KATANA ZERO

팀원 : 안승현, 김동평, 박세진

목차

- KatanaZero 소개
- 개발 목표
- 담당 포지션
- 계획
  - 안승현
    - 작업 일정
    - 1~3주차
  - 김동평
    - 작업 일정
    - 1~3주차
  - 박세진
    - 작업 일정
    - 1~3주차
- 구현에 어려웠던 점
- 해결
- 개발 후 느낀점 혹은 추후 개선 방향

KATANAZERO 소개

- 카타나 제로란?
  - 네온으로 점철된 펑키한 그래픽, 부드럽고 섬세한 도트, 강렬한 색감과 유혈을 자랑하는 2D 액션 게임으로, 매력적인 배경 설정과 그래픽, 강렬한 다크 신스웨이브 풍 사운드트랙, 화려한 게임플레이와 깊은 누아르 색의 스토리로 큰 호평을 받았다. 숙련을 요하는 난이도와 무한한 도전이 가능하다는 특징
  - 스피디한 2D 횡스크롤 방식의 게임. 체력 개념이 따로 없고 거의 모든 공격이나 함정에 한 대만 맞으면 즉사하기 때문에 빠른 선제 공격과 회피 기동을 이용해 아예 공격을 허용하지 않는 식으로 플레이
- UNITY ENGINE (2022.3.10f1) 사용
- 게임메이커 스튜디오2 전용 언팩툴인 언더테일 툴로 리소스 추출

개발 목표

- 원작의 Katana Zero의 핵심 요소들을 유사하게 구현하는 것을 목표

담당 포지션

- 안승현 - 핵심 코어 , AI 기능 구현 및 어시스트 담당
- 김동평 - 게임 매니저, 씬 구성, 옵션, UI, 비디오 기능(리플레이 기능) 담당
- 박세진 - 플레이어 기능 담당

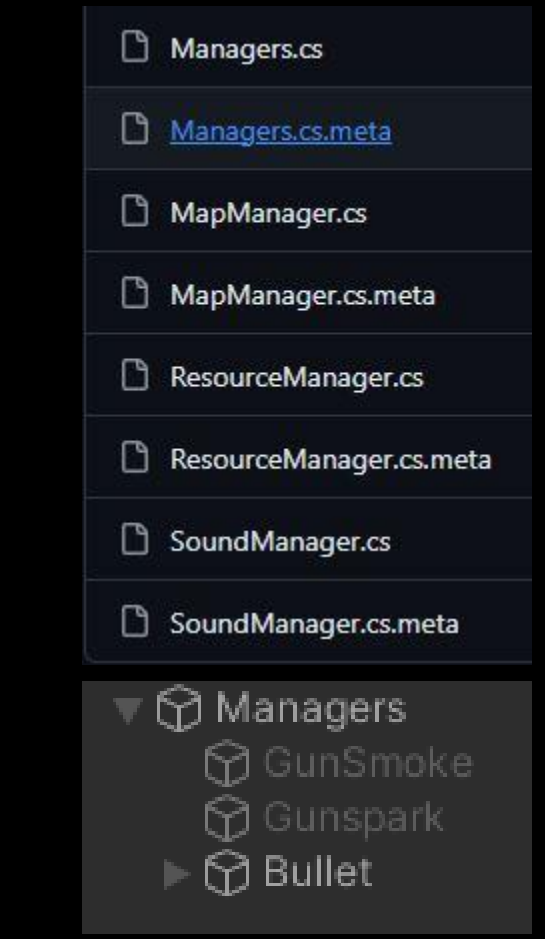
작업 일정

- 예상 작업일정
  - 1주차: 코어 및 AI 구현
  - 2주차: 원거리 몹, 보스 구현
  - 3주차: 몬스터 이펙트, 사운드 추가, 디테일 추가 및 버그 수정

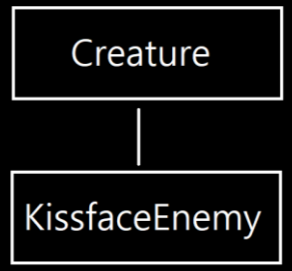
- 실제 작업 일정
  - 1주차: 코어 기능, 오브젝트 풀링, A* 구현, Enemy 클래스 구조 설계
  - 2주차: 투사체 구현, 다양한 Enemy 추가 및 애니메이션 설정, 이펙트 추가
  - 3주차: 게임 테스트, 버그 수정, 사운드 추가, 카메라 흔들림 구현, 팀 작업 어시스트, 플레이어간 상호작용 함수 구현 등

1주차 – 구조 설계(안승현)

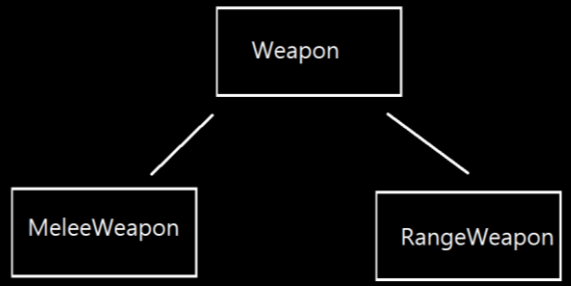
- 게임 리소스 파일 뜯기
  - 몬스터, 보스 몬스터, 이펙트, 사운드 등 담당 포지션의 리소스 추출
- 코어 기능 구현
  - Object Pooling(오브젝트 풀링), SoundManager(사운드 매니저) 등 각 종 매니저 클래스 설계
- A* 구현
- Enemy 오브젝트 애니메이션 설정
  - Idle, Walk, Run, Attack, dead 애니메이션 만들고 설정
- 적 AI 클래스 구조 설계
  - Enemy 클래스를 생성해 Enemy가 사용하는 기능들을 모듈화 하여 구현
  - 상태 머신, 무기, 상태 창 등 각종 컴포넌트 제작



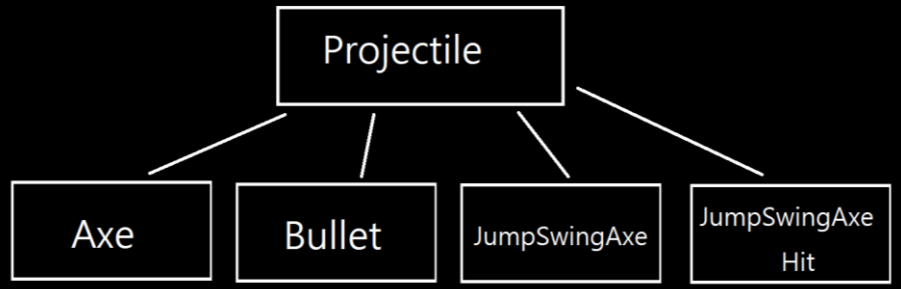
1주차 – 구조 설계(안승현)



생명체 클래스(AI)



무기 클래스



투사체 클래스

1주차 – 구조 설계(안승현)

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

public class PrioirtyQueue<T> where T : IComparable<T>
{
    List<T> _heap = new List<T>();

    public void Push(T data)
    {
        _heap.Add(data);

        int now = _heap.Count - 1;

        while(now > 0)
        {
            int next = (now - 1) / 2;
            if (_heap[now].CompareTo(_heap[next]) < 0)
                break;

            T temp = _heap[now];
            _heap[now] = _heap[next];
            _heap[next] = temp;

            now = next;
        }
    }
```

```cs
    public T Pop()
    {
        T ret = _heap[0];

        int lastIndex = _heap.Count - 1;
        _heap[0] = _heap[lastIndex];
        _heap.RemoveAt(lastIndex);
        lastIndex--;

        int now = 0;
        while(true)
        {
            int left = 2 * now + 1;
            int right = 2 * now + 2;

            int next = now;
            if(left <= lastIndex && _heap[next].CompareTo(_heap[left]) < 0)
                next = left;
            if(right <= lastIndex && _heap[next].CompareTo(_heap[right]) < 0)
                next = right;

            if (next == now)
                break;

            T temp = _heap[now];
            _heap[now] = _heap[next];
            _heap[next] = temp;

            now = next;
        }

        return ret;
    }

    public int Count { get { return _heap.Count; } }
}
```

유니티 C#에선 우선순위 큐 자료구조가 존재하지 않아 직접 우선순위 큐를 작성하여 구현

1주차 – 구조 설계(안승현)

```
public class MapManager
{
    public Grid CurrentGrid { get; private set; }

    public int MinX { get; set; }
    public int MaxX { get; set; }
    public int MinY { get; set; }
    public int MaxY { get; set; }

    public int SizeX { get { return MaxX - MinX + 1; } }
    public int SizeY { get { return MaxY - MinY + 1; } }

    bool[,] _collision;

      // 상 하 좌 우 좌상, 좌하, 우상, 우하
      int[] _nextY = new int[] { 1, -1, 0, 0, 1, -1, 1, -1};
      int[] _nextX = new int[] { 0, 0, -1, 1, -1, -1, 1, 1};
      int[] _cost = new int[] { 10, 10, 10, 10, 14, 14, 14, 14};
```

현재 씬의 그리드, 그리드의 최대 x,y 최소 x,y 값을 담은 변수
그리드의 x,y 크기
그리드 크기만큼의 bool 배열
A*시 필요한 이동, 비용 배열 선언

1주차 – 구조 설계(안승현)

```
public bool CanGo(Vector3Int cellPos)
{
    if (cellPos.x < MinX || cellPos.x > MaxX)
        return false;
    if (cellPos.y < MinY || cellPos.y > MaxY)
        return false;

    int x = cellPos.x - MinX;
    int y = MaxY - cellPos.y;
    return !_collision[y, x];
}
```

현재 위치를 매개변수로 받아와
범위 및 물체 존재 여부를
확인하여 bool 변수로 반환한다.

1주차 – 구조 설계(안승현)

```
public void LoadMap(string mapName)
{
    GameObject go = Managers.Resource.Instantiate($"Prefabs/Map/{mapName}");
    go.name = mapName;

    Transform col_transform =  go.transform.Find("Tilemap_Base");
    if (col_transform != null)
        col_transform.GetComponent<TilemapRenderer>().enabled = false;
    Transform moveable_transform = go.transform.Find("Tilemap_Moveable");
    if (moveable_transform != null)
        moveable_transform.GetComponent<TilemapRenderer>().enabled = false;

    CurrentGrid =  go.GetComponent<Grid>();

    // Collision 관련 파일
    TextAsset txt = Managers.Resource.Load<TextAsset>($"Map/{mapName}");
    StringReader reader = new StringReader(txt.text);

    MinX = int.Parse(reader.ReadLine());
    MaxX = int.Parse(reader.ReadLine());
    MinY = int.Parse(reader.ReadLine());
    MaxY = int.Parse(reader.ReadLine());

    int xCount = MaxX - MinX + 1;
    int yCount = MaxY - MinY + 1;
    _collision = new bool[yCount, xCount];

    for (int y = 0; y < yCount; y++)
    {
        string line = reader.ReadLine();
        for (int x = 0; x < xCount; x++)
        {
            _collision[y, x] = (line[x] == '1' ? true : false);
        }
    }
}
```

현재 씬의 이름으로 Grid Prefab으로 이루어진 오브젝트를 불러와 해당 맵 정보를 저장한다.

```
public List<Vector3Int> FindPath(Vector3Int startPos, Vector3Int endPos, bool ignoreDestCollision = false)
{
    bool[,] visited = new bool[SizeY, SizeX];

    int[,] open = new int[SizeY, SizeX];

    // 비용 초기화
    for (int y = 0; y < SizeY; y++)
        for (int x = 0; x < SizeX; x++)
            open[y, x] = Int32.MaxValue;

    Pos[,] parent = new Pos[SizeY, SizeX];

    PrioirtyQueue<PQNode> pq = new PrioirtyQueue<PQNode>();

    Pos pos = Cell2Pos(startPos);
    Pos dest = Cell2Pos(endPos);

    if (pos.X < 0 || pos.X >= SizeX || pos.Y < 0 || pos.Y >= SizeY)
        return new List<Vector3Int>();

    open[pos.Y, pos.X] = 10 * (Math.Abs(dest.Y - pos.Y) + Math.Abs(dest.X - pos.X));

    // F : 최종 비용 (G + H)
    // G : 시작점에서 해당 좌표까지의 비용
    // H : 목적지까지 얼마나 가까운지 비용
    pq.Push(new PQNode() { F = 10 * (Math.Abs(dest.Y - pos.Y) + Math.Abs(dest.X - pos.X)), G = 0, Y = pos.Y, X = pos.X });
    parent[pos.Y, pos.X] = new Pos(pos.Y, pos.X);

    while (pq.Count > 0)
    {
        PQNode node = pq.Pop();
        if (visited[node.Y, node.X])
            continue;

        visited[node.Y, node.X] = true;

        if (node.Y == dest.Y && node.X == node.Y)
            break;

        for (int i = 0; i < _nextY.Length; i++)
        {
            Pos next = new Pos(node.Y + _nextY[i], node.X + _nextX[i]);

            if(!ignoreDestCollision || next.Y != dest.Y || next.X != dest.X)
            {
                if (CanGo(Pos2Cell(next)) == false)
                    continue;
            }

            if (visited[next.Y, next.X])
                continue;

            // 비용 계산
            int g = node.G + _cost[i];
            int h = 10 * ((dest.Y - next.Y) * (dest.Y - next.Y) + (dest.X - next.X) * (dest.X - next.X));
            if (open[next.Y, next.X] < g + h)
                continue;

            open[next.Y, next.X] = g + h;
            pq.Push(new PQNode() { F = g + h, G = g, Y = next.Y, X = next.X });
            parent[next.Y,next.X] = new Pos(node.Y, node.X);
        }
    }

    return CalcCellPathFromParent(parent, pos, dest);
```

1주차 – 구조 설계(안승현)

현재 위치에서 목적지 까지 갈
수 있는지 A*알고리즘으로 찾아
List<Vector3Int> 형으로
반환한다.

1주차 – 구조 설계(안승현)

```
List<Vector3Int> CalcCellPathFromParent(Pos[,] parent, Pos start, Pos dest)
{
    List<Vector3Int> cells = new List<Vector3Int>();

    int y = dest.Y;
    int x = dest.X;

    if (y < 0 || y >= SizeY || x < 0 || y >= SizeX)
        return cells;

    while (parent[y, x].Y != y || parent[y, x].X != x)
    {
        cells.Add(Pos2Cell(new Pos(y, x)));
        Pos pos = parent[y, x];
        y = pos.Y;
        x = pos.X;
    }
    cells.Add(Pos2Cell(new Pos(y, x)));
    cells.Reverse();

    if (x != start.X || y != start.Y)
    {
        cells.Clear();
    }


    return cells;
}
```

FindPath 함수 에서 찾은 부모 배열의 정보를 기반으로 배열을 뒤집어서 반환한다.

1주차 – 구조 설계(안승현)

```
Pos Cell2Pos(Vector3Int cell)
{
    return new Pos(MaxY - cell.y, cell.x - MinX);
}

Vector3Int Pos2Cell(Pos pos)
{
    return new Vector3Int(pos.X + MinX, MaxY - pos.Y, 0);
}
```

Pos Cell2Pos(Vector3Int cell)
- 월드 좌표를 배열 좌표로 변환

Vector3Int Pos2Cell(Pos pos)
- 배열 좌표를 월드 좌표로 변환

2주차 – ENEMY 세부 구현(안승현)

- Bullet, Axe , ThrowAxe 등 투사체 구현
  - 각 투사체는 방향을 가지며 매 프레임마다 이동하도록 구현 생성 시 초기화 필요
- Enemy 종류 별로 구현
  - Grunt, Ganster, Pomp, Cop,Shotgun 종류 별로 아이들, 걷기, 순찰, 공격, 사망 상태 머신 구현
  - 애니메이션 구현
- BossEnemy 클래스 구현
  - 아이들, 패링, 히트,저항, 사망, 확인 사살, 다수의 공격패턴 구현
  - 애니메이션 구현
- Raycast에 사용될 레이어 설정
- Enemy 공격 이펙트, 사망 이펙트 추가
  - Effect 클래스를 통해 해당 Effect를 실행하고 일정 시간 후 비활성화 되도록 설정
  - 오브젝트 풀링이 되도록 설정해 놓아서 효율적인 리소스 활용이 가능

2주차 – ENEMY 세부 구현(안승현)

```
public class StateMachine : MonoBehaviour
{
    [SerializeField]
    Define.State eState;

    public Define.State EState {  get { return eState; } }

    [SerializeField]
    BaseState tempState;

    public BaseState TempState { get { return tempState; } }

    BaseState curState;

    public BaseState CurState {  get { return curState; } }

    Dictionary<Define.State, BaseState> stateDic = new Dictionary<Define.State, BaseState>();
```

```
public void DefaultState(Define.State eState, BaseState state)
{
    tempState = state;
    AddState(eState, state);
    ChangeState(eState, true);
}

public bool ChangeState(Define.State eState, bool ignoreCondition = false)
{
    if (stateDic.TryGetValue(eState, out BaseState state) == false)
        return false;

    if(!ignoreCondition && state.CheckCondition() == false)
        return false;

    if (curState != null)
        curState.ExitState();
    tempState = curState;
    this.eState = eState;
    curState = stateDic[eState];
    curState.EnterState();

    return true;
}
```

```
public bool AddState(Define.State eState, BaseState state)
{
    if (stateDic.ContainsKey(eState))
        return false;

    int stateIndex = (int)eState;
    if ((int)Define.State.Skill0 <= stateIndex && stateIndex < (int)Define.State.Max)
    {
        skillCount++;
    }

    stateDic.Add(eState, state);

    return true;
}

public bool RemoveState(Define.State eState)
{
    if(stateDic.ContainsKey(eState))
    {
        stateDic.Remove(eState);
        return true;
    }

    return false;
}
```

FSM (유한 상태 머신)

- 현재 상태(enum, baseState) 로 관리하며
- 이전 상태(enum, baseState) 로 관리
- 등록된 상태를 Dictionary (O(1)) 자료구조로 관리
- DefaultState : 기본 상태 등록
- ChangeState : baseState 상속받은 상태를 해당 Dictionary에 존재 여부를 파악하여 존재 시 해당 조건을 검사하여 상태 전환 후 bool형으로 결과 반환
- AddState : 중복 키 검사 후 해당 상태 값 추가
- RemoveState : 해당 키를 존재 확인 후 Dictionary에서 값 제거

2주차 – ENEMY 세부 구현(안승현)

```
public abstract class BaseState
{
    protected Creature creature;
    protected Controller controller;

    public BaseState(Creature creature, Controller controller)
    {
        this.creature = creature;
        this.controller = controller;
    }

    public virtual bool CheckCondition() { return true; }

    public abstract void EnterState();

    public abstract void ExitState();

    public abstract void FixedState();

    public abstract void UpdateState();
}
```

BaseState

- StateMahcine에서 사용하는 상태 클래스
- 모든 상태는 BaseState를 상속받아 구현한다.
- 생성자 : 해당 게임 오브젝트의 Creature클래스, Controller 클래스를 매개변수로 받아 저장한다.
- CheckCondition : 상태 전환 선행 조건을 체크하는 함수
- EnterState : 상태 전환 시 수행하는 초기화 함수
- ExitState : 해당 상태를 빠져 나갈시 수행하는 함수
- FixedState : 고정 프레임 즉 물리 연산을 수행할 함수
- UpdateState : 매 프레임 호출되는 함수

2주차 – ENEMY 세부 구현(안승현)

```
public override void InitStateMachine()
{
    AiThrowWeaponState throwWeaponState = new AiThrowWeaponState(this, GetController);
    GetStateMachine.AddState(Define.State.Skill0, throwWeaponState);

    AiJumpSwingState jumpSwingState = new AiJumpSwingState(this, GetController);
    GetStateMachine.AddState(Define.State.Skill1, jumpSwingState);

    AiAttackState atkState = new AiAttackState(this, GetController);
    GetStateMachine.AddState(Define.State.Skill2, atkState);

    AiJumpAttackState jumpAtkState = new AiJumpAttackState(this, GetController, new Vector2(2, 2));
    GetStateMachine.AddState(Define.State.Skill3, jumpAtkState);

    AiKissfaceIdleState idleState = new AiKissfaceIdleState(this, GetController, 0.0f, 1.5f);
    GetStateMachine.AddState(Define.State.Idle, idleState);

    AiKissfaceHitState hitState = new AiKissfaceHitState(this, GetController);
    GetStateMachine.AddState(Define.State.Hit, hitState);

    AiBlockState blockState = new AiBlockState(this, GetController);
    GetStateMachine.AddState(Define.State.Block, blockState);

    AiWakeUpState wakeUpState = new AiWakeUpState(this, GetController);
    GetStateMachine.AddState(Define.State.WakeUp, wakeUpState);

    AiKissfaceResistanceState KissfaceResistanceState = new AiKissfaceResistanceState(this, GetController);
    GetStateMachine.AddState(Define.State.Resistance, KissfaceResistanceState);

    AiKissfaceDeadState kissfaceDeadState = new AiKissfaceDeadState(this, GetController);
    GetStateMachine.AddState(Define.State.GroundDead, kissfaceDeadState);

    AiToBattleState toBattleState = new AiToBattleState(this, GetController);
    GetStateMachine.DefaultState(Define.State.ToBattle, toBattleState);
}
```

Kissface(보스)

- Enemy클래스를 상속받은 Kissface 클래스
- Kissface(보스)클래스의 상태머신 초기화 코드
- 오버라이드를 통해 기존의 코드를 재정의
- Skill은 kissface(보스)가 Idle상태일때 랜덤으로 선택한다.
- Idle과 Skill0~4를 제외한 상태는 조건 없이 바로 상태 전환이 가능하다.
- 대부분의 행동들은 유기적으로 연결되어 있으며 최종적으로 Idle로 돌아가도록 설계

2주차 – ENEMY 세부 구현(안승현)

```
public override void EnterState()
{
    creature.GetRigidbody.velocity = Vector3.zero;
    creature.GetAnimator.SetFloat("MoveSpeed", 0.0f);
    curTime = Random.Range(_minThinkTime, _maxThinkTime);

    nextState = creature.GetStateMachine.RandomSelectSkill(creature.GetStateMachine.UseableSkill());
}
```

```
public override void FixedState()
{
    if (controller.Target)
    {
        Vector2 dir = (controller.Target.transform.position - creature.transform.position).normalized;

        if (dir.x > 0)
            creature.GetController.LookDir = Vector2.right;
        else
            creature.GetController.LookDir = Vector2.left;
    }

    curTime -= Time.fixedDeltaTime;
    if (curTime <= 0.0f)
    {
        if (creature.GetStateMachine.ChangeState(nextState))
            return;

        nextState = creature.GetStateMachine.RandomSelectSkill(creature.GetStateMachine.UseableSkill());
    }
}
```

AiKissfaceIdleState

- AiIdleState를 상속받은 클래스
- 상태머신의 시작점이 되는 상태 (Idle)
- EnterState시 각종 값을 초기화하고 다음에 사용할 스킬을 랜덤을 통해서 Enum타입인 nextState에 저장
- FixedState에서 고정 프레임으로 플레이어를 바라보고 대기 시간이 끝나면 nextState 상태로 전환한다.
- 만약 다음 상태 변환이 실패하면 다시 랜덤을 돌린다.

2주차 – ENEMY 세부 구현(안승현)

```
public class Managers : Singleton<Managers>
{
    static MapManager map = new MapManager();
    static ResourceManager resource = new ResourceManager();
    static SoundManager sound = new SoundManager();

    public static MapManager Map { get { return map; } }
    public static ResourceManager Resource { get { return resource; } }
    public static SoundManager Sound { get { return sound; } }
```

```
public GameObject Instantiate(string key, Transform parent = null)
{
    GameObject prefab = Load<GameObject>(key);
    if(prefab == null)
    {
        Debug.LogWarning($"Failed to load Prefab {key}");
        return null;
    }

    GameObject go;
    if(prefab.TryGetComponent(out Pooling pooling))
    {
        go = Pop(prefab, parent);
        go.SetActive(true);
        return go;
    }

    go = Object.Instantiate(prefab, parent);
    go.name = prefab.name;

    return go;
}
```

```
public void Destory(GameObject go, float time = 0.0f)
{
    if (go == null)
        return;

    // 오브젝트 풀링
    if(go.TryGetComponent(out Pooling pooling))
    {
        // 게임오브젝트 비활성화
        go.SetActive(false);
        Push(go);
        return;
    }

    Object.Destroy(go, time);
}
```

Effect 오브젝트 생성

- 1주차에 구현한 ResourceManager를 이용해 게임 오브젝트를 불러온다. 만약 Polling 컴포넌트를 가지고 있으면 해당 오브젝트를 생성 및 또는 Pop한다.
- 사용이 끝난 오브젝트는 ResourceManager의 Destory함수를 이용해 해당 오브젝트를 제거하거나 만약 Pooling 컴포넌트가 있다면 비활성화 한 후 Push한다.

3주차 – 사운드 추가 및 기타 작업(안승현)

- Enemy 사운드 추가
- 보스 전투 준비 애니메이션 추가
- 카메라 흔들림 구현
  - 보스 대시 공격할 시 카메라가 흔들림 구현
- 플레이어와 상호작용 함수 구현
  - Ihitable 인터페이스로 공격 판정 통일
  - 총알 패링 함수를 만들어 플레이어 쪽에서 함수 호출로 간편히 패링 할 수 있도록 구현
- 팀원 작업 어시스트
  - 버그 수정, 게임 테스트, 리플레이 기능 등

3주차 – 사운드 추가 및 기타 작업(안승현)

```
public class Managers : Singleton<Managers>
{
    static MapManager map = new MapManager();
    static ResourceManager resource = new ResourceManager();
    static SoundManager sound = new SoundManager();

    public static MapManager Map { get { return map; } }
    public static ResourceManager Resource { get { return resource; } }
    public static SoundManager Sound { get { return sound; } }
```

```
public void Play2D(string path)
{
    AudioClip audioClip = Managers.Resource.Load<AudioClip>(path);
    if (isInit == false)
    {
        Init();
    }

    if (audioClip == null)
        return;

    AudioSource audioSource = _audioSources2D;
    audioSource.PlayOneShot(audioClip);
}
```

SoundManager

- 1주차때 만들어둔 싱글톤인 Manager의 static 변수인 sound 객체를 이용해서 사운드 호출이 필요한 스크립트에서 간단하게 사운드 경로를 지정해서 호출해주면 된다.
- 싱글톤으로 구현해서 접근이 편리하다.
- AudioSource 클래스의 PlayOneShot 함수는 하나의 AudioSource로도 여러 사운드를 출력할 수 있는 함수다. 사운드 출력 후 에는 사용자가 직접 해당 사운드에 대해 수정을 가할 수 없는 단점이 존재한다.

3주차 – 사운드 추가 및 기타 작업(안승현)

```
public interface IHitable
{
    public void OnHit(GameObject owner, bool isGround = true);

    public void OnHit(GameObject owner, float damage, bool isGround = true);
}
```

IHitable 인터페이스

- IHitable 인터페이스를 정의함으로써 이 기능을 필요로 할때 상속받아서 구현함으로써 좀더 유연하게 코드 작성이 가능하다.
- 협업을 할 때에는 다른 개발자가 이 인터페이스를 상속 받고 구현 그리고 호출 할때는 이 인터페이스를 상속 받았는지만 확인하면 되기 때문에 각자의 코드에 집중할 수 있는 장점이 있습니다.

작업 일정(김동평)

- 예상 작업일정
  - 1주차: 타일맵 정리, 소스정리, 맵 배치
  - 2주차: 맵(3개)제작, 오브젝트구분(바닦,벽 등), 플랫폼 세팅, 문 등 오브젝트 설치
  - 3주차: 타이틀, 깨지는 타일맵, 깨지는 오브젝트 구현 및 슬로우 및 비디오 구현, 버그 픽스 및 bgm과 배경효과 추가

- 실제 작업 일정
  - 1주차: 타일맵 정리, 소스정리, 맵배치
  - 2주차: 맵(3개)제작, 오브젝트구분(바닦,벽 등), 플랫폼 세팅, 문 등 오브젝트 설치
  - 3주차: 펜(Fan)추가, 보스 룸, 슬로우 구현, 몬스터 배치, 타이틀, 옵션, 게임매니저, 버그 픽스 및 bgm과 배경효과, 비디오효과 ,엔딩구현

1주차 – 구조 설계(김동평)

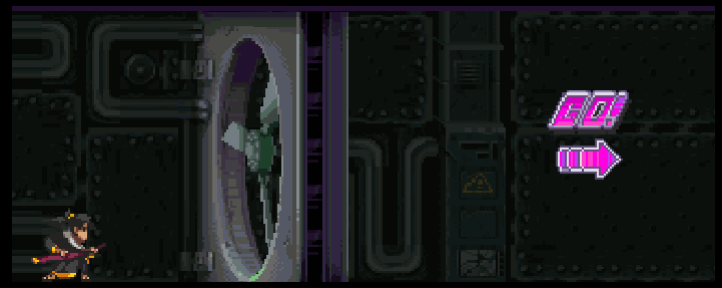
- 게임 리소스 파일 뜯기
  - 맵, 오브젝트, UI 이미지 파일 및 효과, 배경음 추출
- 맵 구현
  - 기본적인 전체 씬(타이틀,Factory0,Factory1,Factory2,BossRoom,Endding 총 6 씬) 구현
- 상호작용 가능한 오브젝트 추가
  - 바닥 콜라이더, 벽 콜라이더, 문 상호작용, 깨지는 타일 상호작용
- 깨지는 타일
  - 콜라이더 충돌을 통해 리지드바디 적용된 타일들이 분리 및 Color.a값 수정으로 통로 밝아지게 되도록 수정

2주차 – ENEMY 세부 구현(김동평)

- 맵 제작, 오브젝트구분(바닥, 벽 등), 플랫폼 세팅, 문 등 오브젝트 설치
- 타이틀, 옵션 구현
  - 키보드 선택창 : 이넘값을 통한 방향키이동
  - 사운드 옵션 : 키보드 입력으로 슬라이더값을 크게 작게 할 수 있도록 함로고
  - 애니메이션: 애니메이션 이벤트를 통한 사운드 및 애니메이션효과
- 씬에 몬스터 배치
  - Enemy Prefab 배치
- 플레이어 기믹 슬로우 구현
  - 슬로우 게이지가 있으면 timeScale을 통한 슬로우.
  - 사용에는 unscaledDeltaTime, 충전에 deltaTime사용
  - 슬로우 시 FadeOut 특정오브젝트(총알,도끼,펜) 오더 인 레이어를 통해 밝게 표시

3주차 – 사운드 추가 및 기타 작업(김동평)

- 게임매니저, 버그 수정 및 배경음과 배경효과, 비디오효과, 엔딩 구현
- 씬 클리어 조건 추가
  - 씬 에서 Enemy의 수를 받고 Enemy 사망 판정수를 매치해서 같으면 다음 스테이지로 넘어가는 트리거 작동
- Factory2의 팬 오브젝트 추가와 상호작용 요소 구현
  - 팬 날에 충돌 시 플레이어가 뒤로 밀려나간다.
- 리플레이 기능 추가
  - 플레이 사망이 또는 스테이지 클리어시 Recorder클래스를 통해 매 프레임에 만들어진 Texture2D를 List로 저장하고 리스트를 불러와 Sprite로 만들어 화면에 띄움
  - 사망 시 : index끝에서 돌려 되감기하고 씬 재시작
  - 클리어시 : 씬 처음부터 클리어까지 저장한 List를 인덱스를 통해 재생 -재생,정지
  - 되감기 : 이넘값을 만들어 index로 구현



작업 일정(박세진)

- 예상 작업일정
  - 1주차: 플레이어 이동,벽점프,기본공격,구르기 기능 및 애니메이션 구현
  - 2주차: 오브젝트 사용 원거리공격,총알 패링 기능 구현
  - 3주차: 플레이어 움직임 최적화, 사운드 및 이펙트,버그수정 및 스테이지 추가

- 실제 작업 일정
  - 1주차: 플레이어 이동,벽점프,기본공격,구르기 애니메이션 구현
  - 2주차: 마우스방향에 따른 공격및 공격판정, 플레이어 움직임 최적화
  - 3주차: 벽점프,패리 기능 구현, 플레이어 움직임 최적화,사운드,이펙트 추가, 버그수정

1주차 – 구조 설계(박세진)

- 플레이어 이동
  - 플레이어 좌우 움직임은 GetAxisRaw이용하여 받아온 값으로 Vector에 입력하여 플레이어의 속도를 곱해 구현
  - 플레이어의 방향과 입력값이 반대일경우 eulerAngles.y-180의 평균값으로 수정하여 방향을 조정
  - 플레이어 점프는 rigid.AddForce이용하여 구현했고 jumpCount변수를 이용하여 공중점프를 제한
  - 하향 점프는 CollisionEnter를 통해 플레이어가 서있는 바닥이 특정한 태그라면 IgnoreCollision를 통해 그 바닥과 플레이어의 충돌을 짧은 시간 동안 무시하도록 구현
  - 착지 판정은 정확도를 위해 플레이어의 앞과 뒤에 각각 RaycastHit2D를 만들어 둘 중 하나라도 null이 아니고, 거리가 지정값 이하라면 플레이어가 지면에 착지했음을 판단
- 플레이어 애니메이션 구현

2주차 – ENEMY 세부 구현(박세진)

- 플레이어 공격
  - 마우스 포인터의 위치값과 플레이어의 위치값으로 Mathf.Atan2,Rad2Deg를 이용해 각도를 나타내는 변수를 새로 만들어서 적용해 플레이어를 기준으로 마우스방향으로 공격이 가능하도록 구현
  - 공격 판정은 이펙트 크기에 맞는 사각형 투명한 오브젝트를 만들어 TriggerEnter를 통해 지정된 태그를 가진 게임오브젝트들과 상호작용하도록 구현 공격판정 오브젝트는 Instantiate를 통해 생성하고 짧은시간후 Detroy를 통해 파괴
  - 공격 판정에서 레이어를 이용해 총알 오브젝트를 받아와 충돌시 Bullet 클래스의 ReflectDir 함수를 호출해 총알을 팅겨낸다.
- 플레이어 구르기
  - 구르는 동안은 지정된 시간동안 플레이어의 레이어를 변경하여 피격판정을 무시하도록 구현

3주차 – 사운드 추가 및 기타 작업(박세진)

- 플레이어 벽타기 구현
  - 플레이어가 공중에서 벽에 닿았을때 다른 판정들과 겹치지 않도록 Raycast를 이용하여 구현
  - 플레이어 벽체크가 활성화 되었을때 지정한 속도로 벽에서 미끄러져 내려오고 키입력을 받았을때 AddForce를 이용하여 정해진 방향으로 점프하도록 구현 또한 연속으로 작동하여 벽에 부딪히지않도록 Invoke함수 활용하여 짧은시간 동안 조작이 불가능하게 방지
- 사운드 구현
- 이펙트 구현
  - 이펙트 착지,점프 등의 먼지 이펙트는 RaycastHit2D의 point를 통해 위치를 지정한 후 Instantiate를 통해 생성하고 짧은 시간 후 Destroy를 통해 파괴됩니다. 달리기 이펙트의 경우에는 플레이어 방향에 따라 회전하도록 구현했습니다.
- 플레이어 피격
  - 플레이어가 적에게 공격받았을 때 보고 있는 방향의 반대방향으로 날아가며 피격 애니메이션이 재생되도록 했고,bool값을 주어서 이동관련 입력을 막고, 레이어를 수정하여 더 이상 피격판정이 이루어지지 않도록 방지

구현에 어려웠던 점

- 구현에 어려움보단 팀 포트폴리오이기에 협동 작업 역할 분담 및 각자 만든 작업물의 상호작용, 또는 깃 허브 연동하는게 어려웠다. 왜냐하면 씬, 태그, 레이어, 같은 게임 오브젝트를 수정하면 그 값이 덮혀 버리기 때문에 소통이 되지 않으면 그 날 작업한 내용이 날아가서 난감했다.

해결

- 씬은 씬 담당자만이 작업을 하도록하고 만약 변경사양이 있으면 담당자에게 전달사항을 전달해 수정하도록 하였다.
- 각자 만든 작업물은 조장인 내가 인터페이스 클래스를 작성하여 각자의 코드에서 호출 하여 상호작용 하도록 작성하였다. Ex) 피격 인터페이스, 총알 반사 함수

개발 후 느낀점 혹은 추후 개선 방향

- 안승현 – 처음 해본 팀플과 GitHub 문제로 기간내에 해결할 수 있을지 걱정이였지만 팀원분들이 자기가 맡은 역할을 잘 수행해주어서 좋았고 아쉬운점은 팀원간 의사소통을 적극적으로 하지 않아서 아쉽습니다. 다음 기회에는 적극적인 소통을 하도록 노력하겠습니다.
- 김동평 –게임 매니저를 만드는 과정에서 유기적인 상호작용을 하는데에 있어 코드가 정리가 더 필요하다 느꼈습니다. 미리 전체적인 틀을 잡아두고 했다면 좀더 효율적인 구성과 정리가 잘 되었지 않았을까하는 아쉬움이 있습니다. 그리고 제가 맡은 파트가 게임매니저 및 게임플레이 기믹인데, 분업 작업해서 하나의 게임으로 합치고 완성 했다는것이 좋은 경험이었습니다. 전체적인 기간의 플랜이 예상한것과 거의 맞게 완성 되었다는것 또한 굉장히 좋았습니다.
- 박세진 – 팀 작업에서 메인파트라고 할 수 있는 플레이어 파트를 맡게 되어서 걱정이 많았지만 핵심 기능들은 대부분 구현해서 다행이라고 생각합니다. 하지만 작업하는 과정에서 다른 팀원들의 작업내용과 합치게 되었을 때 플레이어의 수정해야할 부분이 끝없이 생겨 추가하고 싶었던 내용들을 작업하지 못해 아쉬움이 있습니다. 추후 이 게임을 더 작업하게 된다면 일단 플레이어의 현재 상태를 좀 더 명확하게 설계해서 애니메이션을 더 부드럽게 수정하고 싶고 스테이지의 물건들을 이용하는 원거리 공격도 추가로 작업하고 싶습니다.