문제해결능력응용

Penguin run

202127002 전가은

GIT주소 : https://github.com/JGE77531/JGEUnity

Commit SHA-1 : e77afa23a88d3088d5283a7222ac76a422319b02

목차

01. 개요

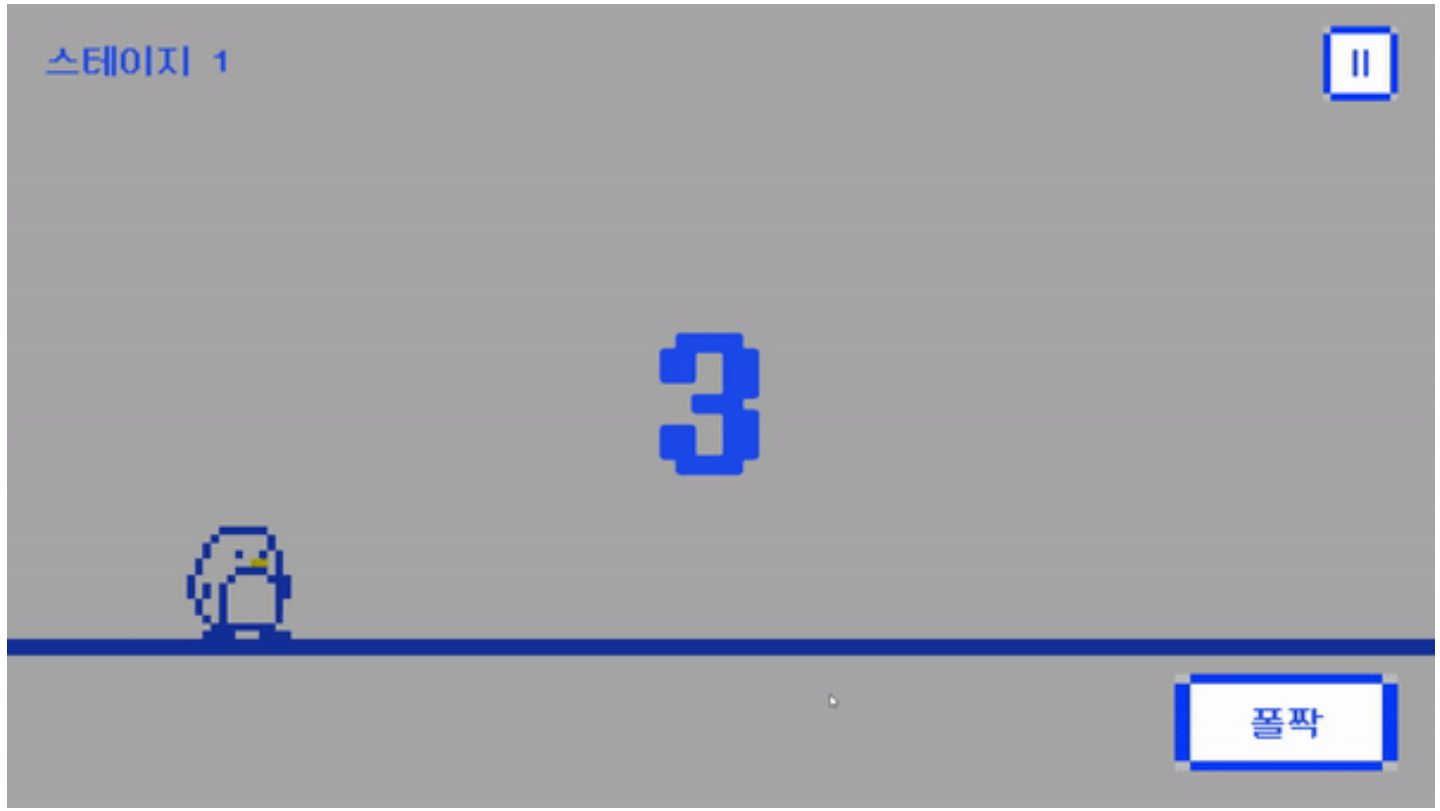
게임 요약

[타이틀 화면] 유저의 이름 입력 및 시작

01. 개요

게임 요약

[메인 화면] 시작 전 3부터 카운트 다운 진행



01. 개요

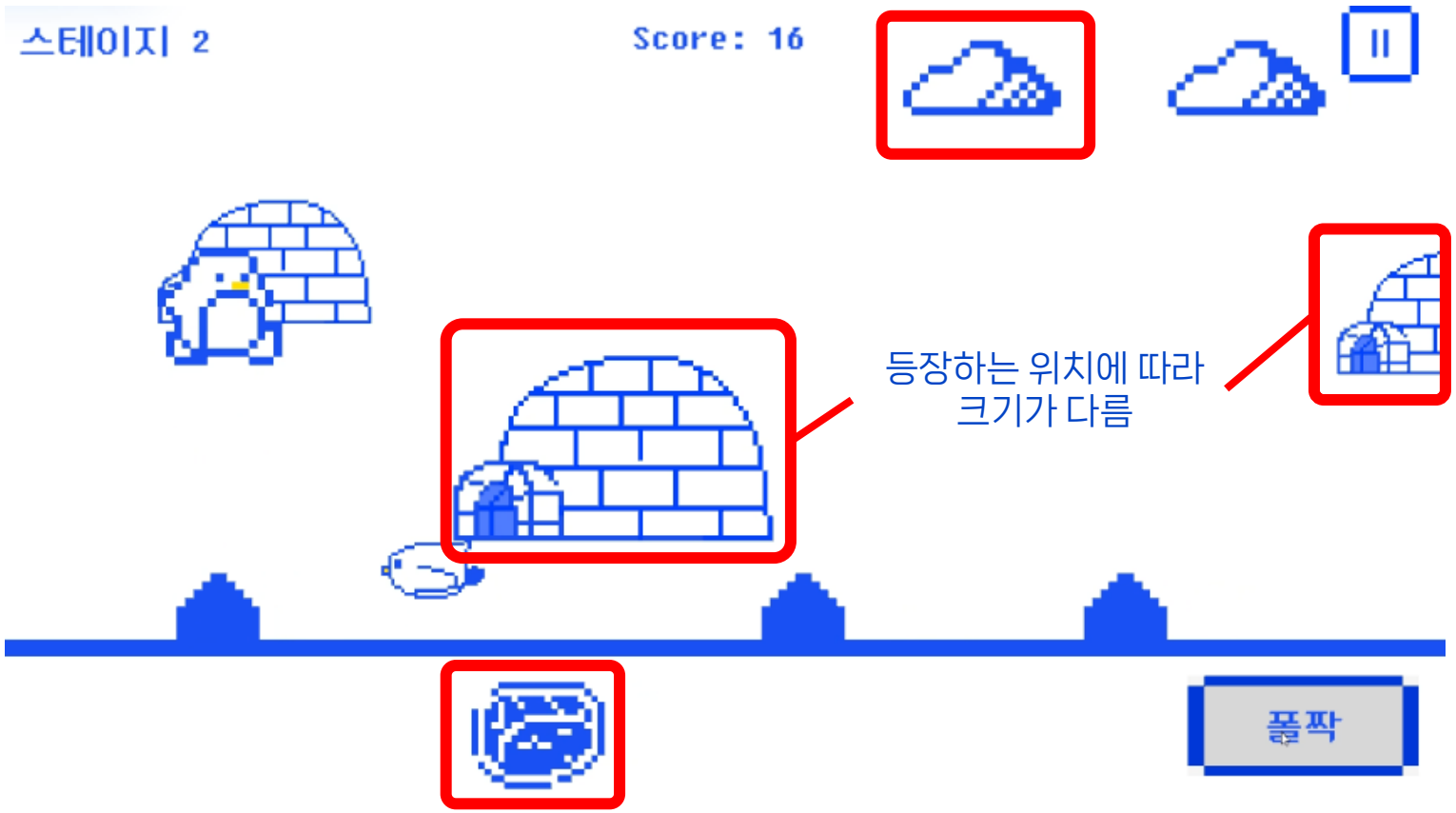
게임 요약

[메인 화면] 플레이어 : 점프는 2단까지 가능

01. 개요

게임 요약

[메인 화면] 배경 오브젝트 : 총 3개 <물웅덩이> <이글루> <구름>



01. 개요

게임 요약

[메인 화면] 장애물 2개 <돌> <펭귄>

01. 개요

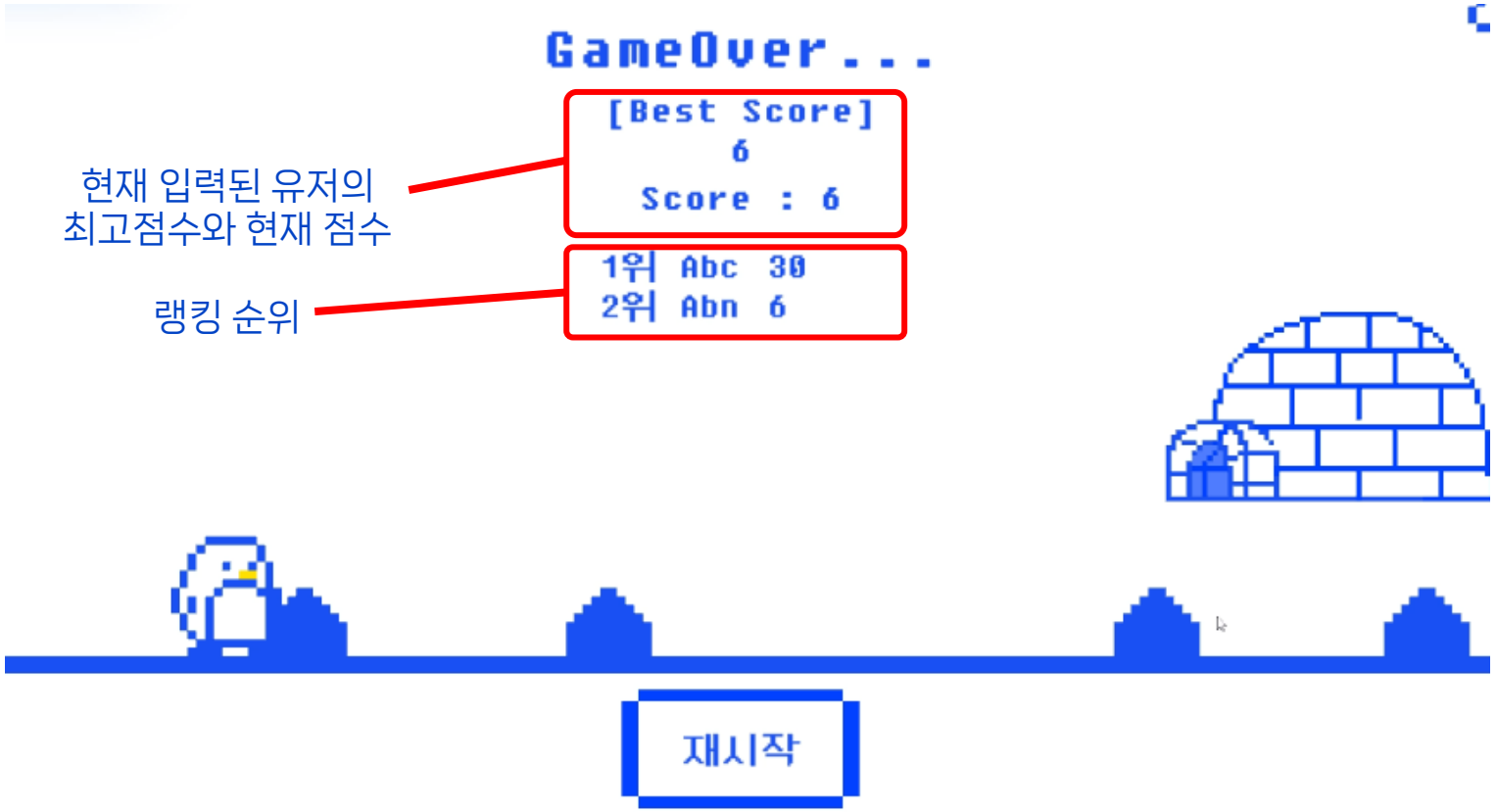
게임 요약

[메인 화면] 스테이지 : 게임 플레이 시간이 15를 넘길 때마다 1씩 증가하며, 게임 속도가 빨라짐

01. 개요

게임 요약

[종료 화면] 유저의 최고 점수와 현재 점수, 랭킹을 10위까지 출력



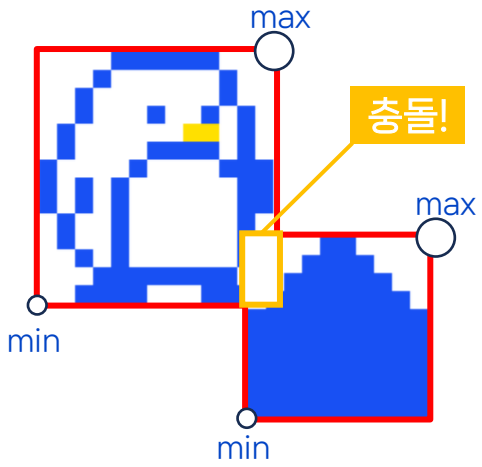
현재 입력된 유저의
최고점수와 현재 점수

랭킹 순위

01. 개요

게임 엔진 설계의 주요점

1. 충돌 처리 : AABB



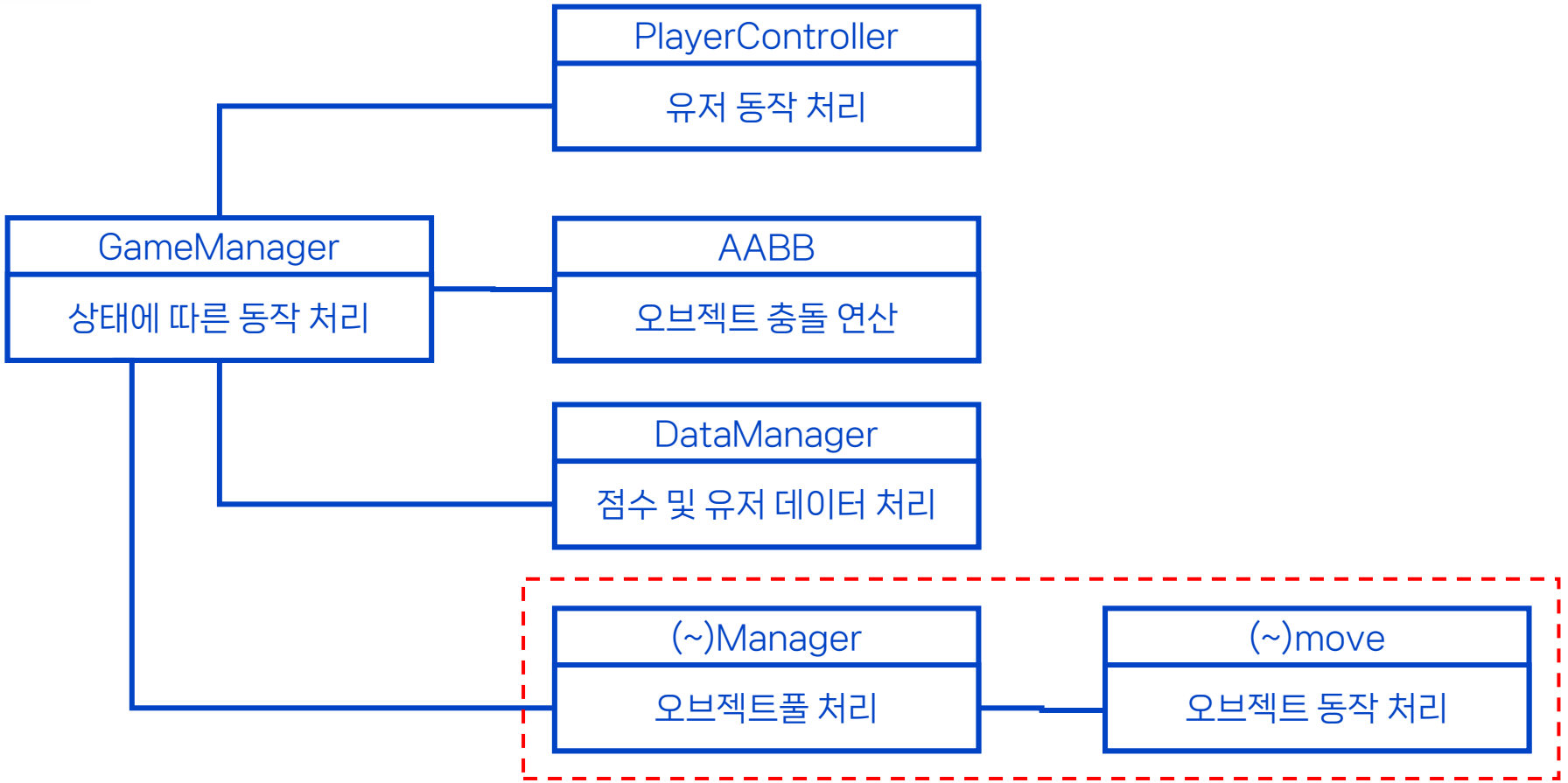
2. 이벤트 : UnityEvent

3. 유저 및 랭킹 : PlayerPrefs

PlayerPrefs	
유저 이름	유저 점수

02. 모듈설명

구조화



* 모든 장애물 및 배경 오브젝트에 동일하게 적용

02. 모듈설명

이벤트

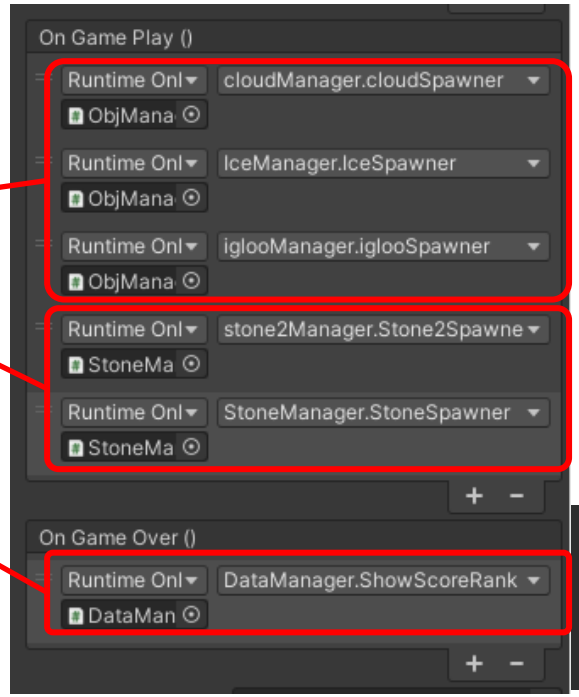
- GameManager.cs

: 게임 상태별 화면 출력, 게임 시간, 이벤트 관리

상태 제어는 해당 스크립트에서만 진행

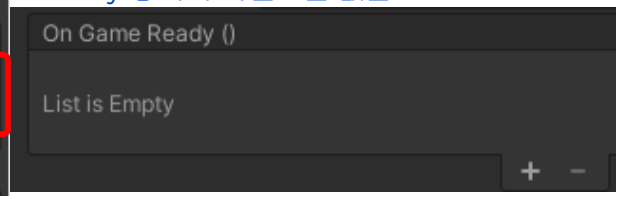
OnGamePlay
* 배경 오브젝트 관리
* 장애물 관리

OnGameOver
* 랭킹 출력




```
public enum GameState
{
    Ready, //타이틀
    Play, //게임 시작 중
    End //게임 오버
}
void Update()
{
    if (state == GameState.Ready)
    {
        GameReady();
        isCounting = false;
    }
    if (state == GameState.Play)
    {
        Stagetext.text = "스테이지 "
            + stageCount;
        GamePlay();
    }
    if (state == GameState.End)
    {
        GameOver();
    }
}
```

* 초기화는 각 관리 Manager가 진행하여 Ready 상태의 이벤트는 없음




```
public UnityEvent OnGameReady;
public UnityEvent OnGamePlay;
public UnityEvent OnGameOver;

//게임 진행 중
public void GamePlay()
{
    ReadyCanvas.SetActive(false);
    PlayCanvas.SetActive(true);
    OverCanvas.SetActive(false);

    PlayCanvas.SetActive(true);
    state = GameState.Play;
    OnGamePlay.Invoke();

    //플레이 시간 측정
    playTime += Time.deltaTime;

    if (playTime >= speedIncreaseInterval)
    {
        IncreaseGameSpeed();
        playTime = 0f;
    }
}
//게임 오버
public void GameOver()
{
    OnGameOver.Invoke();
    ReadyCanvas.SetActive(false);
    PlayCanvas.SetActive(false);
    OverCanvas.SetActive(true);

    Time.timeScale = 0;
    overtext.enabled = true;
}
```

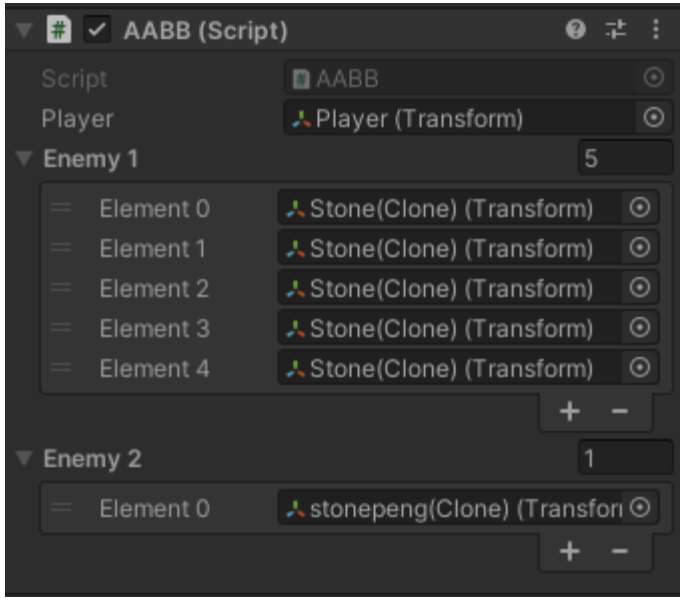
02. 모듈설명

충돌처리

- AABB.cs

: 유저와 장애물의 충돌 여부를 검사

유저의 위치 좌표와 모든 장애물 위치 좌표를 가지고 검사함
→활성화된 오브젝트만 검사




```
public bool IsAABBCollision(Transform player, Transform[] enemy1, Transform[] enemy2)
{
    Vector2 playerMin = player.position - player.localScale / 20f;
    Vector2 playerMax = player.position + player.localScale / 20f;

    foreach (Transform enemy in enemy1)
    {
        if (enemy.gameObject.activeSelf)
        {
            Vector2 enemyMin = enemy.position - enemy.localScale / 20f;
            Vector2 enemyMax = enemy.position + enemy.localScale / 20f;

            if (playerMin.x <= enemyMax.x && playerMax.x >= enemyMin.x &&
                playerMin.y <= enemyMax.y && playerMax.y >= enemyMin.y)
            {
                return true;
            }
        }
    }

    foreach (Transform enemy in enemy2)
    {
        if (enemy.gameObject.activeSelf)
        {
            Vector2 enemyMin = enemy.position - enemy.localScale / 20f;
            Vector2 enemyMax = enemy.position + enemy.localScale / 20f;

            if (playerMin.x <= enemyMax.x && playerMax.x >= enemyMin.x &&
                playerMin.y <= enemyMax.y && playerMax.y >= enemyMin.y)
            {
                return true;
            }
        }
    }

    return false;
}
```

02. 모듈설명

유저 및 랭킹

- DataManager.cs
  : 기록이 필요한 데이터 관리 및 출력 진행

① Ready 상태일 때

- 유저 정보 가져오기

GetInt(BestScore_)	
[유저이름] nowName	[점수] 0

게임 시작 시 현재 입력한 유저의 이름 "now Name" 과 점수를 가져와 이전 기록을 불러옴

② End 상태일 때

- 유저의 최고 점수 기록

SetInt(BestScore_)	
[유저이름] nowName	[점수] bestScore

게임 종료 시 최고 점수가 갱신되면 유저 이름과 최고 점수를 저장하여 기록함

- 유저의 최고 점수 가져오기

SetInt(BestScore_)	
[유저이름] nowName	[점수] 0

유저의 최고 점수를 불러옴

```
if (gameManager.state == GameState.Ready)
{
    scoreText.text = "";
    //플레이타임 초기화
    playTime = 0;

    playCount = PlayerPrefs.GetInt("Playcount", 0);

    UpdatePlayerNameList();
    currentScore = 0;
    nowName = nickname.text;
    bestScore = PlayerPrefs.GetInt("BestScore_" + nowName, 0);
}
```

```
if (gameManager.state == GameState.End)
{
    playCount = PlayerPrefs.GetInt("Playcount", 0);
    ShowScoreRank();

    if (currentScore > bestScore)
    {
        IncreasePlayCount();
        bestScore = currentScore;
        currentScoreText.text = "Score : " + currentScore;
        bestScoreText.text = "[Best Score]₩n" + bestScore;
        PlayerPrefs.SetInt("BestScore_" + nowName, bestScore);
    }

    currentScoreText.text = "Score : " + currentScore;

    // playerName에 해당하는 bestScore 데이터를 가져와서 출력
    int playerBestScore = PlayerPrefs.GetInt("BestScore_" + nowName, 0);
    bestScoreText.text = "[Best Score]₩n" + playerBestScore;
}
```

02. 모듈설명

유저 및 랭킹-클래스

- ScoreEntry

: 유저 이름과 점수를 담음
데이터의 구조화를 위해 사용

```
// 점수와 이름을 담는 클래스
public class ScoreEntry
{
    public string Name { get; private set; }
    public int Score { get; private set; }

    public ScoreEntry(string name, int score)
    {
        Name = name;
        Score = score;
    }
}
```

ShowScoreRank()
: ScoreEntry 객체를 생성하여 이름과 점수를 담고, 이들을 리스트에 저장하여 점수 랭킹을 구성함

UpdatePlayerNameList()

: 게임 시작 시 유저 이름 리스트를 업데이트함

① "PlayerNames" 키에 저장된 데이터 불러옴

② 가져온 이름들을 쉼표로 구분하여 배열로 변환

③ "uniqueNames" List에 이름 저장

④ 현재 유저의 이름 "nowName"이 없으면 추가

⑤ "uniqueNames" List를 쉼표로 구분하여 "PlayerNames" 키에 다시 저장

```
public void ShowScoreRank()
{
    List<ScoreEntry> rankList = new List<ScoreEntry>();

    foreach (string key in PlayerPrefs.GetString("PlayerNames", "").Split(','))
    {
        int score = PlayerPrefs.GetInt("BestScore_" + key, 0);
        rankList.Add(new ScoreEntry(key, score));
    }

    rankList.Sort((a, b) => b.Score.CompareTo(a.Score));

    int rankCount = Mathf.Min(rankList.Count, 10); // 최대 10개의 순위를 표시

    string rankText = "";
    string nameText = "";
    for (int i = 0; i < playCount; i++)
    {
        int rank = i + 1;
        nameText += rank + "위 " + rankList[i].Name + "₩n";
        rankText += rankList[i].Score + "₩n";
    }

    RankNames.text = nameText;
    RankScores.text = rankText;
}

private void UpdatePlayerNameList()
{
    string[] playerNames = PlayerPrefs.GetString("PlayerNames", "").Split(',');
    List<string> uniqueNames = new List<string>(playerNames);

    if (!uniqueNames.Contains(nowName) && !string.IsNullOrEmpty(nowName))
    {
        uniqueNames.Add(nowName);
        PlayerPrefs.SetString("PlayerNames", string.Join(",", uniqueNames.ToArray()));
    }
}
```

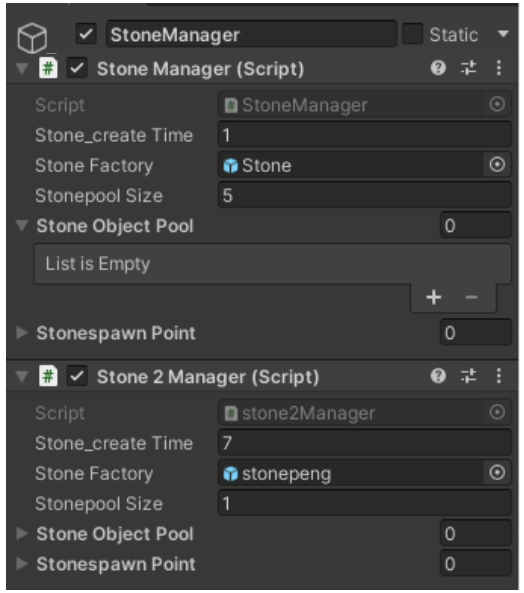
02. 모듈설명

오브젝트 풀

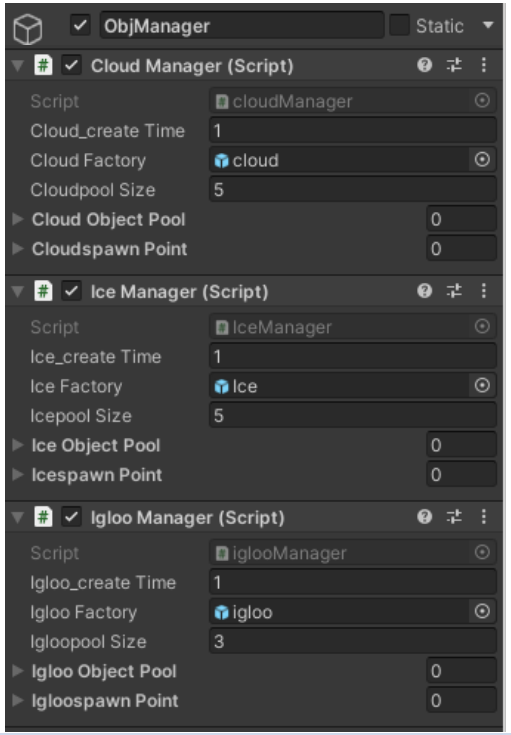
- (~)Manager.cs
  : 반복적인 생성이 필요한 오브젝트에 적용

오브젝트 풀 진행

장애물



배경




```
void Start()
{
    gameManager = GameObject.Find("GameManager").GetComponent<GameManager>();
    aabb = GameObject.Find("CollisionManager").GetComponent<AABB>();

    //stone 복제
    stone_createTime = Random.Range(stone_minTime, stone_maxTime);

    aabb.enemy1 = new Transform[stonepoolSize];
    stoneObjectPool = new List<GameObject>();
    for (int i = 0; i < stonepoolSize; i++)
    {
        GameObject stone = Instantiate(stoneFactory);
        stoneObjectPool.Add(stone);
        stone.SetActive(false);
    }
}
void StoneSpawner()
{
    if (!gameManager.isCounting)
    {
        stone_currentTime += Time.deltaTime; // 현재시간을 측정

        if (stone_currentTime > stone_createTime) // 현재 시간이 돌 생성 시간을 넘으면
        {
            if (stoneObjectPool.Count > 0)
            {
                GameObject stone = stoneObjectPool[0];
                stoneObjectPool.Remove(stone);
                stone.SetActive(true); // 돌을 활성화

                if (stonespawnPoint.Length > 0)
                {
                    int index = Random.Range(0, stonespawnPoint.Length);
                    stone.transform.position = stonespawnPoint[index].position;
                }
            }

            stone_currentTime = 0;
            stone_createTime = Random.Range(stone_minTime, stone_maxTime);
        }
    }
}
```

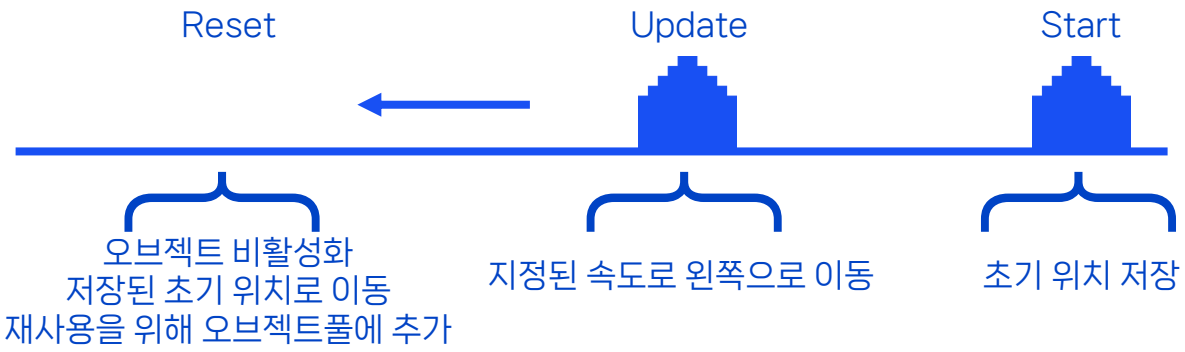
02. 모듈설명

오브젝트 동작

- (~)move.cs

: 오브젝트 활성화 후 동작

재사용을 위한 동작 및 초기화 진행




```
void Start()
{
    gameManager = GameObject.Find("GameManager").GetComponent<GameManager>();
    stoneManager = GameObject.Find("StoneManager").GetComponent<StoneManager>();
    startPos = this.transform.position;
}


void Update()
{
    if (!gameManager.isCounting)
    {
        transform.Translate(Vector2.left * Speed * Time.deltaTime);

        if (transform.position.x <= -11f)
        {
            stoneReset();
        }

        if (gameManager.state == GameState.Ready)
        {
            stoneReset();
        }
    }
}

public void stoneReset()
{
    gameObject.SetActive(false); // 돌을 비활성화
    this.transform.position = startPos; // 처음 위치로 이동
    stoneManager.stoneObjectPool.Add(gameObject); // 돌을 다시 stoneObjectPool에 추가
}
```

02. 모듈설명

유저 동작

- PlayerController.cs

  : 점프 동작

```
public float jumpForce = 11f; // 점프 힘
public float gravity = 20f; // 중력 가속도

private float verticalVelocity = 0f; // 수직 속도
private Vector2 startPos;
private bool isJump = true;
```

Jump()함수를 호출하는
버튼 클릭

게임 상태=Ready,
Y축이 -2.15f보다
작을 경우

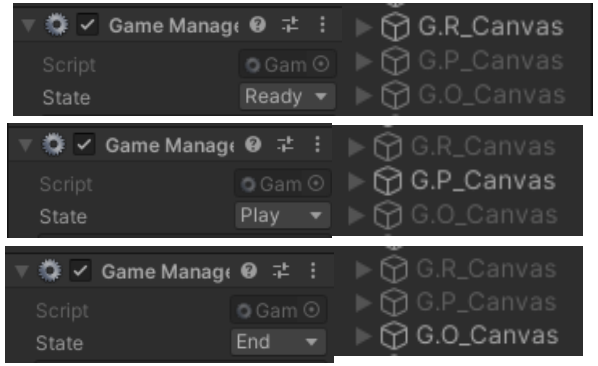
현재 위치를 초기 위치로 이동
(화면 밖으로 이탈 방지)

중력을 적용하여 수직 속도 조절
후 이동

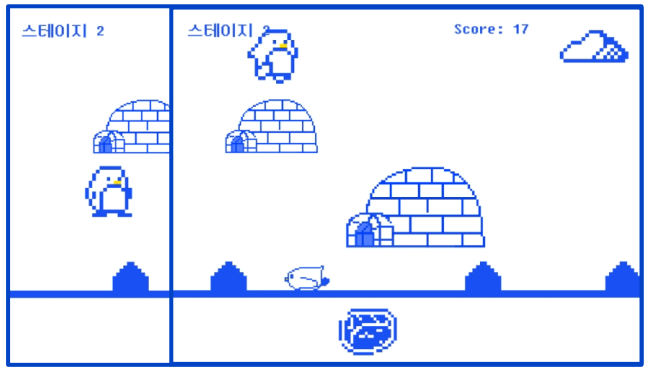
```
void Update()
{
    if (gameManager.state == GameState.Ready)
    {
        transform.position = startPos;
        isJump = true;
    }

    if (transform.position.y <= -2.15f)
    {
        transform.position = startPos;
        isJump = true;
    }

    ApplyGravity(); // 중력 적용 및 이동
}

private void Jump()
{
    gravity = 20f;
    if (!gameManager.isCounting && isJump)
    {
        isJump = false;
        verticalVelocity = jumpForce;
    }
}

private void ApplyGravity()
{
    if (transform.position.y <= -2.15f)
    {
        gravity = 0;
    }
    else
    {
        gravity = 20f;
    }
    verticalVelocity -= gravity * Time.deltaTime;
    transform.position += new Vector3(0f, verticalVelocity * Time.deltaTime, 0f);


    if (transform.position.y <= -2.15f)
    {
        transform.position = startPos;
    }
}
```

03. 렌더링 결과

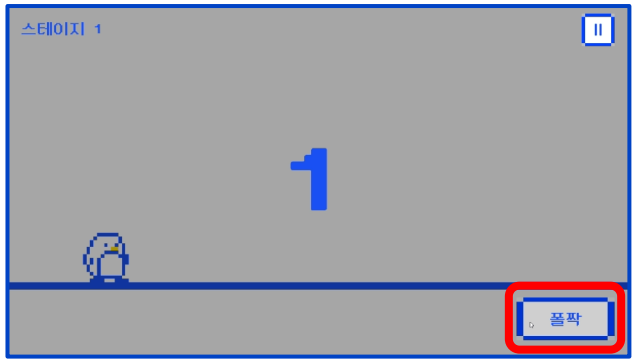
① 각 상태별 화면 출력 = true



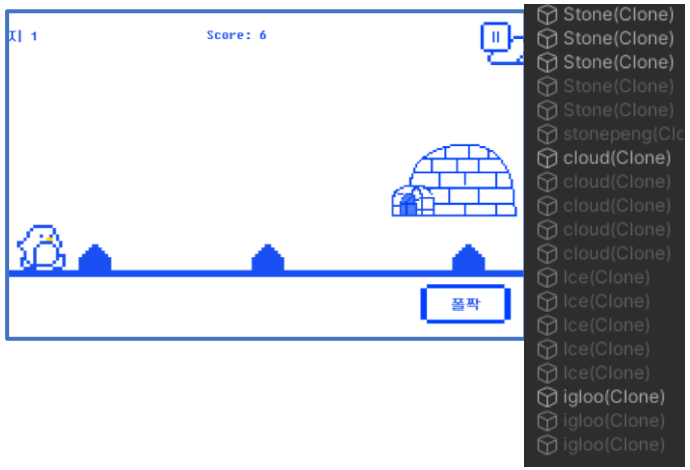
② 점프 2회 = true



③ 카운트 시 점프 작동X = true



④ 오브젝트 활성화 및 비활성화 = true



⑤ 장애물 충돌시 게임오버, 기록 출력 = true