-클라이언트 프로그래머 지원-

-서현재-

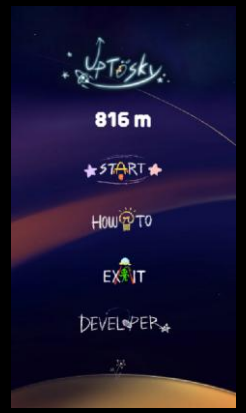
기술스택
- DiectX (기초)
- Solved.ac – Platinum5
- C# (중급)
- C++ (중급)
- Unity (중급)
- Unreal (기초)
- Git

1. The Sword
itch.lo 출시

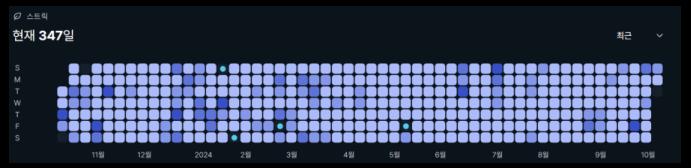
2. MessMath
3회 웅진 씽크빅 게임 개발 챌린지 본선 진출

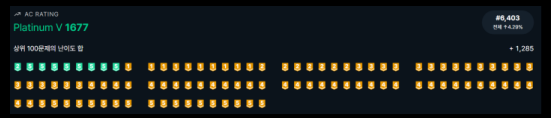
3. 던전포차
인디크래프트 출품작

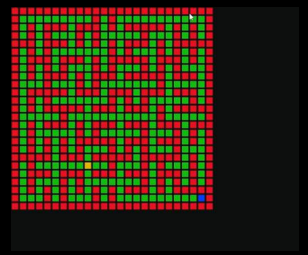
4. UpToSky
구글 플레이 출시 게임



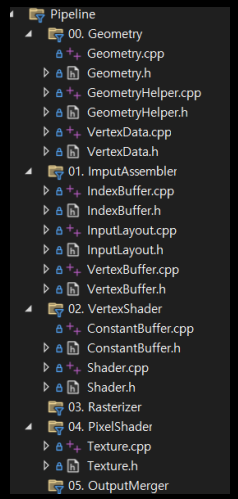
꾸준한 알고리즘 공부







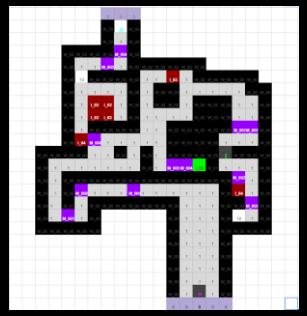
길 찾기 알고리즘
공부용 미로 찾기



렌더링 파이프라인에 따라서 코드를 작성하고 적용해보았습니다.
DirectX11 책에 나오는 내용을 가져오되, 제 방식대로 코드를 이해하고 재구성

https://youtu.be/Swc5I2-JCMQ

개요: RPG형식 1인 전략 게임
개발 기간 : 2024.05~
사용 도구 : Unity, C#, Git, GithubDesktop
개발 인원 : 4인(프로그래머2, 아트1, 기획자1)
담당 업무: 던전 맵 자동생성 툴 제작, 전투 게임 로직 구현, UI자동화, Exel 데이터 연동, 다국언어

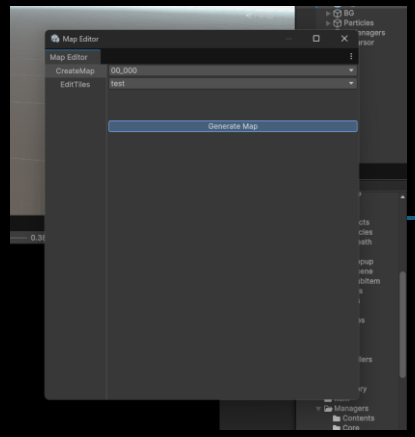


Map 자동생성 Tool로직

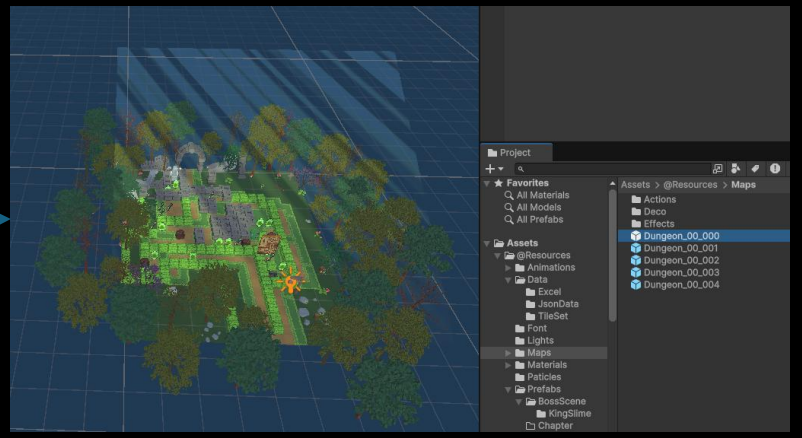
Map 데이터와 데이터 정보값 저장.

도입배경	**큐브 모양의 맵을 하나하나 이어서 만들기 위해서 많은 시간이 소요됨.**
개선된 사항	맵 데이터가 존재할 경우 맵 생성 시간이 **90%** 이상 단축됨.
개선된 사항	코드를 보면서 로직을 총괄적으로 관리 가능

Exel Map Data 만들기 → Exel을 읽고, Editor에서 Hierarchy창에 오브젝트들 생성 → 오브젝트들을 Prefab화 → Map 저장 → 게임 시작시 Map을 불러오기

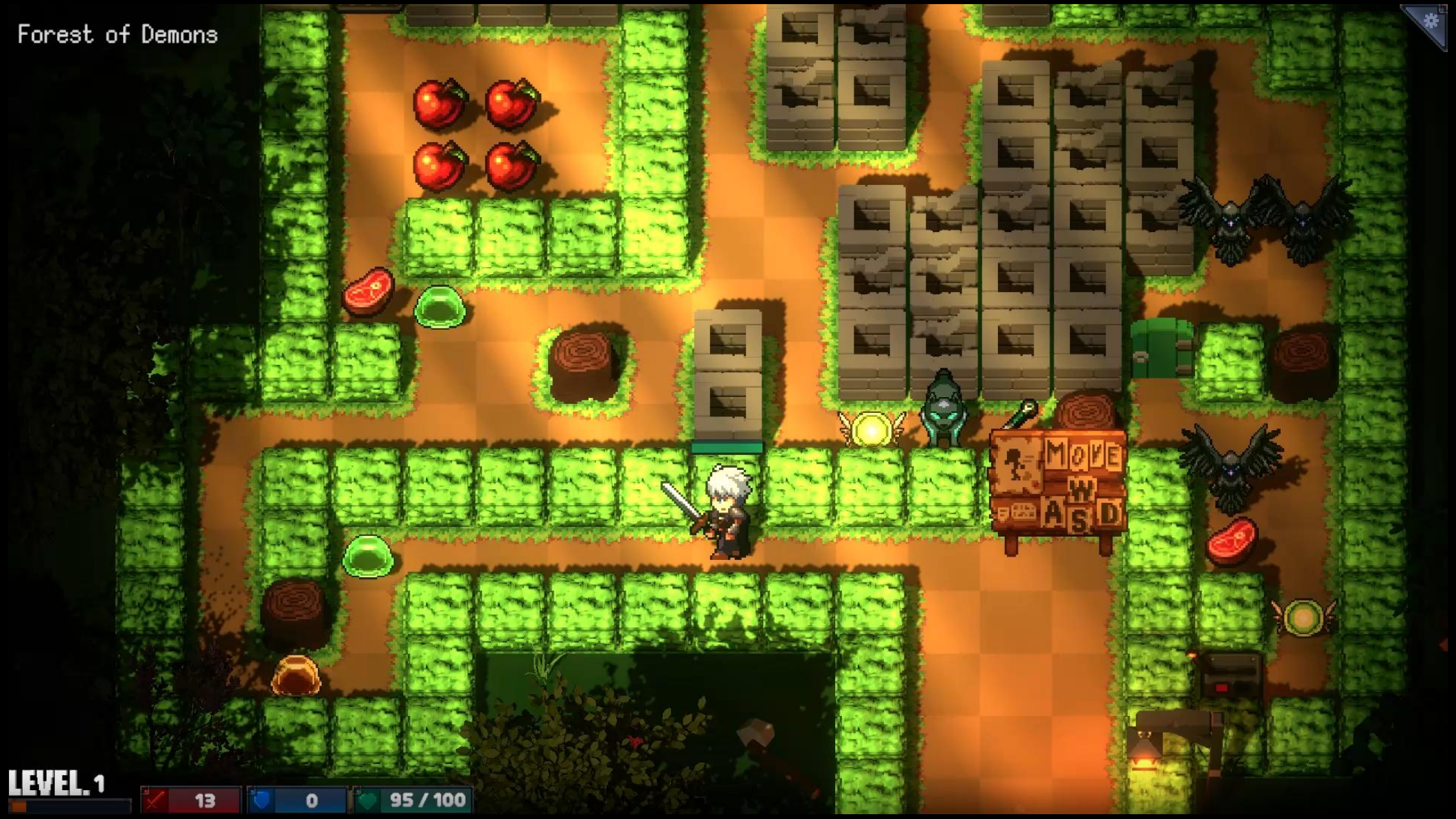


MapEditor 창 생성.



맵 자동 생성 결과 모습 (자동 프리랩화)

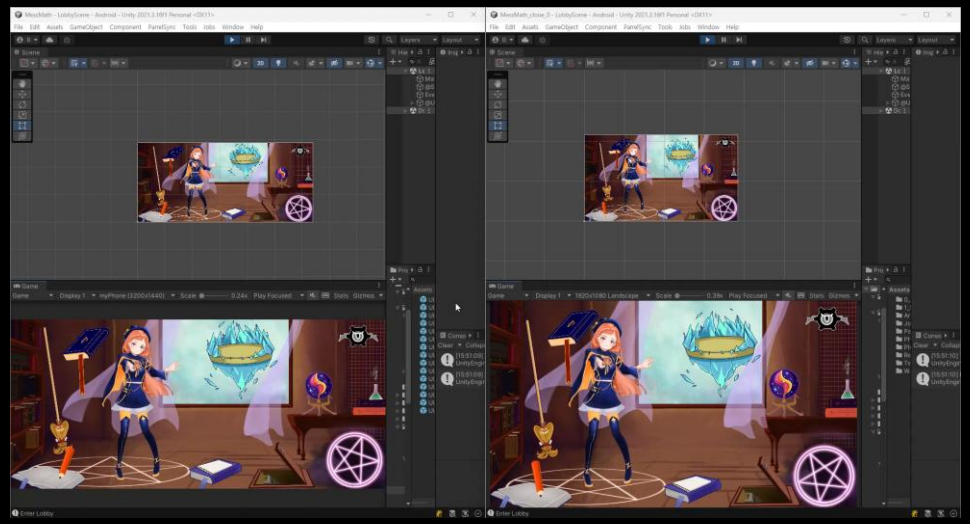
전투 영상



https://youtu.be/UUPn5YygFgE?si=fOjGS7frNqcJWdBN

목적:	게임 개발 공모전을 위한 수학 게임
개발 기간 :	2023.04~ 2023.08
담당 기능을 개발하기 위한 사용 도구 :	Unity, C#, Git, GithubDesktop, 포톤, MathpidAPI, Firebase
개발 인원 :	5인(프로그래머4, 아트1)
담당 업무:	애니메이션, UI, 포톤 서버, 멀티, MathpidAPI연동, 데이터 연동, 스토리모드, 연습모드

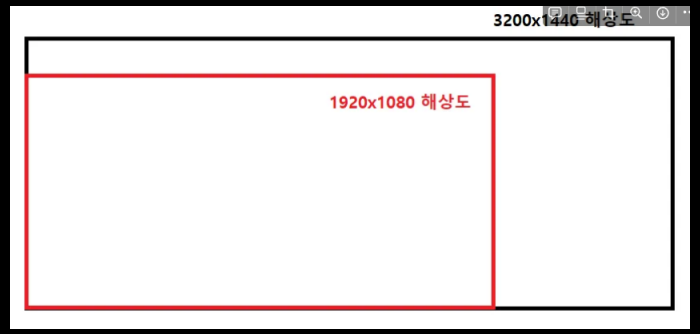
해상도 트러블 슈팅



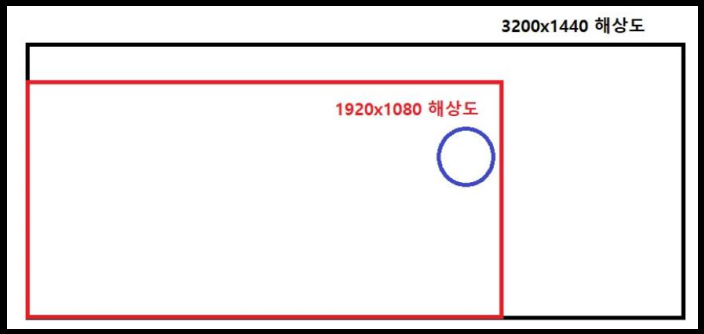
서로 다른 해상도를 가진 두 기기로 PVP를 진행하면 플레이가 이상해진다.

해상도가 다른 두 환경에서는 물체의 위치가 이상하게 잡힌다.

이렇게 되는 경우 두 화면에서 동일한 좌표를 가지지만 화면상의 위치는 달라지게 된다.



1920x1080 해상도의 기기에서는 오른쪽 가장자리에 매우 가까이 있다고 보일 것이고, 3200x1440 해상도에서는 거의 화면의 정중앙에서 살짝 오른쪽위로 이동한 정도의 위치에 있다고 보일 것이다.



그리고 플레이어의 이동속도도 문제가 있었다. 플레이어는 동일한 단위(unit)으로 이동하므로 작은 해상도를 가진 플레이어가 상대적으로 더 높은 이동속도를 가지게 되었다.

문제 해결

앞선 상황에서 문제를 해결하기 위해 오브젝트 위치를 왜곡하여 해결하려했다.

↓

MasterClient에서 생긴 이 오브젝트의 정보를, 다른 Client에서 수신할 때는 자기 해상도에 맞게끔 조정해서 가져오는것이다.

↓

그러면 동일한 위치에 있는것처럼 보일 것이다.

↓

그래서 우리는, MasterClient에서 오브젝트의 전체 해상도 대비 위치값, 그러니까 x좌표를 그냥 보내는것이 아니라, (x좌표 / 가로 해상도)와 (y좌표 / 세로 해상도)를 보내기로 결정했다.

↓

이 비율을 전송하면, 수신한 Client는 자신의 해상도에 이 비율을 곱해서 이 오브젝트가 있어야 할 위치를 도출해낼 수 있을것이다.

```
Vector3 transPosIntoRatio()
{
    float actualH = Screen.width / 3200f * 1440f;
    float xRatio = RT.position.x / Screen.width;
    float yRatio = (RT.position.y - ((Screen.height - actualH) / 2f)) / actualH;

    return new Vector3(xRatio, yRatio, 0f);
}

void transRatioIntoPos(Vector3 vector3)
{
    float xRatio = vector3.x;
    float yRatio = vector3.y;
    float actualH = Screen.width / 3200f * 1440f;

    float xPos = Screen.width * xRatio;
    float yPos = actualH * yRatio + ((Screen.height - actualH) / 2f);

    curPos = new Vector3(xPos, yPos, 0f);
}
```

연산 과정을 보면, 좀 특이한 과정이 하나 있는데, 이는 레터박스 때문이다.

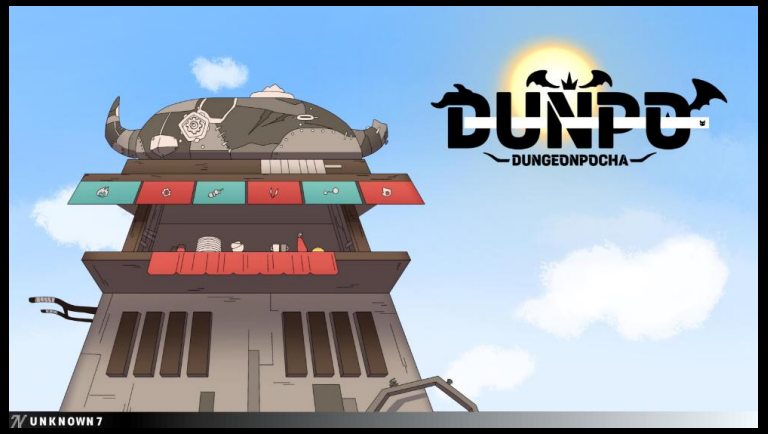
우리는 여태까지 가변 해상도를 고려하지않고 3200x1440 해상도를 기준으로 작업했고, 1920x1080 해상도처럼 가로로 눕혔을 때 세로로 더 높은 해상도는 위 아래로 레터박스가 생긴다.

그래서 왼쪽과 같이 구현했다.

```
public class PlayerControllerOnlyinPvp : MonoBehaviourPun, IPunObservable
{
  float referenceWidth = 3200f; // 기준 해상도의 너비
  float referenceHeight = 1440f; // 기준 해상도의 높이
  float currentWidth = Screen.width; // 현재 화면의 너비
  float currentHeight = Screen.height; // 현재 화면의 높이

  float widthRatio;
  float heightRatio;
  float adjustedSpeed;

  void Awake()
  {
    widthRatio = currentWidth / referenceWidth;
    heightRatio = currentHeight / referenceHeight;

    adjustedSpeed = _speed * Mathf.Min(widthRatio, heightRatio);
  }
}
```

캐릭터 속도 동기화 문제도 해상도와 비슷하게 해결할 수 있을거같았다.

Awake 단계에서 해상도에 맞는 _speed의 adjustedSpeed를 산출하고 이를 _speed에 적용했다.



https://youtu.be/ieY6bY_Rgyg?si=J2sLyG0pGJHbj8KL

목적: 인디크래프트 공모전 게임
개발 기간 : 2024.02~ 2024.04
사용 도구 : Unity, C#, Git, GithubDesktop
개발 인원 : 7인(프로그래머3, 아트3, 기획자 1)
담당 업무: 애니메이션, UI, 전투 구현, 다국 언어, 데이터 연동


```
IEnumerator CoSpawnMonster()
{
    while (true)
    {
        WaitForSeconds spawnDelay = new WaitForSeconds(0.2f);
        yield return spawnDelay;

        if (Managers.Game.MonsterCount < 100)
            CreateMonsterIcon();
    }
}
```

던전에 입장하면
CoSpawnMonster가 실행된다.

0.2초마다 총 몬스터가 100마리
미만이라면 계속
CreateMonsterIcon을 실행한다.

```
public void CreateMonsterIcon()
{
    Transform parent = GameObject.Find("MiniDungeon").gameObject.transform;

    GameObject startPoint = GameObject.Find("MonsterStartPoint");
    MonsterController monster = Managers.Resource.Instantiate("Creature/Monster", parent).GetComponent<MonsterController>();
    monster.GetComponent<SpriteRenderer>().sprite = Managers.Resource.Load<Sprite>($"Sprites/MiniDungeon/Monsters/Mon_{Managers.Game.Dungeon

    Managers.Game.Monsters.Add(monster.GetComponent<MonsterController>());
    Managers.Game.MonsterCount++;
    monster.Attack = Managers.Game.DungeonStage.MonATk + Managers.Game.DungeonStage.NextATK * 1;
    monster.Speed = Managers.Game.DungeonStage.MonSPD + Managers.Game.DungeonStage.NextSPD * 1;
    monster.Hp = Managers.Game.DungeonStage.MonHP + Managers.Game.DungeonStage.NextHP * 1;
    monster.name = $"Monster";
    monster.transform.position = new Vector3(startPoint.transform.position.x, startPoint.transform.position.y + Random.value, 1f);
    monster._originalYPos = monster.transform.localPosition.y;
    monster._originalXPos = monster.transform.localPosition.x;
    monster.StartCoMoveToHunter();
}
```

CreateMonsterIcon은
스테이지 정보에 맞게
몬스터 능력치가 설정되고
플레이어 스폰 쪽으로
계속해서 이동한다.

전투 구현

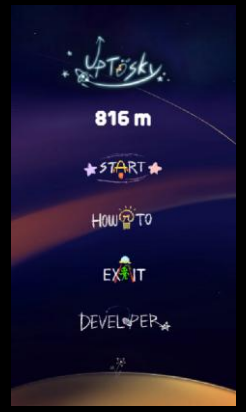
```
IEnumerator CoAttack(HunterController hunter)
{
    WaitForSeconds delay = new WaitForSeconds(0.5f);
    if (_isAttack == false)
    {
        while (true)
        {
            if (hunter == null)
                break;
            if (hunter.Hp <= 0)
                break;

            _isAttack = true;
            gameObject.transform.position += new Vector3(-0.05f, 0, 0);
            hunter.Hp -= this.Attack;
            yield return delay;
            gameObject.transform.position += new Vector3(+0.05f, 0, 0);
            yield return delay;
        }
    }

    _isAttack = false;
    StartCoMoveToHunter();
}
```

플레이어 아이콘을 만나면 그 상대로 공격을 하고 피가 없을 때 까지 공격한다.

```
IEnumerator CoAttack(MonsterController monster)
{

    while (true)
    {
        WaitForSeconds attackSpeed = new WaitForSeconds(Managers.Data.DungeonInfoDic[Managers.Game.curLevelIndex.ToString()].SpeedByLevel); //!!!

        if (monster == null)
            break;
        if (this.Hp <= 0)
            break;
        if (monster.Hp <= 0)
            break;

        _isAttack = true;

        gameObject.transform.position += new Vector3(+0.05f, 0, 0);
        monster.Hp -= this.Attack;
        GameObject dungeonHitStar = Managers.Resource.Instantiate("MiniDungeon/DungeonHitStar", transform);
        dungeonHitStar.transform.position = (gameObject.transform.position + monster.transform.position) / 2;
        float playTime = dungeonHitStar.GetComponent<Animator>().GetCurrentAnimatorStateInfo(0).length;
        Destroy(dungeonHitStar, playTime);

        if ((monster.Hp) > 0)
        {
            //OnPlayRandomAttackSound();
        }

        yield return attackSpeed;
        gameObject.transform.position += new Vector3(-0.05f, 0, 0);
        yield return attackSpeed;
    }

    _isAttack = false;
    StartCoMoveToMonster();
}
```

```
public void OnAttack(MonsterController monster)
{
    if (monster == null)
        return;
    _monster = monster;
    StartCoroutine(CoAttack(monster));
    monster.OnAttack(this);
}
```

플레이어도 몬스터를 만나면 공격을 시작한다.



https://youtu.be/44kZ5xZ0Jso?si=ShWp25twiPBS69ae

목적: 구글플레이 출시 게임
개발 기간 : 2023.02~ 2024.04
사용 도구 : Unity, C#, Git, GithubDesktop
개발 인원 : 3인(프로그래머3)
담당 업무: 애니메이션, UI, 게임 로직, Google Ads

Google Ads 적용

```
private InterstitialAd interstitialAd;
public InterstitialAd InterstitialAd { get { return interstitialAd; } }

public void Init()
{
    // Initialize the Google Mobile Ads SDK.
    MobileAds.Initialize((InitializationStatus initStatus) =>
    {
        // This callback is called once the MobileAds SDK is initialized.
        LoadInterstitialAd();

    });
}
// 전면 광고 로드
// These ad units are configured to always serve test ads.
```

광고 내용을 로드하는 함수.

```
// 전면 광고 로드
/// <summary>
/// Loads the interstitial ad.
/// </summary>
public void LoadInterstitialAd()
{
    // Clean up the old ad before loading a new one.
    if (interstitialAd != null)
    {
        interstitialAd.Destroy();
        interstitialAd = null;
    }

    Debug.Log("Loading the interstitial ad.");

    // create our request used to load the ad.
    var adRequest = new AdRequest.Builder().Build();

    // send the request to load the ad.
    InterstitialAd.Load(_adUnitId, adRequest,
        (InterstitialAd ad, LoadAdError error) =>
        {
            // if error is not null, the load request failed.
            if (error != null || ad == null)
            {
                Debug.LogError("interstitial ad failed to load an ad " +
                               "with error : " + error);
                return;
            }

            Debug.Log("Interstitial ad loaded with response : "
                      + ad.GetResponseInfo());

            interstitialAd = ad;
        });

}
```

실제로 광고를 보여줄 때 사용되는 함수

```
// 전면 광고 표시
/// <summary>
/// Shows the interstitial ad.
/// </summary>

public void ShowAd()
{
    if (interstitialAd != null && interstitialAd.CanShowAd())
    {
        Debug.Log("Showing interstitial ad.");
        interstitialAd.Show();
        RegisterEventHandlers(interstitialAd);
        RegisterReloadHandler(interstitialAd);
    }
    else
    {
        Debug.LogError("Interstitial ad is not ready yet.");
    }
}
```

Google Ads

광고를 끄는 등 이벤트가
발생될 때 사용되는 함수.
우리의 경우 광고를 끄면
GameScene으로 이동.
광고의 에러가 생겨도
GameScene으로 이동한다.

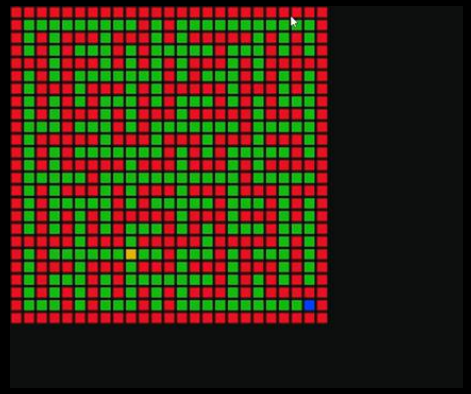
```
private void RegisterEventHandlers(InterstitialAd ad)
{
    // Raised when the ad is estimated to have earned money.
    ad.OnAdPaid += (AdValue adValue) =>
    {
        Debug.Log(String.Format("Interstitial ad paid {0} {1}.",
            adValue.Value,
            adValue.CurrencyCode));
    };
    // Raised when an impression is recorded for an ad.
    ad.OnAdImpressionRecorded += () =>
    {
        Debug.Log("Interstitial ad recorded an impression.");
    };
    // Raised when a click is recorded for an ad.
    ad.OnAdClicked += () =>
    {
        Debug.Log("Interstitial ad was clicked.");
    };
    // Raised when an ad opened full screen content.
    ad.OnAdFullScreenContentOpened += () =>
    {
        Debug.Log("Interstitial ad full screen content opened.");
    };
    // Raised when the ad closed full screen content.
    ad.OnAdFullScreenContentClosed += () =>
    {
        Debug.Log("Interstitial ad full screen content closed.");
        //interstitialAd.Destroy();
        // TODO
        Debug.Log("To Game Scene");
        UnityEngine.SceneManagement.SceneManager.LoadScene("GameScene");

    };
    // Raised when the ad failed to open full screen content.
    ad.OnAdFullScreenContentFailed += (AdError error) =>
    {
        Debug.LogError("Interstitial ad failed to open full screen content " +
                       "with error : " + error);

        Debug.Log("To Game Scene");
        UnityEngine.SceneManagement.SceneManager.LoadScene("GameScene");

    };
}
```

다음 전면 광고를 미리 로드하는 함수.

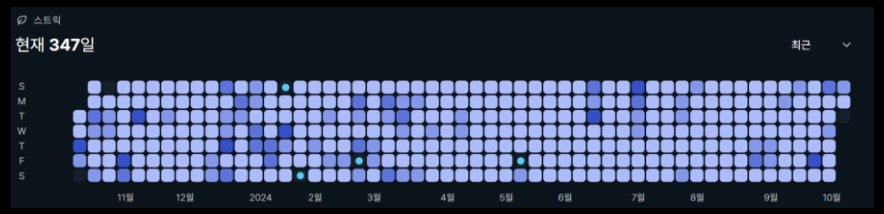
광고를 한번 보여주면 Reload함수가
실행되어 광고를 저장하고 있다.
다시 광고가 나오면 저장된 광고가
나오고 Reload를 하면서 반복.

```
// 다음 전면 광고 미리 로드
private void RegisterReloadHandler(InterstitialAd ad)
{
    // Raised when the ad closed full screen content.
    ad.OnAdFullScreenContentClosed += () =>
    {
        Debug.Log("Interstitial Ad full screen content closed.");

        // Reload the ad so that we can show another as soon as possible.
        LoadInterstitialAd();
    };
    // Raised when the ad failed to open full screen content.
    ad.OnAdFullScreenContentFailed += (AdError error) =>
    {
        Debug.LogError("Interstitial ad failed to open full screen content " +
                       "with error : " + error);

        // Reload the ad so that we can show another as soon as possible.
        LoadInterstitialAd();
    };
}
```

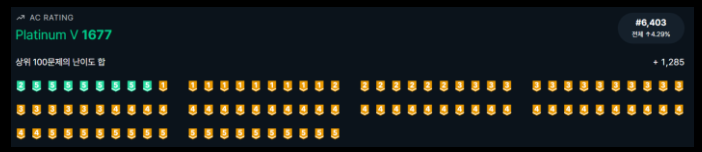
기타이력
알고리즘

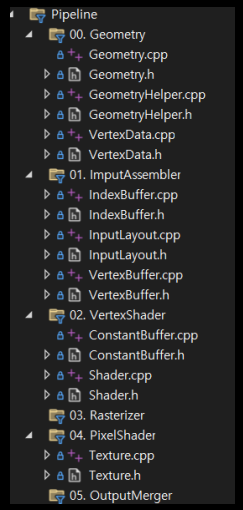


길찾기 알고리즘 공부용 미로 찾기



꾸준한 알고리즘 공부





렌더링 파이프라인에 따라서 코드를 작성하고 적용해보았습니다.
DirectX11 책에 나오는 내용을 가져오 되, 제 방식대로 코드를 이해하고 재구성했습니다.