



국민대학교  
소프트웨어융합대학  
소프트웨어학부


# 캡스톤 디자인 I

## 종합설계 프로젝트

프로젝트 명	<i>Tall Tales</i>
팀 명	<i>Choi's</i>
문서 제목	결과보고서

Version	1.2
Date	2022-JUN-17

팀원	최 민혁 (조장)
	최 원준

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

#### CONFIDENTIALITY/SECURITY WARNING

이 문서에 포함되어 있는 정보는 국민대학교 소프트웨어융합대학 소프트웨어학부 및 소프트웨어학부 개설 교과목 다학제간캡스톤디자인I 수강 학생 중 프로젝트 "Tall tales"를 수행하는 팀 "Choi's"의 팀원들의 자산입니다. 국민대학교 소프트웨어학부 및 팀 "46팀"의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다.

## 문서 정보 / 수정 내역

<b>Filename</b>	최종보고서-Tall Tales.doc
<b>원안작성자</b>	최원준
<b>수정작업자</b>	최원준

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2022-05-17	최원준	1.0	최초 작성	
2022-05-20	최원준	1.1	사진추가	
2022-05-25	최원준	1.2	자기평가 추가	

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 목 차

1	개요 .....	4
1.1	프로젝트 개요 .....	4
1.2	추진 배경 및 필요성 .....	4
2	개발 내용 및 결과물 .....	5
2.1	목표 .....	5
2.2	연구/개발 내용 및 결과물 .....	6
2.2.1	연구/개발 내용 .....	6
2.2.2	시스템 기능 요구사항 .....	25
2.2.3	시스템 구조 및 설계도 .....	26
2.2.4	활용/개발된 기술 .....	27
2.2.5	현실적 제한 요소 및 그 해결 방안 .....	33
2.2.6	결과물 목록 .....	33
3	자기평가 .....	34
4	참고 문헌 .....	35
5	부록 .....	36
5.1	사용자 매뉴얼 .....	36
5.2	배포 가이드 .....	36
5.3	테스트 케이스 .....	37

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

# 1 개요

## 1.1 프로젝트 개요

프로젝트의 주제는 2D 종스크롤 슈팅 게임을 개발하는 것이다.

기본적으로 위에서 적이 내려오며 플레이어를 공격하고, 플레이어는 공격을 피하면서 적을 공격해 점수를 모으고 스테이지를 클리어해, 보스 스테이지에 도달하게 된다.

보스 스테이지에서 보스를 물리치면 게임의 엔딩을 볼 수 있다.

전체 구성은 게임을 더욱 편하게 이끌어줄 상점 시스템과, 게임의 스토리 등을 확인할 수 있는 게임 로비 부분과, 본격적인 게임을 하는 인 게임 부분으로 나뉘어져 있다, 플레이어는 인게임과 로비의 이동을 반복하며 게임 클리어에 도전하게 된다.

## 1.2 추진 배경 및 필요성

높아지는 그래픽과, 사양이 주류가 된 최근의 게임 시장에서 간단한 2D 도트 게임이지만 큰 인기를 끈 '뱀파이어 서바이벌' 이라는 게임이 있었다. 이 게임을 보면서 왜 유저들이 이런 간단한 게임에 열광했는지에 대해 생각해 보게 되었고, '간단하게 즐길 수 있다', '캐릭터를 성장시키는 재미가 있다' 라는 두가지 요소에 주목하게 되었다.

이를 통해 바쁜 현대인들이, 간단하게 즐길 수 있으면서도, 강한 중독성으로 계속해서 플레이를 할 수 있는 게임을 만들어 보고싶다는 생각을 하게 되었다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2 개발 내용 및 결과물

### 2.1 목표

게임 씬 3 개로 구성된 게임을 개발한다.

게임 타이틀 씬, 게임 로비 씬, 인게임 씬 이다.

게임 로비에서는 게임 스토리를 확인할 수 있는 NPC 와 플레이어의 스텟을 올려주는  
상점 NPC 를 개발한다.

포탈을 만들어 포탈을 통해 인게임으로 이동하게 한다.

위에서 생성되어 각자 다른 움직임으로 플레이어를 공격하는 적들을 개발한다.

보스는 여러가지 패턴을 가지고, 일정 체력별로 다른 패턴을 보여주게 개발한다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2.2 연구/개발 내용 및 결과물

### 2.2.1 연구/개발 내용

#### 씬이동

```

public class Loading : MonoBehaviour
{
    public static string nextScene;
    private bool canOpen = true;

    private void Start()
    {
        StartCoroutine(LoadScene());
    }

    string nextSceneName;
    public static void LoadScene(string sceneName)
    {
        nextScene = sceneName;
        SceneManager.LoadScene("Loading");
    }
}

```

씬이동 함수를 정적 함수로 두어서, 여러 씬 에서 이동할 때 함수를 불러서 사용할 수 있게 만들었다. 함수의 매개변수는 이동할 씬의 이름이 된다.

우리 게임에서는 처음 타이틀에서 게임 로비에서, 인게임에서 게임 로비로 돌아올 때 쓰였다.

 국민대학교 소프트웨어학부 다학제간캡스톤디자인I	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 적의 움직임

### 호박

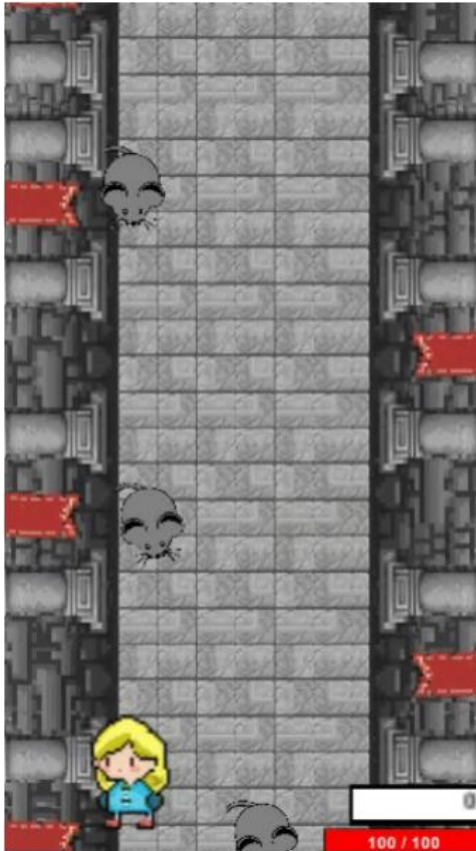


```
void simpleMove()
{
    float distanceY = Speed * Time.deltaTime;
    this.gameObject.transform.Translate(0, -1 * distanceY, 0);
}
```

전 프레임이 완료되기까지 걸린 시간인 Time.deltaTime 을 이용해 한 프레임에 같은 이동값을 가지게 하고, Y축을 이동하게 만들었다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

쥬



```
private void Start()
{
    if (enemyName == "enemy001")
    {
        dirVec = Player.instance.transform.position - transform.position;
        float angle = Mathf.Atan2(dirVec.y, dirVec.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.AngleAxis(angle + 90, Vector3.forward);
    }
}

void moveControl()
{
    transform.position = transform.position + dirVec.normalized * Speed * Time.deltaTime;
}
```

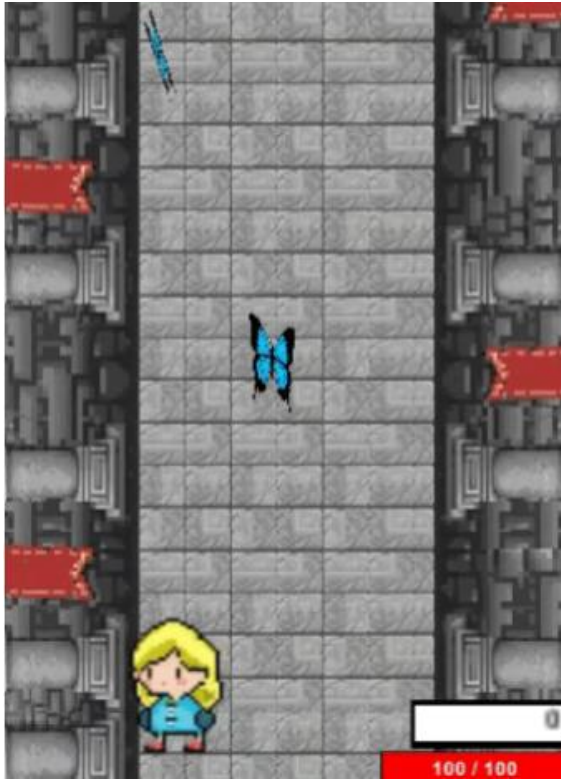
기본 이동은 호박과 동일한 일직선 움직임이다.

대신 생성되는 순간 플레이어의 위치 값을 받아오고, angle 값에 플레이어 위치를 향한 각도를 계산하고 이동하게 만들었다.



 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 나비

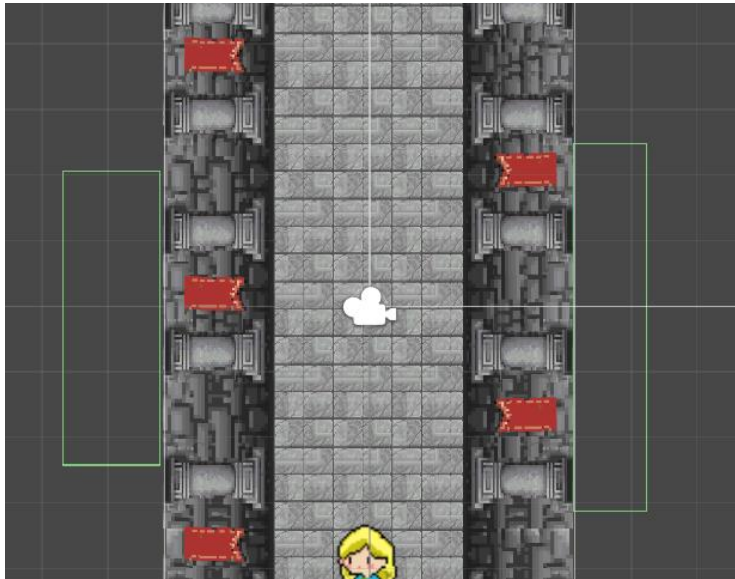


```
void fly()
{
    transform.Rotate(Player.instance.transform.position - transform.position); // 팔각대기
    transform.position = transform.position + (Player.instance.transform.position - transform.position).normalized * Speed * Time.deltaTime; // 쫓아오기
}
```

플레이어 방향을 계속 계산하며, 플레이어를 따라가게 만들었다.



## 펜



펜은 기존 적들과는 다르게 위에서 내려오는 것이 아닌 옆에서 나오도록 하였다.

플레이어가 위만 신경 쓰는 것이 아니라 옆에서 나오는 적도 신경 쓰게 하여서 더 재밌고 어렵게 느끼도록 하려고 하였다. 우선 맵의 옆쪽에 펜이 생성될 영역을 만들어주었다.

```
private IEnumerator Spawn()
{
    while(true)
    {
        yield return new WaitForSeconds(delay);

        Vector3 spawnPos = get_RandomPosition();
        if(dir == 1)
        {
            GameObject instance = objectManager.MakeObj("penRight");
            instance.transform.position = spawnPos;
        }
        else
        {
            GameObject instance = objectManager.MakeObj("penLeft");
            instance.transform.position = spawnPos;
        }
    }
}
```

그리고 그 정해진 영역 내에서 랜덤으로 위치를 생성해 펜 객체를 생성하게 하였다.

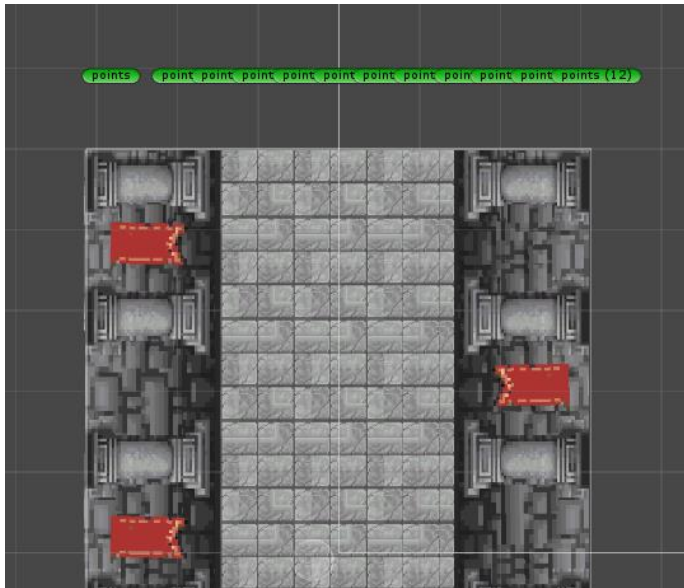
펜 객체는 호박과 같은 일직선 움직임이지만, 방향이 x축 방향이다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

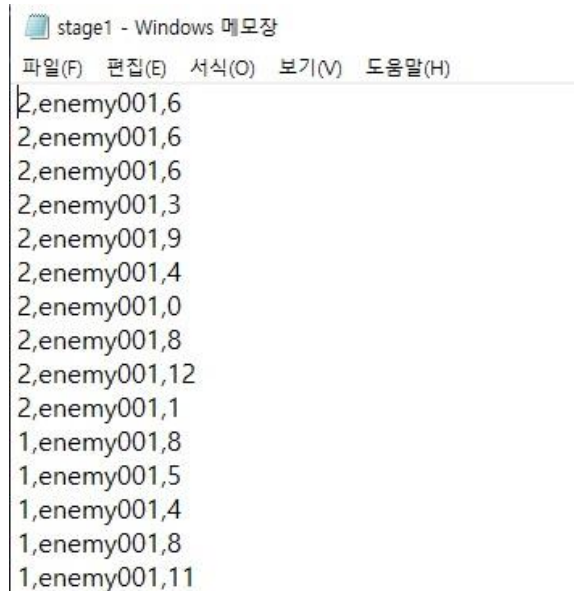
### 적 스폰

위와 같은 펜을 제외한 3가지의 타입의 적들은 모두 위에서 생성되어 내려온다.

펜을 생성했던 것처럼 위에서 영역을 지정해주고, 랜덤하게 생성되게 할 수도 있지만, 조금 더 난이도를 높이고 재밌게 만들기 위해서, 직접 적을 언제 어디서 스폰할 수 있게 스테이지 파일을 만들어 관리하기로 하였다.



맵 위에 적들이 생성될 영역을 만들고 위치를 12개로 잡아주었다.



텍스트 파일에, 생성되는 시간, 생성될 객체의 이름, 생성될 위치 포인트 3가지의 정보를 적은 스테이지 파일을 만들었다.



```
void ReadSpawnFile()
{
    // 변수 초기화
    spawnList.Clear();
    spawnIndex = 0;
    spawnEnd = false;
    // 리스폰 파일 읽기
    TextAsset textFile = Resources.Load("stage1") as TextAsset;
    StringReader stringReader = new StringReader(textFile.text);

    while (stringReader != null)
    {
        string line = stringReader.ReadLine();

        if (line == null)
            break;
        // 리스폰 데이터 생성
        Spawn spawnData = gameObject.AddComponent<Spawn>();
        spawnData.delay = float.Parse(line.Split(',')[0]);
        spawnData.type = line.Split(',')[1];
        spawnData.point = int.Parse(line.Split(',')[2]);
        spawnList.Add(spawnData);
    }
    // 텍스트파일 닫기
    stringReader.Close();
    nextSpawnDelay = spawnList[0].delay;
}
```

그리고 그 스테이지 파일을 코드에서 읽어서 동작하도록 하였다.

텍스트 파일을 열고, 각 항목들을 변수에 저장한 뒤에 동작 하도록 만들었다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 상점 구현

### 로비상점



```

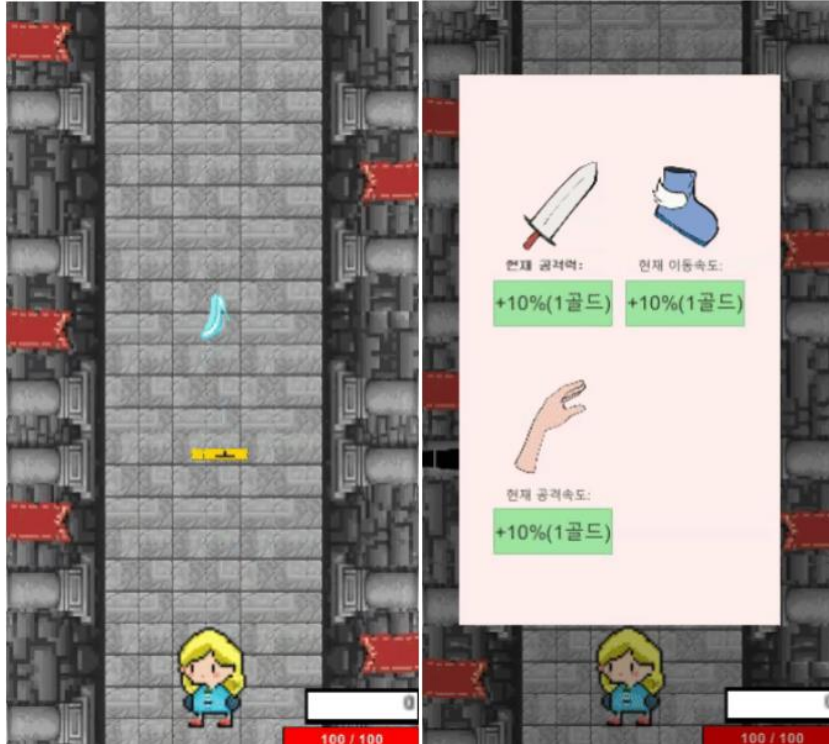
public void UpgradePower()
{
    //Debug.Log("888888 888888?");
    if (!PlayerPrefs.HasKey("Power")) {
        PlayerPrefs.SetInt("Power", 0);}
    if (!PlayerPrefs.HasKey("Price1"))
        PlayerPrefs.SetInt("Price1", 1);
    if (PlayerPrefs.HasKey("Ink"))
    {
        if (PlayerPrefs.GetInt("Ink") >= PlayerPrefs.GetInt("Price1"))
        {
            PlayerPrefs.SetInt("Ink", PlayerPrefs.GetInt("Ink") - PlayerPrefs.GetInt("Price1"));
            PlayerPrefs.SetInt("Price1", PlayerPrefs.GetInt("Price1") + 1); // 888888 888888
            Debug.Log("888888 " + PlayerPrefs.GetInt("Price1") + "888888 888888");
            PlayerPrefs.SetInt("Power", PlayerPrefs.GetInt("Power") + 1); // 888888 888888 888888
            Debug.Log("888888 " + PlayerPrefs.GetInt("Power") + "888888 888888");
        }
    }
}

```

상점의 각 항목은 유니티의 데이터 저장 방법 중 하나인 PlayerPrefs 를 사용해서 구현했다. Hash 자료구조인 PlayerPrefs으로, key값을 호출해서 set함수를 이용해 데이터를 읽고 저장해서 플레이어의 스텟을 올려주게 된다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 인게임상점



상점의 항목은 로비상점과 같이 playerprefs를 이용했고, 차이점은 적들에게 티켓 확률 변수를 넣어두고, 일정 확률로 상점에 들어갈 수 있는 티켓을 생성하게 했다. 해당 티켓을 먹으면, 인 게임 내에서만 임시로 사용할 수 있는 인 게임 상점으로 들어가서, 적들을 죽이고 모은 골드를 사용할 수 있게 하였다.

 <div> <b>국민대학교</b>  <b>소프트웨어학부</b>  <b>다학제간캡스톤디자인I</b> </div>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 피격 모션



```

void OnDamaged(Vector2 targetPos)
{
    gameObject.layer = 10;

    spriteRenderer.sprite = char_sprite[1]; // 이미지 바꿈

    spriteRenderer.color = new Color(1, 1, 1, 0.6f);

    int dirc = transform.position.x - targetPos.x > 0 ? 1 : -1;
    StartCoroutine(KnockBack(dirc));

    Invoke("OffDamaged", 0.4f);
}

void OffDamaged()
{
    gameObject.layer = 7;
    spriteRenderer.sprite = char_sprite[0];
    spriteRenderer.color = new Color(1, 1, 1, 1);
}

```

우석 데미지를 입었을 때 불리는 OnDamaged 함수와, 다시 원래 상태로 되돌아가는 OffDamaged 함수를 만들었다. 데미지를 입으면 먼저 살짝 투명해지면서, 일정 시간동안 데미지를 입지 않는 무적상태가 존재하고, 옆으로 조금 이동하게 구현을 하였다. 구현 할 때 사용한 개념은 유니티의 layer라는 개념이었다. 피격에 사용할 레이어를 하나 만든뒤, 피격 레이어는 적의 레이어와 상호작용을 꺼두고, OnDamaged 함수가 실행되면 피격 레이어로 변경해 상호작용을 막아 무적효과를 주게 하고 OffDamaged가 실행되면 다시 원래의 레이어로 돌아오게 해서 구현하였다.



 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 게임 오버



```
// 게임종료 체크 (플레이어 체력으로)
if(Player.health <= 0 )
{
    Time.timeScale = 0.0f;
   GameOver();
}
```

```
public void GameOver()
{
    dead.SetActive(true); //애니메이션 활성화
    dead.transform.position = Player.instance.transform.position; //위치잡아줌
    Player.instance.spriteRenderer.color = new Color(1, 1, 1, 0f);

    gameOverImg.SetActive(true);
    StartCoroutine(goGameOverScene());
}
```

게임매니저 스크립트에서 계속해서 플레이어의 체력을 체크하고 0 이하로 내려가면 게임내 시간을 멈추어서 정지시키고 이후에 캐릭터 죽음 애니메이션과 게임오버 글씨가 나오게 된 후 로비씬으로 이동하게 된다.



 <div> <p>국민대학교</p> <p>소프트웨어학부</p> <p>다학제간캡스톤디자인</p> </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 보스 패턴

일정 시간이 되면, 보스가 위에서 내려오게 만들었다.

보스는 여러가지 패턴으로 공격을 하는데 패턴을 한가지씩 살펴보겠다.

### 1. 원형패턴

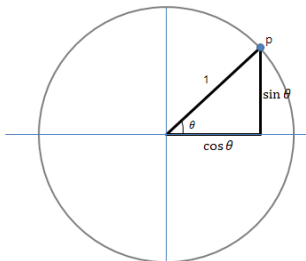


```
//원형 패턴
void FireRight()
{
    int bulletNum = 20;
    for(int index = 0; index < bulletNum; index++)
    {
        Debug.Log(index);
        GameObject bullet = objectManager.MakeObj("bulletBossSisters");
        //Debug.Log("보스탄막 생성");
        bullet.transform.position = transform.position;
        bullet.transform.rotation = Quaternion.identity;

        Rigidbody2D rigid = bullet.GetComponent<Rigidbody2D>();
        Vector2 dirVec = new Vector2(Mathf.Cos(Mathf.PI * 2 * index / bulletNum),
                                     Mathf.Sin(Mathf.PI * 2 * index / bulletNum));
        rigid.AddForce(dirVec.normalized * 10, ForceMode2D.Impulse);

        Vector3 rotVec = Vector3.forward * 360 * index / bulletNum + Vector3.forward * 90;
        bullet.transform.Rotate(rotVec);
    }
}
```

보스의 무기인 빨간 유리구두를 원형으로 발사하는 패턴이다.



한 번에 발사할 탄막의 개수인 bulletNum의 개수를 설정하고, 그 개수만큼 360도에서 나누어 한 탄막에 쓰일 각도를 생각하고, 그 각도의 sin값과 cos값을 이용해 좌표를 설정해 주었다.

이후 그 좌표 값에서 탄알이 생성되어 날라가게 된다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2. 발 패턴



위에서 발이 내려오는 패턴이다.

애니메이션을 위해 여러 장의 발의 이미지를 그리고, 각 이미지에 충돌을 처리할 콜라이더를 붙여주었다. 플레이어가 충돌할 경우, 현재 체력의 절반이 날아가게 설정하였다.

```
//발 패턴 (왼쪽)
void kickLeft()
{

    Debug.Log("왼쪽 발 패턴");
    Instantiate(boss_foot_L);

}

//발 패턴 (오른쪽)
void kickRight()
{

    Debug.Log("오른쪽 발 패턴");
    Instantiate(boss_foot_R);

}
```

만든 발 객체를 설정해둔 위치에서 왼쪽 또는 오른쪽을 설정해 생성하도록 하였다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인 </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

### 3. 손가락 패턴



```
//s자 패턴
void finger_S(int dir)
{
    float x_dir ;
    if(dir == 0){x_dir = Random.Range(0f, 1.5f); } //0이면 오른쪽
    else{ x_dir = Random.Range(-3f, -1f); }
    StartCoroutine(finger_spawn(x_dir));
}

IEnumerator finger_spawn(float x_dir)
{
    for(int i = 0; i < 10 ; i ++ )
    {
        GameObject finger = objectManager.MakeObj("bossFinger");
        finger.transform.position = new Vector3(x_dir, 2.5f, 0);
        yield return new WaitForSeconds(0.2f);
    }
}
```

발 패턴이 내려와 있으면, 플레이어의 이동이 제한되기 때문에, 이동할 수 있는 범위에서의 공격이 추가되면 더 재밌어질 것이라고 생각했고, 손가락이 나오는 패턴을 만들었다.

발이 나온 반대 방향에서, 랜덤한 위치에 손가락이 생성되고, 설정해둔 x좌표의 제한 범위만큼 밑으로 내려가면서 이동하게 하였다. 결과적으로 지그재그처럼 움직이는 손가락 패턴이 되었다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

#### 4. 보스 페이즈

```

void Think()
{
    if(health >= (maxHP/4)*3)
    {
        phase_One();
    }
    else if(health >= (maxHP/4)*2)
    {
        phase_Two();
    }
    else
    {
        phase_Three();
    }
}

void phase_One()
{
    Debug.Log("phase_One 실행 중");
    FireRight();
    Invoke("Think", 3f);
}

```

```

void phase_Three()
{
    int pattern = Random.Range(0, 5);

    switch(pattern)
    {
        case 0 :
            foot_finger();
            break;

        case 1 :
            megalodon();
            break;

        default :
            FireRight();
            break;
    }

    Invoke("Think", 3f);
}

void phase_Two()
{
    int pattern = Random.Range(0, 5);
    Debug.Log("패턴 넘버 : " + pattern);
    switch(pattern)
    {
        case 0 :
            kickLeft();
            break;

        case 1 :
            kickRight();
            break;

        default :
            FireRight();
            break;
    }

    Invoke("Think", 3f);
}

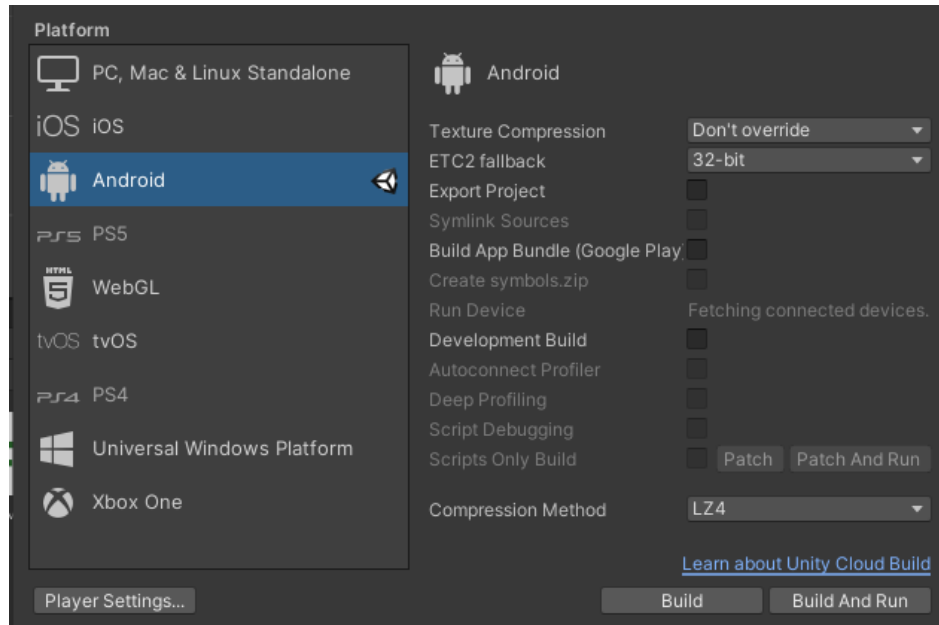
```

이렇게 만든 패턴들을 조합해서, 보스의 체력에 따라 다른 패턴이 나오게 보스 페이즈를 구현했다. 첫번째 페이즈에서는 원형 탄막 패턴, 이후에는 발과, 손가락패턴을 랜덤으로 골라서 나오게 구현했다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 모바일

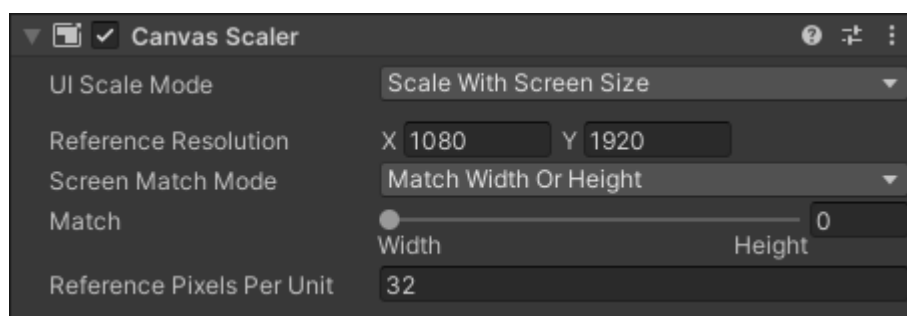
모바일 환경은 안드로이드로 설정했다.



유니티에서는 안드로이드 빌드를 제공하고 있었기 때문에 큰 어려움은 없었다.  
 신경 써야할 부분은 크게 2가지였다.

## 화면비율

각자 가지고 있는 스마트폰 마다 화면비율은 다양하고, 우리가 설정한 UI들은 컴퓨터에 맞추어져 있었기 때문에 화면비율에 맞추어 수정할 필요가 있었다.



UI 에 속하는 모든 오브젝트들의 스케일 모드를, 화면이 늘어남에 따라 늘어나는 모드로 바꿔주고, 기본설정을 9 : 16 화면 비율로 설정했다.

 <div> <b>국민대학교</b>  <b>소프트웨어학부</b>  <b>다학제간캡스톤디자인I</b> </div>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

```

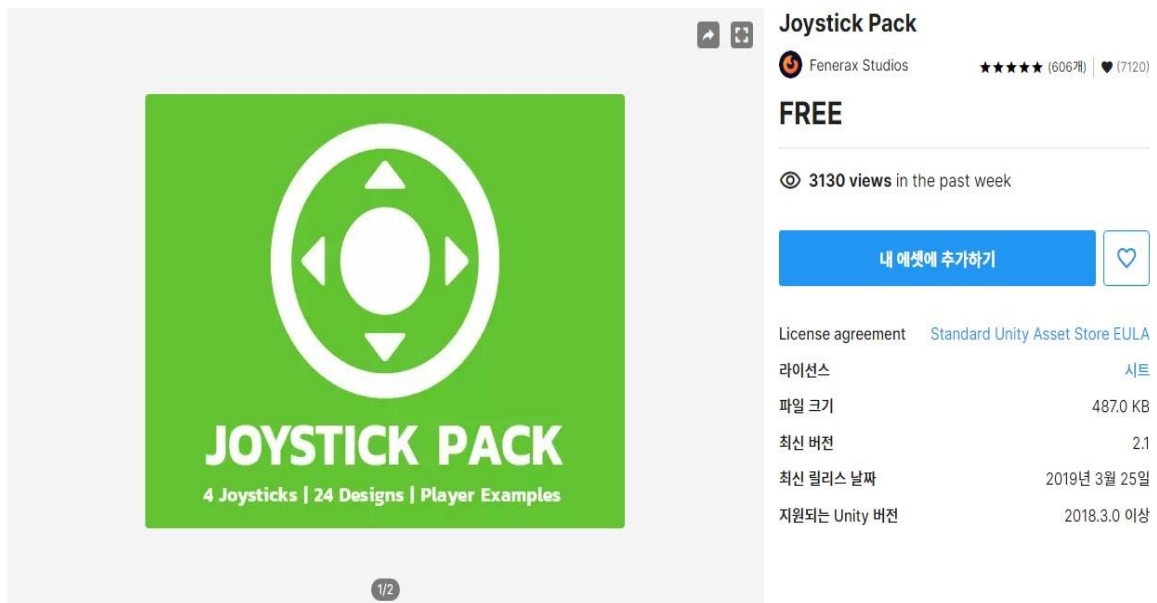
void Start()
{
    Camera camera = GetComponent<Camera>();
    Rect rect = camera.rect;
    float scaleheight = ((float)Screen.width / Screen.height) / ((float)9 / 16); // (가로 / 세로)
    float scalewidth = 1f / scaleheight;
    if (scaleheight < 1)
    {
        rect.height = scaleheight;
        rect.y = (1f - scaleheight) / 2f;
    }
    else
    {
        rect.width = scalewidth;
        rect.x = (1f - scalewidth) / 2f;
    }
    camera.rect = rect;
}

```

다음은 카메라의 비율을 설정했다. 기기의 화면 비율을 가져와서, 9:16의 화면비율로 설정해주고 남은 부분은 검은색으로 나오게 코드 설정을 해주었다.

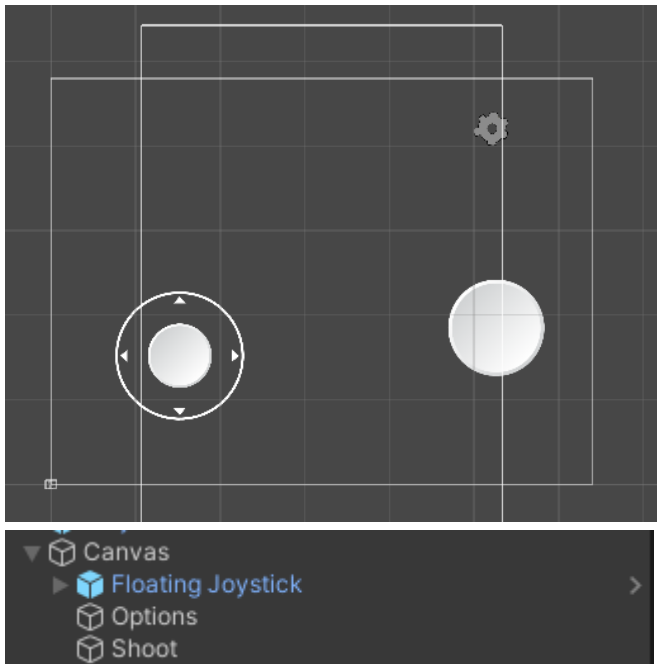
## 조이스틱

다음은 기존 키보드로 이용하던 게임을 모바일에 맞춘 터치로 변경해야 한다는 것이었다. 화면에 버튼을 놓는 방법도 있지만, 게임인만큼 조이스틱을 추가하면 좋겠다고 생각했다.



유니티 에셋중에서 무료로 배포하고 있는 조이스틱 에셋을 찾을 수 있었고, 이것을 사용하기로 하였다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

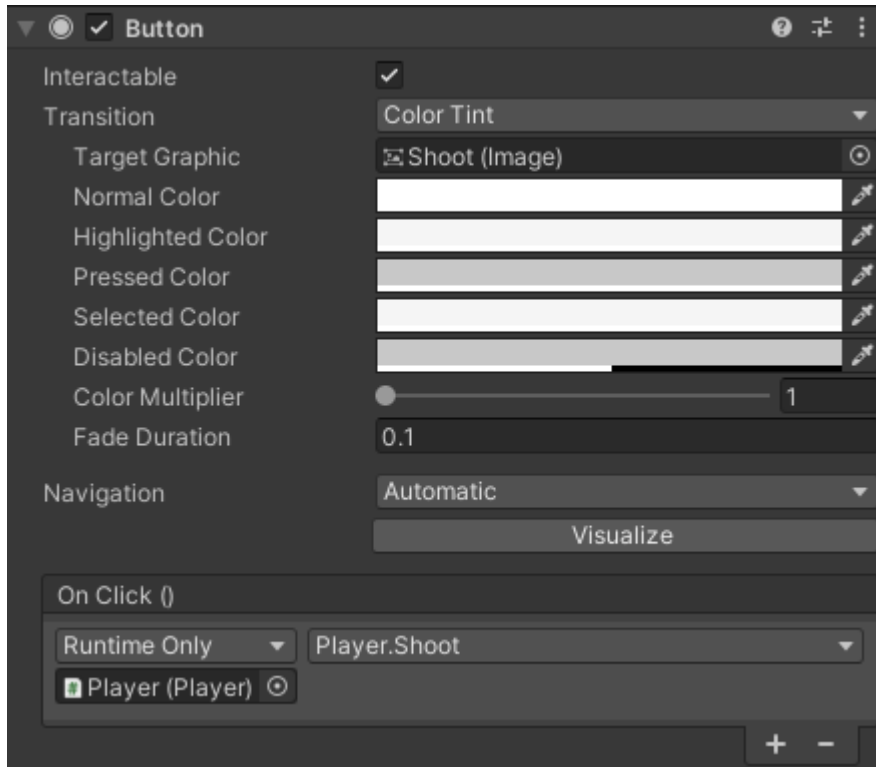


게임 화면부분에 조이스틱과 스페이스버튼을 대신한 버튼을 추가했다.

```
private void Move(){
    float x = joy.Horizontal;
    float y = joy.Vertical;
    Vector2 moveVec = new Vector2(x, y);
    Rigidbody2D rigid = GetComponent<Rigidbody2D>();
    rigid.MovePosition(rigid.position + moveVec * Speed * Time.deltaTime);
}
```

조이스틱을 움직이면, x좌표와 y좌표값이 생성이 되고, 코드부분에서 그 생성된 좌표 값의 벡터를 만들어서 움직이도록 구현했다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25



버튼부분에서는, 유니티 버튼 컴포넌트를 추가하고, 클릭되었을 때, 실행될 코드의 함수를 지정 해주었다.



 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2.2.2 시스템 기능 요구사항

### 1. 게임 타이틀

- 버튼을 누르면 게임 로비화면으로 이동 (달성)
- 종료 버튼을 누르면 게임 종료 (달성)
- 로드 버튼을 누르면 저장된 게임으로 이동 (달성)
- 배경음이 나오게 (달성)

### 2. 게임 로비

- 게임 설명과, 상점안내를 도와줄 NPC 제작 (달성)
- 상점에서 업그레이드 가능하게 (달성)
- 포탈을 통해 인 게임 안으로 이동 가능하게 (달성)
- NPC 를 통해 캐릭터를 선택할 수 있게 (미달성)

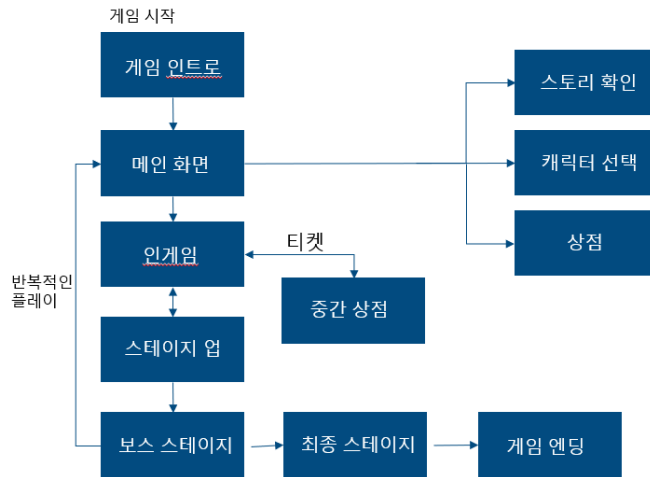
### 3. 인 게임

- 배경 위에서 아래로 움직이는 중 스크롤 배경 제작 (달성)
- 각자 다른 이동패턴을 지닌 적들이 위에서 아래로 내려옴 (달성)
- 왼쪽과 오른쪽에서도 공격하는 적 추가 (달성)
- 적을 공격해 없으면, 동전(재화)를 떨어트림 (달성)
- 일정 확률로, 인게임에서 상점을 사용할 수 있는 티켓이 나옴 (달성)
- 적 피격, 플레이어 데미지 피격 모션을 개발 (달성)
- 게임 종료시 게임 로비화면으로 이동 (달성)
- 보스는 일정 시간 이후에 출현 (달성)
- 보스는 5 가지의 공격 패턴 (달성)
- 보스의 체력이 일정 이하로 떨어지면  
다른 공격 패턴을 보여주는 '보스 페이즈' (달성)
- 보스 스테이지를 클리어하면 최종 스테이지로 이동 (미달성)
- 최종 스테이지를 클리어하면 엔딩 씬으로 이동. (미달성)

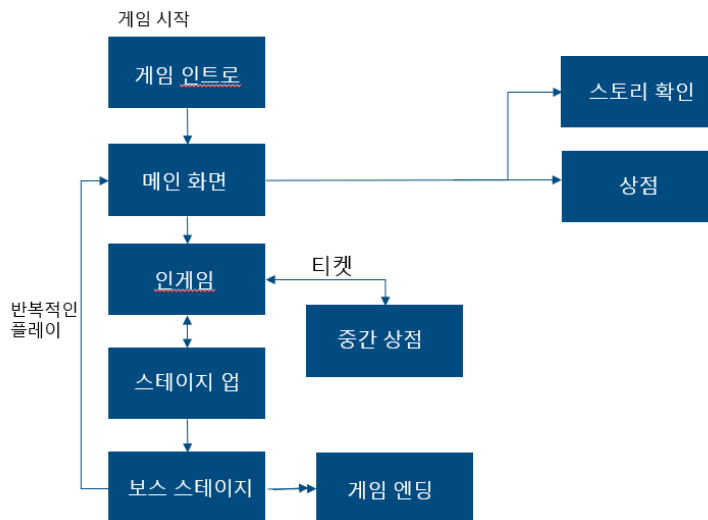
미달성 부분은 여러가지 캐릭터를 시간이 부족해 개발하지 못하고 한가지 캐릭터로 게임을 만들었기 때문에 생긴 문제이다. 로비 화면에서 NPC를 통해 캐릭터 선택은 하지 못하고 보스를 죽이면 최종 스테이지가 아닌 바로 엔딩 씬으로 이동하게 만들었다.

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

### 2.2.3 시스템 구조 및 설계도



기존의 게임 전체의 구조를 나타낸 표이다.



최종 완성된 게임의 구조이다.

원래 계획과 달라진 점은 캐릭터 선택과, 최종스테이지 부분이다. 원래 계획은 동화의 여러가지 주인공을 만들어서, 원하는 캐릭터를 선택해 캐릭터에 맞춘 적과 공격들을 하며 재미를 느끼게 하는 부분 이었는데, 개발 시간과 인력 부족으로 캐릭터를 한가지만 완성시켰다.

최종스테이지 부분도 여러 캐릭터에 맞춘 특징을 살린 스테이지를 구상했으나 캐릭터 선택 부분이 되지 않으면서 삭제되었다.

 국민대학교 소프트웨어학부 다학제간캡스톤디자인I	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2.2.4 활용/개발된 기술

### 오브젝트 풀링



탄막 슈팅 게임이다 보니, 플레이어가 쏘는 총알, 몰려오는 적들, 보스가 공격하는 총알 등등 한 화면에 수많은 객체가 생성되고 동시에 사라지는 형태의 게임 진행인데, 이 객체들을 생성하고 지우는데 유니티에서 가비지 컬렉션이 생성하고 지우고를 반복하면서 프로그램에 부하를 늘리는 문제가 존재했다. 실제로 성능이 낮은 노트북으로 게임을 실행하고, 적의 개체를 100개이상 과하게 늘린 경우에는 게임이 멈추는 현상도 존재했다.

결국 객체를 관리하는 최적화 문제를 해결할 필요가 있었고, 해결방안으로 오브젝트 풀링이라는 방법이 존재했다.



```
void Awake()
{
    enemy001 = new GameObject[20];
    enemy002 = new GameObject[10];
    enemy003 = new GameObject[10];
    bossSisters = new GameObject[1];

    itemCoin = new GameObject[10];
    itemRing = new GameObject[3];
    itemTicket = new GameObject[1];
    itemInk = new GameObject[1];

    bulletPlayer = new GameObject[20];
    bulletBossSisters = new GameObject[50];

    penRight = new GameObject[5];
    penLeft = new GameObject[5];
    bossFinger = new GameObject[15];

    Genarate();
}

void Genarate()
{
    for (int index = 0; index < enemy001.Length; index++)
    {
        enemy001[index] = Instantiate(enemy001Prefab);
        enemy001[index].SetActive(false);
    }

    for (int index = 0; index < enemy002.Length; index++)
    {
        enemy002[index] = Instantiate(enemy002Prefab);
        enemy002[index].SetActive(false);
    }

    for (int index = 0; index < enemy003.Length; index++)
    {
        enemy003[index] = Instantiate(enemy003Prefab);
        enemy003[index].SetActive(false);
    }
}
```

매번 객체를 생성하는 것이 아니라, 일정량의 객체를 미리 만들어 놓고, 이것을 가져다 사용하는 방식이다. 이렇게 하면 게임이 실행될 때 객체를 생성하는 시간만 기다리면 (로딩시간) 이후에는 별 걱정없이 가져와서 사용만 하면 된다. 사진처럼 사용할 여러가지 객체들의 배열을 생성하고, 배열길이를 할당한다. 배열안에 각 객체들을 생성하고 동시에 SetActive함수를 이용하여 비활성화한다. 이제 객체를 부를때는 SetActive(true)로 활성화해서 사용하고, 객체를 제거할 때는 false로 다시 비활성화를 해서 배열로 돌려보낸다.

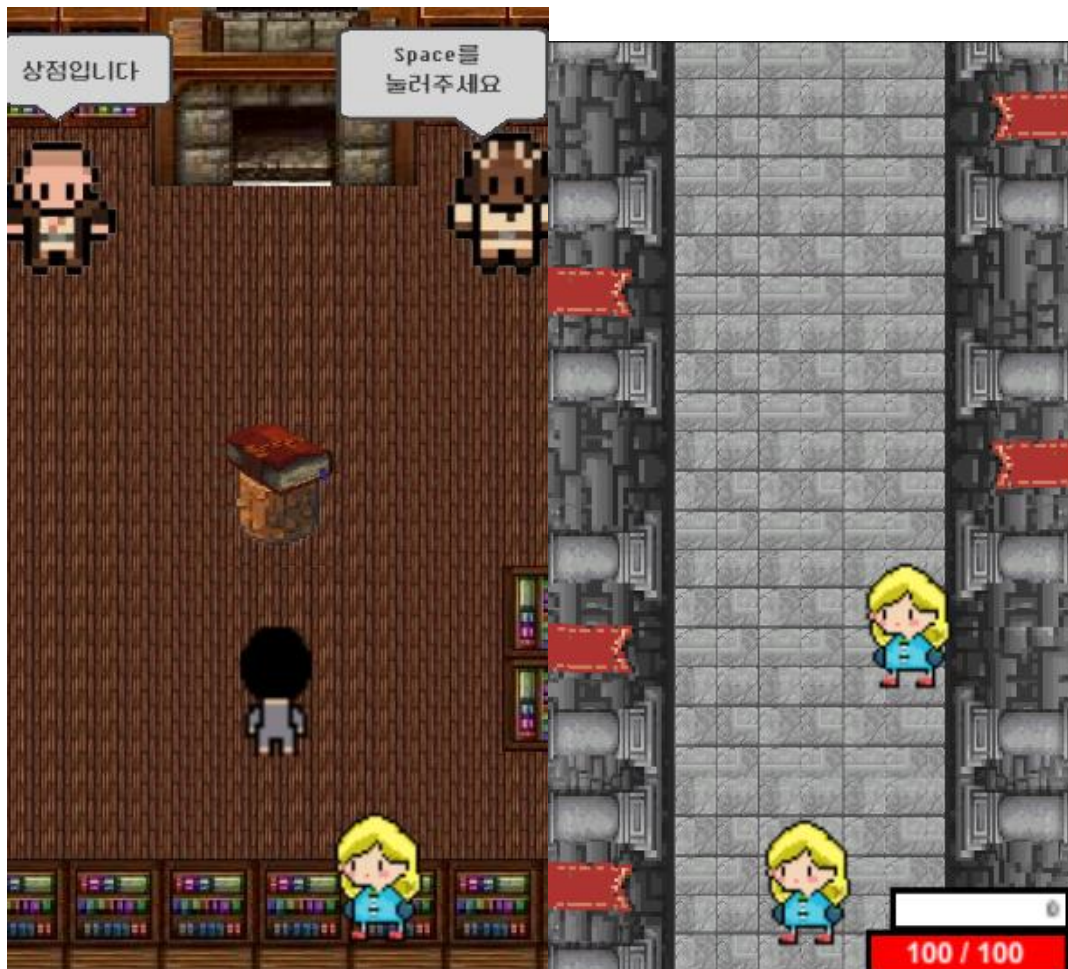
 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 싱글턴

우리 게임은 로비 씬과 인 게임씬을 계속해서 반복하면서 플레이하는 게임이다. 이렇게 씬 이동을 하다 보면, 인게임에서 플레이어 객체가 사라지는 문제가 생겼다. 이는 유니티에서 씬이 종료되면 가지고 있는 객체들을 지우고 다시 씬이 시작되면 생성하는 방식에서 생긴 문제였다.

```
private void Awake()
{
    DontDestroyOnLoad(gameObject);
}
```

그에 대한 해결법으로, 유니티에서는 DontDestroyOnLoad함수를 제공하고 있었다. 저 함수를 사용한 객체는 씬이 이동해도 사라지지 않는다.



사라지지 않는 문제는 해결되었지만, 이번에는 씬을 이동할 때 마다, 객체가 추가되어 생성되는 문제가 발생했다. 플레이어 객체는 전체 프로그램에서 한 개만 존재하여야 한다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

여기서 디자인 패턴중 하나인 싱글턴 패턴을 떠올려고 이를 적용하였다.

```

public class Player : MonoBehaviour
{
    // 싱글턴
    public static Player instance = null;

    public static float health = 100;
    public float Speed = 3f;
    public float dmg;
    public float dmg_F;
    public int maxPower;
    public int power;
    public float maxShotDelay;
    public float curShotDelay;
    public int score;
    public int ink;
    public int ring;

    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else
    {
        if (instance != this)
            Destroy(this.gameObject);
    }
}

```

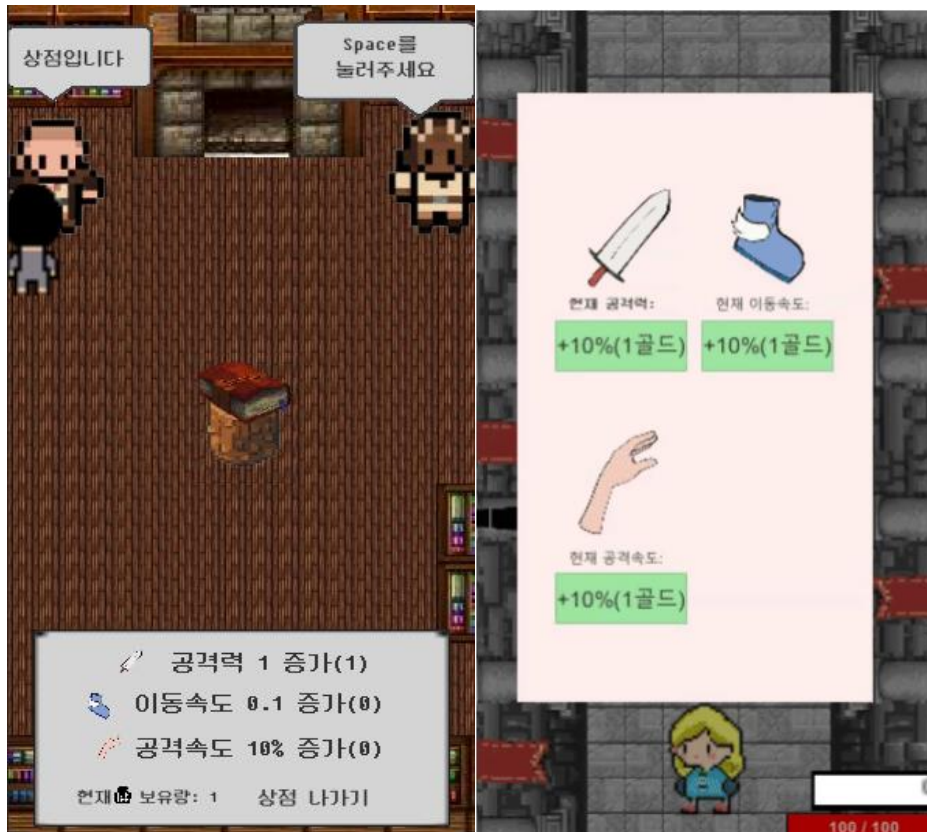
instance라는 플레이어 객체를 선언하고,

이후 인스턴스가 비어 있으면 자기 자신을 인스턴스로 설정하고, 사라지지 않게, 만약 인스턴스가 차 있다면, 플레이어 객체가 존재한다는 뜻이므로 새로 생성된 객체는 삭제하게 하여서 전체 프로그램에서 객체가 한 개만 존재하게 유지할 수 있었다.



 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 데이터 저장



데이터 저장과 사용은 위의 상점 등에서 사용이 필요한 부분이었다.

흔히 사용하는 데이터 저장방식은 DB, Json, Xml 등등 다양한데, 우리 게임에서 관리할 데이터는 플레이어 스피드, 데미지, 체력 등 몇가지 되지 않은 적은 수의 데이터였다.

그래서 유니티에서 제공하는 PlayerPrefs를 사용하기로 하였다.

 <div> 국민대학교  소프트웨어학부  다학제간캡스톤디자인I </div>	결과보고서		
	프로젝트 명	Tall Tales	
	팀 명	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## Static Functions

<a href="#">DeleteAll</a>	preference에서 모든 key와 값들을 제거합니다. 사용 시 경고가 뜹니다.
<a href="#">DeleteKey</a>	키와 대응하는 값을 삭제합니다.
<a href="#">GetFloat</a>	Preference 파일에 존재하는 /key/에 대응하는 값을 반환합니다.
<a href="#">GetInt</a>	Preference 파일에 존재하는 /key/에 대응하는 값을 반환합니다.
<a href="#">GetString</a>	Preference 파일에 존재하는 /key/에 대응하는 값을 반환합니다.
<a href="#">HasKey</a>	키가 존재하는지 확인합니다.
<a href="#">Save</a>	수정된 모든 preferences를 디스크에 씁니다.
<a href="#">SetFloat</a>	/key/로 식별된 Preference의 값을 설정합니다.
<a href="#">SetInt</a>	/key/로 식별된 Preference의 값을 설정합니다.
<a href="#">SetString</a>	/key/로 식별된 Preference의 값을 설정합니다.

Playerprefs는 다음과 같은 함수들을 가진다. 자료구조는 key와 value값으로 이루어진 Hash 형태였다.

```

public void UpgradePower()
{
    //Debug.Log("888888 88° 888888?");
    if (!PlayerPrefs.HasKey("Power")) {
        PlayerPrefs.SetInt("Power", 0);}
    if (!PlayerPrefs.HasKey("Price1"))
        PlayerPrefs.SetInt("Price1", 1);
    if (PlayerPrefs.HasKey("Ink"))
    {

        if (PlayerPrefs.GetInt("Ink") >= PlayerPrefs.GetInt("Price1"))
        {
            PlayerPrefs.SetInt("Ink", PlayerPrefs.GetInt("Ink") - PlayerPrefs.GetInt("Price1"));
            PlayerPrefs.SetInt("Price1", PlayerPrefs.GetInt("Price1") + 1); // 888888 888888
            Debug.Log("888888 " + PlayerPrefs.GetInt("Price1") + "888888 888888");
            PlayerPrefs.SetInt("Power", PlayerPrefs.GetInt("Power") + 1); // 888888 888888 888888
            Debug.Log("888888 " + PlayerPrefs.GetInt("Power") + "888888 888888");

        }

    }
}

```

위그림은 상점에 있는 업그레이드 목록중 하나인 UpgradePower 함수이다. 그림처럼 HasKey로 불러와서 Set함수를 이용해 읽고 쓰기를 할 수 있었다.



 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 2.2.5 현실적 제한 요소 및 그 해결 방안

2.2.2에서 적은 것처럼 현실적으로 개발 시간과, 인력이 부족하여서 캐릭터 선택 시스템과 스토리 확인 시스템을 개발하지 못하였다. 그로 인해 보스 스테이지 이후의 최종스테이지 또한 구현을 하지 못하여, 보스 스테이지 이후에 최종스테이지가 아닌 엔딩썸으로 이동해 게임을 끝내게 하는 방식으로 개발하였다.

## 2.2.6 결과물 목록

### 1. 게임 타이틀

- 버튼을 누르면 게임 로비화면으로 이동
- 종료 버튼을 누르면 게임 종료
- 로드 버튼을 누르면 저장된 게임으로 이동
- 배경음 나오게

### 2. 게임 로비

- 게임 설명과, 상점안내를 도와줄 NPC 제작
- 상점에서 업그레이드 가능하게
- 포탈을 통해 인게임 안으로 이동 가능하게
- NPC 를 통해 캐릭터를 선택할수 있게

### 3. 인 게임

- 배경위 위에서 아래로 움직이는 중 스크롤 배경 제장
- 각자 다른 이동패턴을 지닌 적들이 위에서 아래로 내려옴
- 왼쪽과 오른쪽에서도 공격하는 적 추가
- 적을 공격해 없으면, 동전(재화)를 떨어트림
- 일정 확률로, 인게임에서 상점을 사용할수 있는 티켓이 나옴
- 적 피격, 플레이어 데미지 피격 모션을 개발
- 게임 종료시 게임 로비화면으로 이동
- 보스는 일정 시간 이후에 출현
- 보스는 5 가지의 공격 패턴
- 보스의 체력이 일정이하로 떨어지면 다른 공격 패턴을 보여주는 '보스 페이지'
- 보스가 죽으면 엔딩썸으로 이동.
- 보스 스테이지를 클리어하면 최종 스테이지로 이동

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

### 3 자기평가

#### 최원준

우선 목표로 하던 하나의 완성된 게임의 구현에는 성공하였다.

게임 타이틀 화면부터, 로비화면, 인게임 화면, 보스스테이지의 구현과, 보스가 죽고 나오는 엔딩까지, 게임 구성면에서는 필요한 부분을 모두 갖추었다고 생각한다.

원래 기획했었던 캐릭터 선택과, 잔혹동화에 맞춘 컨셉에 관한 부분까지는 결국 만들지 못했지만, 하나의 게임의 완성이라는 목표에는 도달한 것 같아서 그점은 만족하고 있다.

사용한 기술부분에 나온 문제들은 큰 생각없이 개발을 하다가 마주한 장애물들을 해결했던 부분들에 관한 내용이 많아서 느끼는 것도 많은 부분이 있었다.

생성과 파괴를 반복하다가, 메모리 풀링이라는 좋은 방법을 알게되어 적용하고, 유니티 한 개의 씬에서 개발하면서 문제가 없다고 생각하였다가, 여러 씬을 넣어 빌드를 해보고 나니 발생한 여러가지의 문제점들을 고치면서, 계속해서 점검하면서 개발을 해야한다는 깨달음을 얻는 경험도 있었고, 싱글턴 패턴처럼, 학교 교과과목에서 배웠던 것들을 실제로 개발하면서 발생한 문제를 해결하는데 사용하는 경험도 뜻깊었다. UI 나 캐릭터 애니메이션등 도트 프로그램까지 구입해서 이미지들을 도맡아 작업을 해보았는데, 이것 역시 처음 그림을 그려보다보니 어려운 점도 많았지만, 개인적으로 잘 했다고 느낀 캐릭터 죽음 애니메이션 같은 결과물이 나오는 것을 보고 뿌듯함을 느끼기도 했다. 코드 개발 이외에도 작업을 해보면서, 실제 개발자가 된 후에도 내가 담당하는 부분 이외의 부분과의 협력하는 점에 대해서도 생각해 보게 되었다.

이런저런 좋은 경험들을 적게 되었는데, 스스로 최종 결과물의 사용 가능성을 판단해 본다면, 역시 하나의 완성된 게임이기 때문에 사용 가능성은 충분히 달성했다고 생각한다. 문제는 게임이라는 장르이기 때문에 사용 가능성외에, 게임이 주는 재미 즉 플레이어가 계속 플레이 하고싶은 게임인지에 관해서는 아직 부족하다고 생각하고 있다. 중간평가에서도 시놉시스가 창의적이라는 평가를 받았는데 그 창의적 시놉시스를 살리는 부분은 많이 부족하다고 생각한다. 여러 동화캐릭터를 넣지 못했고, 스토리 부분도 확실히 미약하다. 그렇지만 인게임 부분 난이도가 꽤 어려운 편이고, 보스패턴에서 보여주는 패턴 부분도 매력적이라고 생각하기 때문에 부족한 부분이 많지만 충분히 사용 가능한 즉 재미가 있는 게임이라고 평가하고 싶다.


 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 최민혁

나는 이번 프로젝트에서 기획, 디버깅, 개발을 맡았는데, 기획을 거의 내가 담당했기 때문에 조장이 되었지만 문서작업에는 별 재능이 없어서 다른 조원에게 개발을 하는 와중에 발표자료까지 준비하게 한 점은 고맙게 생각하고 있다. 유튜브나 구글 등지에서 문제를 해결하는 방법을 하루 종일 찾다가 결국 그 정보와 자신의 아이디어를 조합해서 문제를 해결할 때 느끼는 성취감은 이루 말할 수 없었다. 구체적인 예시를 들자면, 발표 내용에도 들어있지만 플레이어 캐릭터가 사망하고 로비로 돌아갈 때 리스폰과 합쳐져서 유령처럼 떠있고 게임 속으로 들어가도 조작이 되지 않는 문제점이 있었다. 몇 번 테스트를 돌리면서 문제점을 파악했더니 이는 DontDestroyOnLoad라는 씬을 사용하면서 Player가 그 씬에 들어가서 계속 나오고, 원래 씬에서 상호작용하던 오브젝트들이 그 Player를 인식하지 못하는 문제가 생겼었다. 그래서 구글에 해결방법을 검색해봤더니 애초에 DontDestroyOnLoad에서 생성되는 걸 전체로 한 오브젝트를 활용하는 법 밖에 나오지 않았다. 그래서 나는 혹시 씬을 옮긴다면 해결되지 않을까 싶어서 MoveGameObjectToScene 메소드를 사용하면 될까 싶어서 써봤더니 몇 번의 시행착오 끝에 의외로 간단히 해결됐다. 그리고 역할분담에 관해서 생각한 것인데 디버깅을 담당하다 보니 내가 담당하지 않은 부분의 코드도 읽고 이해해야 하는데 게임이란 매체의 특성상 스크립트가 여기저기 흩어져 있어서 생각보다 곤란했다. 게임이란 프로젝트가 작은 것도 아닌데 우리 프로젝트보다 훨씬 큰 일반적인 회사에서 만드는 프로젝트들도 이렇게 디버깅을 하려면 전반적인 진행상황에 대해 상당한 이해가 필요하거나, 프로그램의 구조를 상당히 입체적이면서 직관적이게 짜야 한다는 교훈을 얻었다.

## 4 참고 문헌

번호	종류	제목	출처	발행년도	저자	기타
1	웹 페이지	Unity Documentation	<a href="https://docs.unity3d.com/Manual/index.html">https://docs.unity3d.com/Manual/index.html</a>			

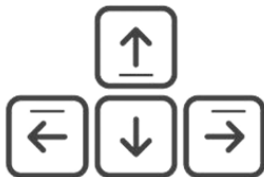
 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

2	웹 페이지	joystick pack	<a href="https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631">https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631</a>			
---	-------	---------------	---	--	--	--

## 5 부록

### 5.1 사용자 매뉴얼

pc



방향키



스페이스바

mobile



터치

### 5.2 배포 가이드

<https://kookmin-sw.github.io/capstone-2022-46/>

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

## 5.3 테스트 케이스

대분류	소분류	기능	테스트 방법	기대 결과	테스트 결과
씬	씬이동	씬이 종료되면 다른 씬으로 이동.	1) 타이틀 메뉴에서 스타트 버튼 누르기 2) 로비에서 포탈로 인게임 이동 3) 인게임에서 죽기	로비와 인게임으로 각각 이동하게 된다.	성공
NPC	설명	말풍선이 나와 대사를 말함	로비화면에서 오브젝트들에게 다가가 스페이스바를 누른다.	말풍선이 나오면서 상호작용을 하며 대사를 보여준다.	성공
	상점	업그레이드 기능	상점 NPC 에게 다가가 대화를 걸면 업그레이드 항목들을 보고 눌러서 업그레이드를 해본다.	소지한 잉크라는 재화가 줄어들면서 스텟이 업그레이드 된다.	성공
적	이동	각 적마다 다른 이동	게임이 시작되고 위에서 적이 내려온다.	각 적마다 다른 움직임으로 플레이어를 공격한다	성공
	드랍	재화 드랍	적을 공격해 죽인다.	죽으면 골드를 드랍하고 일정확률로 티켓을 드랍한다.	성공
플레이어	피격	데미지 입으면 피격모션	적에게 공격을 당한다	적에게 공격당한 반대방향으로 밀려나고 일정시간 무적	성공

 <b>국민대학교</b> <b>소프트웨어학부</b> <b>다학제간캡스톤디자인I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Tall Tales	
	<b>팀 명</b>	Choi's	
	Confidential Restricted	Version 1.2	2022-JUN-25

	게임오버	체력 0 되면 종료	적에게 공격을 당해 체력이 0 이하로 떨어진다.	죽음 애니메이션과 게임오버 글자가 나오면서 로비로 이동	성공
보스	출현	보스스테이지에 진입	일정시간을 적을 피해 버티며 생존하기	위에서 보스가 내려온다	성공
	공격패턴	공격	보스가 나타나고 기다리기	탄막, 발 등등 여러가지 패턴으로 공격	성공
	체력비례 페이즈	체력에 따른 공격패턴	보스를 공격하며 일정 체력 밑으로 떨어트린다	일정 체력 밑으로 가면 새로운 패턴공격이 나오게 된다.	성공
	죽음	죽고 게임클리어	보스체력을 0 으로 만든다.	보스가 죽고, 게임엔딩으로 이동한다.	성공