹을 생성 • Random.Range(-4, 4)를 통해 불덩이가 생성되는 Y축 높이를 무작위로 설정하여 난이도 부여

4단계: UI 갱신 및 게임 흐름 관리 감독 스크립트

GameDirector	• 실시간 UI 갱신: HP 텍스트와 타이머 텍스트를 매 프레임 업데이트 • 가변 난이도 시스템: 남은 시간에 따라 불덩이 생성 주기(span)를 3.0초에서 점진적으로 0.6초까지 단축시켜 긴장감 유도 • 상태 관리: 체력 0 또는 시간 종료 시 BGM을 멈추고 게임 오버 사운드 재생 후 씬 전환

```
// GameDirector.cs 난이도 조절 부분
void Update()
{
    if (isGameOver) return;
    this.time -= Time.deltaTime;
```

```
    if (this.time > 50)
        this.generator.GetComponent<FireBallGenerator>().SetParameter(3.0f);
    else if (this.time > 40)
        this.generator.GetComponent<FireBallGenerator>().SetParameter(2.5f);
    else if (this.time > 30)
        this.generator.GetComponent<FireBallGenerator>().SetParameter(2.0f);
    else if (this.time > 20)
        this.generator.GetComponent<FireBallGenerator>().SetParameter(1.5f);
    else if (this.time > 10)
        this.generator.GetComponent<FireBallGenerator>().SetParameter(1.0f);
    else
        this.generator.GetComponent<FireBallGenerator>().SetParameter(0.6f);

    this.timerText.GetComponent<TextMeshProUGUI>().text = "Time " +
this.time.ToString("F1");

    if (this.time <= 0)
    {
        TriggerGameOver();
    }
}
```

ClearDirector	• 클리어/게임 오버 화면에서 마우스 클릭 시 다시 게임 씬으로 돌아가는 흐름 제어
BGMPlayer	• DontDestroyOnLoad를 사용하여 씬이 바뀌어도 음악이 유지되도록 처리(싱글톤)

```
// BGMPlayer.cs 싱글톤 구현
void Awake() {
    if (instance == null) {
        instance = this;
        DontDestroyOnLoad(gameObject); // 씬 전환 시 파괴 방지
    } else {
        Destroy(gameObject); // 중복 생성 방지
    }
}
```

수정 결론

초기 기획 대비, 실제 개발 과정에서 BGM 지속성(싱글톤), 남은 시간에 따른 난이도 가속 시스템, 카메라 추적 범위 제한 등 게임의 완성도를 높이기 위한 디테일한 로직들이 코드에 추가되었다. 특히 다리 낙하 지연 시간(3초)과 불덩이 생성 로직이 유기적으로 맞물려 기획했던 '긴장감 있는 탈출'이 성공적으로 구현됐다.