뱀파이어 서바이버 모작 기획서

게임 기획 입사 희망자

정찬일

목 차

게임 선정 이유

게임 선정 이유

게임 선정 이유

- 캐주얼, 핵 앤 슬래시 게임
- 전략에서 오는 탄탄한 게임성과 단순한 조작이 주는 엄청난 중독성
- 플랫폼을 가리지 않고 즐길 수 있는 확장성

3가지 이유와 제가 푹빠져 플레이 했었기 때문에
만들어 보기로 했습니다

게임 시스템 분석

게임 시스템 분석

게임 시스템

- 게임을 플레이 한 뒤 재화 획득 가능
- 획득한 재화로 스펙 강화 가능
- 도전 과제, 이스터에그 등으로 캐릭터 해금 가능
- 30분 간 무기, 캐릭터를 강화 해 버티는 것이 게임 목표

캐릭터

- 각 캐릭터마다 스펙과 기본 무기에 차이가 있다.
- 캐릭터는 일정 경험치를 획득하면 레벨업이 가능
- 레벨업 시 강화할 수 있는 선택지는 모두 랜덤
- 캐릭터는 적과 충돌 시에 데미지를 입으며 일정 수치 아래로 내려가면 사망과 함께 게임이 종료

게임 시스템 분석

적

- 적은 캐릭터 무기에 충돌 시에 데미지를 입으며 일정 수치 아래로 내려가면 사망과 함께 경험치 드롭
- 적은 일정 시간이 되면 스폰되며 한 게임 내 플레이 시간이 늘어날수록 체력이 증가함

기타

- 맵은 플레이어의 이동에 따라 지속적으로 확장
- UI는 레벨업과 일시정지가 있다
- 레벨업 UI는 무기 혹은 스펙 강화에 대한 것
- 일시정지 UI는 현재 내 스펙에 대한 것

게임 시스템 기획

게임 시스템 기획

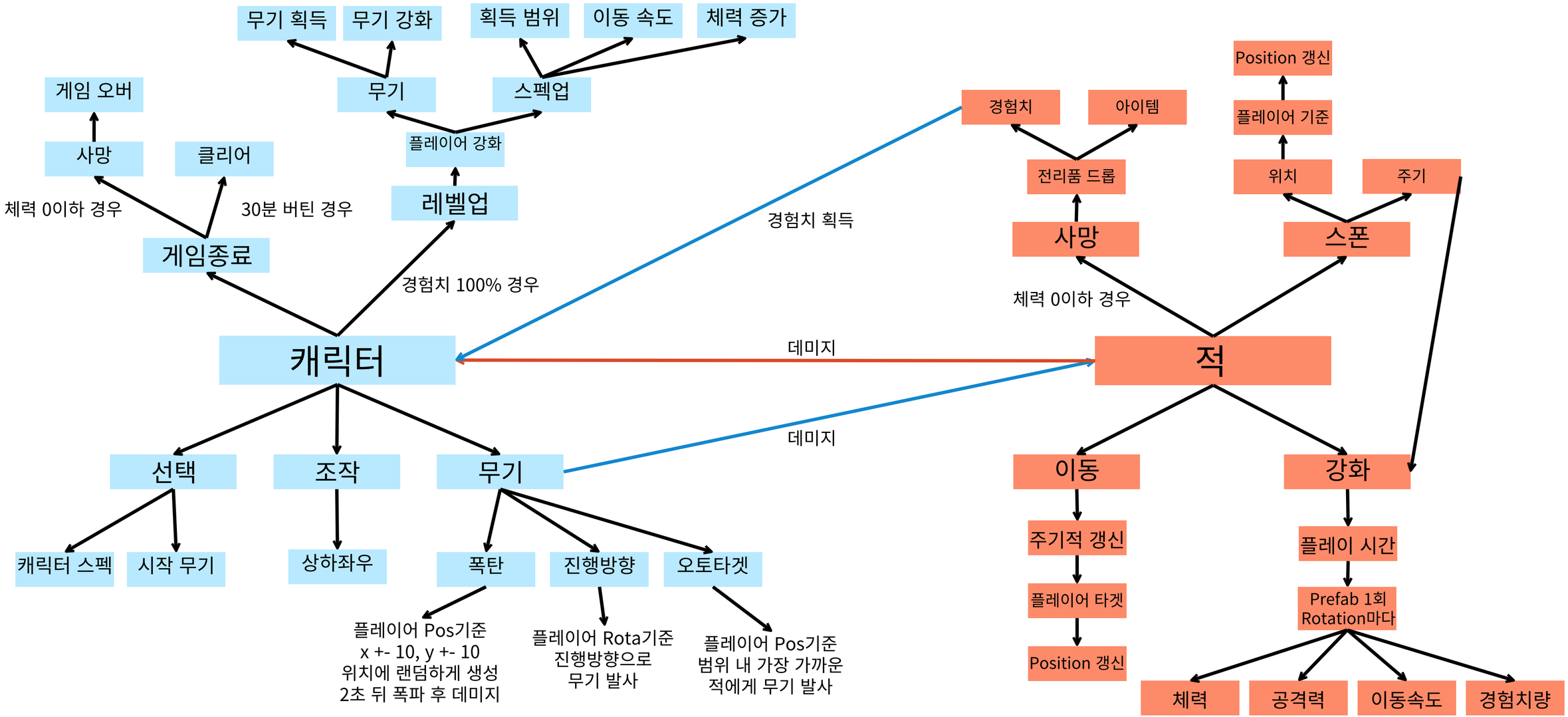
간단하고 완성도 있게

- 1인 기획이기에 모든 캐릭터, 무기 구현은 어렵다고 생각
- 최대한 뱀파이어 서바이버의 시스템과 게임과 비슷하게 기획 및 개발이 목표

구현 주요 목표

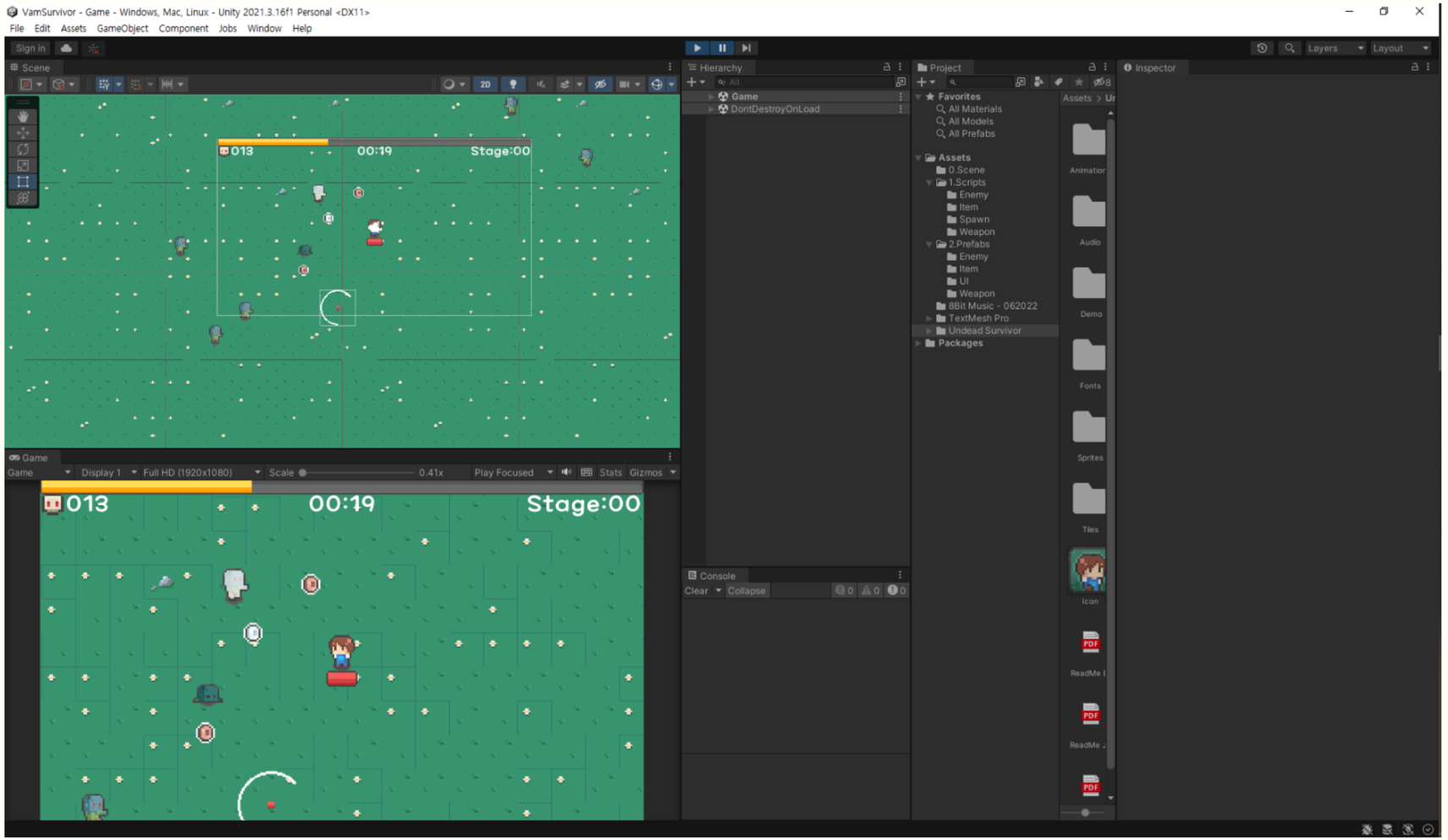
- 레벨업 시 랜덤 강화 선택지
- 각 무기의 개성
- 게임의 목표(30분 생존 시 클리어)
- 적 스폰 및 강화

게임 시스템 기획 - 흐름도


무기 획득
무기 강화
획득 범위
이동 속도
체력 증가
무기
스펙업
게임 오버
사망
클리어
플레이어 강화
레벨업
체력 0이하 경우
30분 버틴 경우
게임종료
경험치 100% 경우
경험치 획득
캐릭터
데미지
데미지
선택
조작
무기
캐릭터 스펙
시작 무기
상하좌우
폭탄
진행방향
오토타겟
플레이어 Pos기준
x +- 10, y +- 10
위치에 랜덤하게 생성
2초 뒤 폭파 후 데미지
플레이어 Rota기준
진행방향으로
무기 발사
플레이어 Pos기준
범위 내 가장 가까운
적에게 무기 발사
경험치
아이템
전리품 드롭
사망
체력 0이하 경우
Position 갱신
플레이어 기준
위치
주기
스폰
적
이동
주기적 갱신
플레이어 타겟
Position 갱신
강화
플레이 시간
Prefab 1회
Rotation마다
체력
공격력
이동속도
경험치량

게임 구현

게임 구현



[GitHub 링크]

[시연 Youtube 링크]

게임 구현

Choose 씬의 Scenechanger 스크립트와 인게임 화면

플레이어_선택

플레이어 선택은 Choose 씬에서 선택한 캐릭터의

고유번호를 받아 Game 씬의 GameController로 전달하게 했다

게임 구현

```
//플레이어 인포에서 받아 온 정보를 바탕으로 플레이어 스탯 설정, 무기 활성화
switch (PlayerInfo.instance.Act)
{
    case "Throw":
        setMaxHp = 150;
        setSpeed = 5;
        ThrowActive = true;
        break;
    case "Shoot":
        setMaxHp = 100;
        setSpeed = 3;
        ShootingActive = true;
        break;
    case "Boom":
        setMaxHp = 200;
        setSpeed = 4;
        BoomActive = true;
        break;
}
```

```
void Start()
{
    //플레이어 애니메이션 기본 설정
    sprite = GetComponent<SpriteRenderer>();
    anim = GetComponent<Animator>();
    anim.SetTrigger("Idle");

    //플레이어 스탯 설정
    maxHp = curHp = GameController.instance.setMaxHp;
    speed = GameController.instance.setSpeed;
}
```

Assembly-CSharp

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트(자산 참조 1개)|참조 5개
public class PlayerInfo : MonoBehaviour
{
    public static PlayerInfo instance;
    public string Act;
    Unity 메시지|참조 0개
    void Awake()
    {
        if (instance != null)
            return;

        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    참조 3개
    public void Data(int i)
    {
        //선택한 플레이어 정보를 게임씬에서 설정하기 위함
        switch (i)
        {
            case 0:
                Act = "Throw";
                break;
            case 1:
                Act = "Shoot";
                break;
            case 2:
                Act = "Boom";
                break;
        }
    }
}
```

Game 씬의 GameController, Player, PlayerInfo 스크립트

플레이어_선택

넘어간 번호는 PlayerInfo에서 switch~case를 통해 Act에 값이 할당되며 할당된 Act값을 바탕으로 GameController에서 캐릭터의 기본 스탯과 무기를 설정한다.

설정된 캐릭터의 스탯과 무기는 Player에서 체력, 스피드 값과 무기 활성화로 사용된다.

게임 구현

```
참조 1개
public void PlayerMove()
{
    inputVec.x = Input.GetAxisRaw("Horizontal") * Time.deltaTime * speed;
    inputVec.y = Input.GetAxisRaw("Vertical") * Time.deltaTime * speed;

    transform.position = new Vector2(transform.position.x + inputVec.x, transform.position.y + inputVec.y);

    if (inputVec.x != 0 || inputVec.y != 0)
    {
        anim.SetTrigger("Move");

        if (inputVec.x <= 0)
            sprite.flipX = true;
        else
            sprite.flipX = false;
    }
    else
    {
        anim.SetTrigger("Idle");
    }
}
```

Game 씬의 Player 스크립트

플레이어_조작

조작법은 WASD를 사용했으며 public inputVec을 선언해 값을 따로 받고 있다.

public을 선언해 따로 값을 받은 것은 무기와 관련이 있다.

게임 구현

```
Unity 메시지 | 참조 0개
void Update()
{
    if (!isLive)
        return;

    if (curHp >= maxHp)
        curHp = maxHp;

    PlayerMove();

    BulletState();
}

참조 1개
void BulletState()
{
    nearstTarget = FindNearestTarget();

    //가까운 타겟 있고, 총알 무기가 활성화 했을 때 총알 무기가 생성
    if (nearstTarget != null && GameController.instance.ShootingActive)
        ShootingBullet();

    //플레이어가 움직이며, 던지기 무기가 활성화 했을 때 던지기 무기 생성
    if ((inputVec.x != 0 || inputVec.y != 0) && GameController.instance.ThrowActive)
        ThrowBullet();

    //폭탄 무기가 활성화 했을 때 폭탄 무기 생성
    if (GameController.instance.BoomActive)
        BoomBullet();
}
```

Game 씬의 Player 스크립트

플레이어_무기

캐릭터는 기본 무기 외에 레벨업을 통해 무기가 활성화되니

조건문을 통해 무기가 생성되도록 하였다.

게임 구현

```
참조 1개
public GameObject FindNearestTarget()
{
    //enemy 태그의 모든 적을 리스트 화
    var objects = GameObject.FindGameObjectsWithTag("enemy").ToList();

    //리스트에 없으면 null 반환
    if (objects.Count == 0)
        return null;

    //플레이어와 적 사이에 거리를 기준으로 맨 처음에 있는 값을 가지고 오게 함
    var neareastObj = objects.OrderBy(obj => { return Vector3.Distance(transform.position, obj.transform.position); }).FirstOrDefault();

    //가장 가까운 적이 내 공격 범위에 없을 경우 null, 있을 경우 그 적을 타겟으로 하기 위해 값을 반환
    if (Vector2.Distance(neareastObj.transform.position, transform.position) >= 7)
        return null;
    else
        return neareastObj;
}
//총알 무기 생성
참조 1개
public void ShootingBullet()
{
    delayTimeShooting += Time.deltaTime;
    if (delayTimeShooting > GameController.instance.ShootingDelay)
    {
        GameObject weapon = GameController.instance.spawn.SpawnAct("weapon", 0);
        weapon.transform.position = transform.position;
        delayTimeShooting = 0;
    }
}
    //던지기 무기 생성
    참조 1개
    public void ThrowBullet()
    {
        delayTimeThrow += Time.deltaTime;
        if (delayTimeThrow > GameController.instance.ThrowDelay)
        {
            GameObject weapon = GameController.instance.spawn.SpawnAct("weapon", 1);
            weapon.transform.position = transform.position;
            delayTimeThrow = 0;
        }
    }
    //폭탄 무기 생성
    참조 1개
    public void BoomBullet()
    {
        delayTimeBoom += Time.deltaTime;
        if (delayTimeBoom >= 2f)
        {
            Transform weapon = GameController.instance.spawn.SpawnAct("weapon", 2).transform;
            delayTimeBoom = 0f;
        }
    }
}
```

Game 씬의 Player 스크립트

플레이어_무기

각 무기마다 독특한 특성을 부여하였다.

총알은 적을 추적해 가까운 적에게 연사하는 것으로

투척무기는 캐릭터의 이동방향으로 던지는 것으로

폭탄은 강하지만 랜덤으로 생성되며 시간이 지나야 데미지를

입히는 것으로 무기를 디자인하였다.

게임 구현

```
//플레이어 레벨업
if (playerCurEXP >= playerMaxEXP)
{
    playerCurEXP = playerCurEXP - playerMaxEXP;
    playerLevel++;
    isLevelUp = true;
    CameraObj.GetComponent<AudioController>().PlayBGM("LevelUp");
    playerMaxEXP += nextExp;
}
```

Game 씬의 GameController 스크립트

플레이어_레벨업

레벨업은 GameController에서 제어하며

요구치(playerMaxEXP) 이상의 경험치를 획득하면 실행된다.

nextExp는 고정값으로 요구치가 일정하게 증가하도록 하였고

레벨업 후 초과된 playerCurEXP는 남아 값이 유지된다.

게임 구현

```
public void GetDamage(float dmg)
{
    if (!isLive)
        return;

    curHp -= dmg;

    GameController.instance.CameraObj.GetComponent<AudioController>().PlayBGM("Hit");

    //죽었을 경우 애니메이션 및 플레이어, 적 움직임을 멈추게 하기 위함
    if (curHp <= 0)
    {
        isLive = false;
        anim.SetTrigger("Dead");
        transform.tag = "Untagged";
    }

    GameController.instance.CameraObj.GetComponent<AudioController>().PlayBGM("Base");
}
```

```
  //플레이어가 죽었을 경우(게임 오버) / 스테이지 60에 도달했을 경우(게임 클리어)
  if (!player.isLive || level >= 60)
  {
      isGameEnd = true;
      Time.timeScale = 0;
      BGMobj.SetActive(false);
  }
```

Game 씬의 Player, GameController 스크립트

플레이어_데미지, 사망

캐릭터가 적과 충돌할 경우 데미지를 받게 되며

데미지를 받아 현재체력(curHp)이 0이하가 되면 실행된다.

적의 이동은 캐릭터의 태그(player)를 쫓기에 적을 멈추기 위해

태그를 없앴고 Dead 애니메이션을 실행하게 했다.

또한 isLive 함수를 false처리하며

Player 스크립트의 Update를 멈추어 무기 생성과 키 입력을

하지 못하도록 하였다.

게임 구현

Game 씬의 에셋과 SpawnPoint 배치

적_스폰

스폰 위치는 맵에 고정되어 있으면 무한히 확장되는 맵의 특성상 적이 올바른 위치에 생성되지 못 할 것을 생각하여 플레이어 아래 SpawnPoint라는 빈 오브젝트를 넣어 플레이어가 어느 위치에 있던 플레이어 주변에서 적이 스폰되도록 설정했다.

게임 구현

Assembly-CSharp — SpawnEnemy

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트|참조 0개
public class SpawnEnemy : MonoBehaviour
{
    public GameObject[] prefabs;
    [SerializeField] List<GameObject>[] enemys;

    Unity 메시지|참조 0개
    void Start()
    {
        enemys = new List<GameObject>[prefabs.Length];

        for (int i = 0; i < enemys.Length; i++)
        {
            enemys[i] = new List<GameObject>();
        }
    }

    참조 0개
    public GameObject SpawnAct(int index)
    {
        GameObject enemy = Instantiate(prefabs[index], transform);

        return enemy;
    }
}
```

bly-CSharp — SpawnController

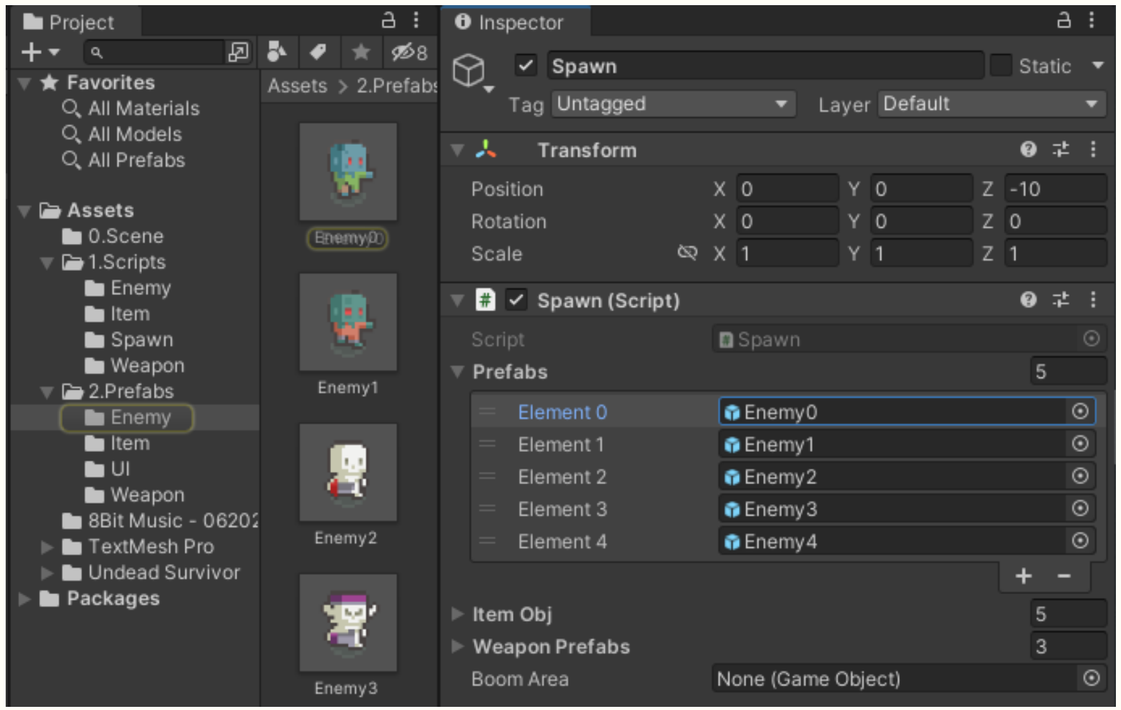
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트(자산 참조 1개)|참조 0개
public class SpawnController : MonoBehaviour
{
    //플레이어 주변에서 적이 생성 되도록 생성 위치 설정
    public Transform[] spawnPoints;

    float spawnDelayTime = 0;

    Unity 메시지|참조 0개
    void Start()
    {
        spawnPoints = GetComponentsInChildren<Transform>();
    }

    Unity 메시지|참조 0개
    void Update()
    {
        if (!GameController.instance.player.isLive)
            return;

        spawnDelayTime += Time.deltaTime;

        //스테이지 상승에 따라 적 생성 속도 증가
        if(spawnDelayTime >= 1f - (0.02 * GameController.instance.enemyUpgrade))
        {
            spawnDelayTime = 0;
            Spawn((int)GameController.instance.level % 5);
        }
    }

    참조 1개
    void Spawn(int index)
    {
        GameObject enemy = GameController.instance.spawn.SpawnAct("enemy", index);
        enemy.transform.position = spawnPoints[Random.Range(1, spawnPoints.Length)].position;
    }
}
```

Game 씬의 SpawnEnemy, SpawnController 스크립트

적_스폰

적은 저장해둔 Prefabs에 맞춰 적이 생성되며 spawnPoints에 저장된 SpawnPoint 중 랜덤한 index의 transfrom.position에 따라 생성 위치가 정해진다.

게임 구현



Game 씬의 Spawn Obj

적_스폰

Spawn스크립트에 5개의 적 Prefabs를 할당했으며

Item과 Weapon도 Spawn스크립트를 통해 생성되기에

두 종류의 Prefabs 역시 할당되어있다.

게임 구현

```
void Start()
{
    //프리펩 할당
    enemys = new List<GameObject>[prefabs.Length];

    for (int i = 0; i < enemys.Length; i++)
    {
        enemys[i] = new List<GameObject>();
    }

    items = new List<GameObject>[itemObj.Length];

    for (int i = 0; i < itemObj.Length; i++)
    {
        items[i] = new List<GameObject>();
    }

    weapons = new List<GameObject>[weaponPrefabs.Length];

    for (int i = 0; i < weaponPrefabs.Length; i++)
    {
        weapons[i] = new List<GameObject>();
    }
}
참조 5개
public GameObject SpawnAct(string type ,int index)
{
    //타입에 따라 다르게 생성

    //적은 생성 지점에서 시작
    if (type == "enemy")
    {
        GameObject enemy = Instantiate(prefabs[index], transform);

        return enemy;
    }
```

```
Enemy0
ns;
ns.Generic;

4
      Unity 스크립트(자산 참조 1개)|참조 0개
5     public class Enemy0 : Enemy
6     {
          Unity 메시지|참조 0개
7         void Start()
8         {
9             float upgrade = GameController.instance.enemyUpgrade;
10
11            ed.speed = (float)(1.5f + (0.2 * upgrade));
12            ed.damage = 1 * upgrade;
13            ed.curHp = 20f * upgrade;
14            ed.maxHp = ed.curHp;
15        }
16    }
17
```

Game 씬의 Spawn, Enemy0 스크립트

적_Prefabs

Spawn의 SpawnAct를 통해 적, 아이템, 무기가 생성되며 GameController에서 관리된다.

시간에 따라 SpawnAct가 받아오는 index값이 변하며 index값을 바탕으로 생성되는 적 외형과 스탯이 바뀐다.

또한 Enemy0 스크립트를 예시로 게임 시간에 따라 적 스탯이 강화된다.

게임 구현

```
    //타겟(플레이어) 방향으로 움직이기 위함
    transform.position = Vector2.MoveTowards(transform.position, target.position, ed.speed * Time.deltaTime);

    sprite.flipX = target.position.x < transform.position.x;
}

//플레이어 공격
Unity 메시지 | 참조 0개
private void OnCollisionEnter2D(Collision2D collision)
{
    if (!GameController.instance.player.isLive)
        return;

    if (collision.transform == target)
    {
        isAttack = true;
    }
}
```

Game 씬의 Enemy

적_이동

적의 이동은 간단하게 Update에서 플레이어의 현재 Position을 주기적으로 받아 그 위치로 적이 움직이게 하였다.

또한 충돌 시 isAttack을 false로 설정하여 충돌 1회당 데미지가 플레이어에게 들어가도록 하였다.

물론 충돌 범위를 벗어 났을 경우 isAttack이 True로 설정되며 재충돌 시 플레이어가 데미지를 입게 된다.

게임 구현

```
//적 데미지 받게 하기 위함
참조 1개
public void GetDamage(float dmg)
{
    if (!isLive)
        return;

    ed.curHp -= dmg;

    transform.Translate(Vector2.zero);
    anim.SetTrigger("Hit");

    if(ed.curHp <= 0)
    {
        Dead();
    }
}

//적 사망 시
참조 1개
public void Dead()
{
    isLive = false;
    anim.SetTrigger("Dead");
    transform.tag = "Untagged"; //플레이어가 태그를 기준으로 가장 가까운 적을 찾기 때문에 죽은 적을 untag하여 빨리 새로운 적을 찾게 하기 위함
    GetComponent<Collider2D>().isTrigger = true;
    GameController.instance.player.nearstTarget = null;
    GameController.instance.killCount++;
}
```

Game 씬의 Enemy

적_사망

플레이어와 똑같이 curHp가 0이하가 되면 사망하며

아이템을 드롭한다.

Dead 애니메이션과 함께 태그를 비활성화하는데 플레이어 총알

무기가 가장 가까운 적을 탐색할 때 죽은 적도 고르기 때문이다.

그리고 KillCount를 증가시켜 사살시킨 적 수를 증가시킨다.

게임 구현

```
//확률에 의해 exp, hp회복, 자석 아이템 드롭
참조 1개
public void DropItems()
{
    GameObject items;
    int idx;
    int rand = Random.Range(0, 100);

    //exp
    if(rand < 82)
    {
        idx = 0;
    }
    else if (rand >= 82 && rand < 91)
    {
        idx = 1;
    }
    else if (rand >= 91 && rand < 94)
    {
        idx = 2;
    }
    else
    {
        // hp, 자석
        if (rand % 2 == 0)
        {
            idx = 3;
        }
        else
        {
            idx = 4;
        }
    }

    items = GameController.instance.spawn.SpawnAct("item", idx);
    items.transform.position = gameObject.transform.position; // 적이 죽은 위치에 아이템 생성
}
```

Game 씬의 Enemy

적_아이템

적의 Dead 애니메이션 실행 2초 뒤 적Obj는 Destroy되며

확률에 따라 랜덤한 아이템이 생성된다.

GameController를 통해 SpawnAct에서 실행되며

아이템의 위치는 적의 position을 받도록 한다.

게임 구현


```
참조 1개
void BaseUI()
{
    //플레이어 Hp UI
    hpImage.fillAmount = GameController.instance.player.curHp / GameController.instance.player.maxHp;
    //플레이어 Exp UI
    expImage.fillAmount = GameController.instance.playerCurEXP / GameController.instance.playerMaxEXP;
    //게임 진행 시간 UI
    time.text = string.Format($"{min:00}:{sec:00}");
    //게임 난이도 UI
    level.text = string.Format($"Stage:{GameController.instance.level:00}");
    //죽은 적 수 UI
    killCount.text = string.Format($"{GameController.instance.killCount:000}");

}
```

Game 씬의 UIController

UI_인게임

인게임 UI는 최상단의 경험치바, 좌측 상단의 킬 수, 중앙 상단의 게임 시간, 우측 상단의 현재 스테이지(적 레벨), 캐릭터 하단의 Hp바가 있다.

경험치바와 Hp바는 fillamount를 통해, 킬 수와 게임 시간, 스테이지는 TMP를 통해 구현했다.

게임 구현

```
UI예상도 => 구현

레벨업!!
[이미지] 이동속도 증가 (현재 플레이어 스탯 => 적용 스탯)
[이미지] 0초간 적 이동속도 감소(감소 퍼센트)
[이미지] 무기 데미지 or 무기 탄환 개수 or 발사 딜레이 감소 (현재 레벨/최대 레벨)
```

메모장을 사용해 기획한 내용과 인게임에서 실제 구현 모습

UI_레벨업

레벨업UI는 GameController의 isLevelUp 값이 True일 때 실행되며 3가지 선택지는 모두 랜덤으로 생성된다.

아이콘을 선택하면 플레이어 무기, 스탯이 강화되며 UI 비활성과 isLevelUp값이 false로, curEXP와 maxEXP도 조정된다.

게임 구현

```
public void SetLevelupUI()
{
    Time.timeScale = 0;

    //중복 값이 temps에 들어가는 것 방지 위함
    List<int> temp = new List<int>();

    for (int i = 0; i < GameController.instance.imageData.ImageDatas.Length; i++)
    {
        temp.Add(i);
    }

    //temps에 넣은 temp값을 삭제하여 중복을 방지
    for (int i = temp.Count - 1; i >= 0; i--)
    {
        int rand = Random.Range(0, temp.Count);
        temps.Add(temp[rand]);

        if (temps.Count >= 3)
            break;

        temp.Remove(temp[rand]);
    }
    for (int i = 0; i < 3; i++)
    {
        LevelupUI(i, temps[i]);
    }

}
```

Game 씬의 UIController

UI_레벨업

UIController_SetLevelupUI에 사전에 만든 무기, 스탯에 대한 Prefab을 temp에 저장하였고 랜덤 값을 넣을 때 마다 해당 값의 Prefab을 삭제하여 중복값이 UI에 표시되는 것을 방지했다.

게임 구현

```
참조 1개
public void LevelupUI(int i, int index)
{
    Sprite sprite = GameController.instance.imageData.SetImageSpriteData(index);
    string str = GameController.instance.imageData.SetImageInfoText(index);

    upgradeImage[i].sprite = sprite;
    upgradeText[i].text = str;
}
//버튼 클릭하면 다시 게임이 진행되며 넣어진 데이터만큼 플레이어 스탯 상승
참조 0개
public void onButtonClick_1()
{
    Time.timeScale = 1;
    GameController.instance.UpgradePlayer(temps[0]);
    GameController.instance.levelUpUI.SetActive(false);
    GameController.instance.CameraObj.GetComponent<AudioController>().PlayBGM("Base");
    temps.Clear();
}
참조 0개
public void onButtonClick_2()
{
    Time.timeScale = 1;
    GameController.instance.UpgradePlayer(temps[1]);
    GameController.instance.levelUpUI.SetActive(false);
    GameController.instance.CameraObj.GetComponent<AudioController>().PlayBGM("Base");
    temps.Clear();
}
참조 0개
public void onButtonClick_3()
{
    Time.timeScale = 1;
    GameController.instance.UpgradePlayer(temps[2]);
    GameController.instance.levelUpUI.SetActive(false);
    GameController.instance.CameraObj.GetComponent<AudioController>().PlayBGM("Base");
    temps.Clear();
}
```

Game 씬의 UIController

UI_레벨업

랜덤으로 정해진 prefab의 정보는 버튼에 전달하고

플레이어가 버튼을 누를 경우 해당 버튼에 옮겨간 prefab의

정보를 바탕으로 GameController를 통해 Player의 스탯, 무기

를 강화하거나 무기를 활성화 할 수 있다.

게임 구현

UI 예상도

Pause
플레이어 // ooo
체력 110 / 140
속도 3ms
줍기 4m

무기1 ■■■□□□□
무기2 ■■■■■□□
무기3 □□□□□□□

플레이 타임 // 07:31
킬 // 486
----
press "활성화버튼명" continue

메모장을 사용해 기획한 내용과 인게임에서 실제 구현 모습

UI_일시정지

일시정지는 게임과 플레이어에 대한 정보가 나와야 한다 생각해 UI내에 플레이어, 무기, 게임 진행 정보를 표시하려 했다.

하지만 게임 진행에 대한 정보는 인게임에서 확인 가능하기에 기획과 다르게 구현을 하게 되었다.

게임 구현

```
public void PauseUI()
{
    SetWeaponInfoText();
    pauseInfo.text = string.Format("일시 정지");
    playerInfo.text = string.Format
        ($"플레이어 정보₩n체력 :: {GameController.instance.player.curHp}/{GameController.instance.player.maxHp}" +
        $"₩n속도 :: {GameController.instance.player.speed}m/s₩n줍기 범위 :: {GameController.instance.itemArea}m");
    weaponInfo.text = string.Format
        ($"삽 레벨 :: {weapon1}₩n" +
        $"총알 레벨 :: {weapon2}₩n" +
        $"폭탄 레벨 :: {weapon3}");
    exitInfo.text = string.Format("ESC를 눌러 계속합니다.");
}
```

Game 씬의 UIController_PauseUI 스크립트

UI_일시정지

SetWeaponInfoText를 통해 무기 활성화 여부에 따라 다른 값이 출력되도록 하였다.

모든 값은 TMP를 통해 구현하였다.

게임 구현

Assembly-CSharp | Item | Start()

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트|참조 3개
public abstract class Item : MonoBehaviour
{
    protected GameObject target;

    Unity 메시지|참조 0개
    void Start()
    {
        //플레이어를 타겟으로 잡게 하기 위함
        target = GameObject.FindGameObjectWithTag("player");
    }
    Unity 메시지|참조 0개
    void Update()
    {
        if (!GameController.instance.player.isLive)
            return;

        //플레이어와 일정한 거리 내에 있으면 자석처럼 따라게 하기 위함
        if (Vector2.Distance(target.transform.position, transform.position) <= GameController.instance.itemArea)
        {

            Vector2 vec = transform.position - target.transform.position;
            vec = vec * Time.deltaTime * GameController.instance.player.speed * 2f; //플레이어 속도의 두 배를 하여 아이템이 습득되지 않고 영영 따라다니는거 방지를 위함

            transform.Translate(new Vector2(-vec.x, -vec.y));
        }
    }
}
```

```
Unity 스크립트(자산 참조 3개)|참조 0개
public class Exp : Item
{
    public float exp;

    Unity 메시지|참조 0개
    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("player"))
        {
            GameController.instance.playerCurEXP += exp;
            Destroy(gameObject);
        }
    }
}
```

Game 씬의 Item, Exp 스크립트

기타_아이템

아이템은 플레이어의 아이템 획득 범위에 충돌할 경우

자석에 끌리듯 플레이어가 획득하게 된다.

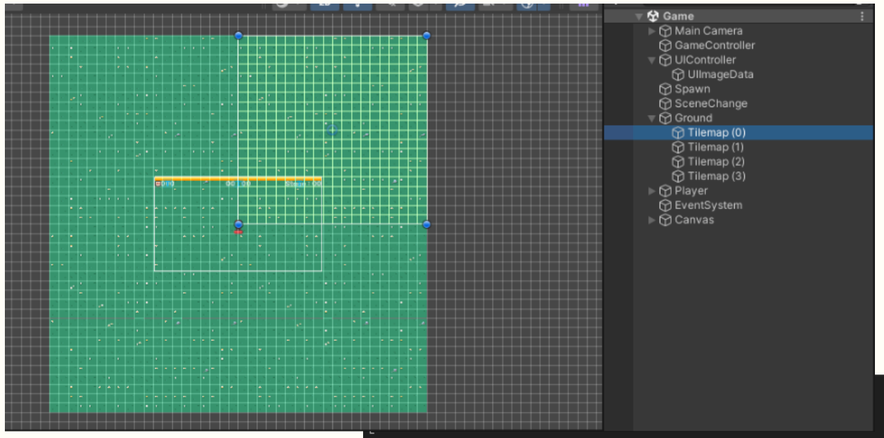
적과 유사하게 플레이어의 position을 따라가도록 하고

속도는 항상 플레이어의 2배가 되도록하였다.

Exp아이템을 예시로, 플레이어와 충돌 시 GameController의

플레이어 스탯에 변화를 일으키며 아이템 Obj는 Destroy된다.

게임 구현




```
Unity 스크립트(자산 참조 4개)|참조 0개
public class Relocation : MonoBehaviour
{
    //맵 이동을 위함
    Unity 메시지|참조 0개
    void OnTriggerExit2D(Collider2D collision)
    {
        if (!collision.CompareTag("area"))
            return;

        Vector3 playerPos = GameController.instance.player.transform.position;
        Vector3 tilemapPos = transform.position;

        float dirX = playerPos.x - tilemapPos.x;
        float dirY = playerPos.y - tilemapPos.y;

        float diffX = Mathf.Abs(dirX);
        float diffY = Mathf.Abs(dirY);

        dirX = dirX > 0 ? 1 : -1;
        dirY = dirY > 0 ? 1 : -1;

        //플레이어 위치와 방향에 따라 트리거가 끝난 바닥의 위치를 상, 하, 좌, 우로 조정하기 위함
        if (diffX > diffY)
            transform.Translate(Vector3.right * dirX * 40);
        else if (diffY > diffX)
            transform.Translate(Vector3.up * dirY * 40);
        else
        {
            transform.Translate(Vector3.right * dirX * 40);
            transform.Translate(Vector3.up * dirY * 40);
        }
    }
}
```

인게임과 Game 씬의 Relocation 스크립트

기타_맵

플레이어가 무한히 움직일 수 있는 게임 특성 상 Tile을 아무리 늘려도 끝이 있어 고민하였다.

그래서 플레이어를 중심으로 맵과 비슷한 충돌 범위를 설정하고 Tile이 충돌을 벗어나게 되면 현재 플레이어에게 입력되고 있는 값에 따라 Tile을 재배치하게 하였다.

그렇게 4개의 Tile을 사용해 무한히 확장되는 맵을 구현했다.

후기

후기

게임 기획의 어려움

뱀파이어 서바이버를 모작으로 했기에 게임을 분석하고 시스템을 기획하는 어려움이 없었지만 백지부터 기획할 때 쉽지 않겠다는 생각을 하게 되었다.

게임 개발의 복잡함

초기엔 기획 없이 개발부터 했는데, 개발을 진행할수록 가이드라인이 없어 헤매는 경우가 많았다.
개발을 진행하며 게임 기획의 중요성을 알았고 상용엔진을 직접 다루며 개발자들의 고충을 알게 되었다.

QA의 중요성

처음 기획과 달리 실제 플레이를 할 때 생각하지 못했던 버그와 개선사항이 쏟아져 나왔었다.
테스트를 하지 않았다면 완성품이 많이 부족했을 것이라 생각하며 QA의 소중함을 알게 되었다.

결론

1인으로 기획한 것을 바탕으로 Unity와 C#을 사용해 게임 개발을 하며 틈틈히 자체 QA를 통해 부족한 부분을 채우며 게임을 완성했다.
모작을 진행하며 게임 개발 전 과정의 어려움을 느꼈고 더욱 노력하여 팀원이 개발을 쉽게 할 수 있도록 좋은 가이드라인을 제시하는 기획자가 되어야겠단 생각을 했다.

감사합니다

게임 기획 입사 희망자

정찬일