Portfolio

Shin Beom Su

“상상을 현실로 만들 수 있는 개발자 신범수입니다.”

Unity 클라이언트 개발자

신범수

SteamWorks 파트너

Google Play Console 개발자

Unity Asset Store 퍼블리셔

역량

- 클라이언트 개발 (Unity, C#)
- 문제 해결 (버그 트래킹, 로직 개선)
- 협업 (Git, PR 및 리포지토리 관리)

강점

- 출시 경험 3회(Steam, Google Play, Unity Asset Store)
- 수상 4회(교내 / 지역 / 전국)
- 인디게임 개발팀 리딩

Mail : qjatn147852@naver.com

Github : https://github.com/shin0624

Blog : https://shin0624.tistory.com

HERESIS

#3D #탈출 #공포 #Steam 출시작

Unity | 3ds Max | C# | Git | Steam

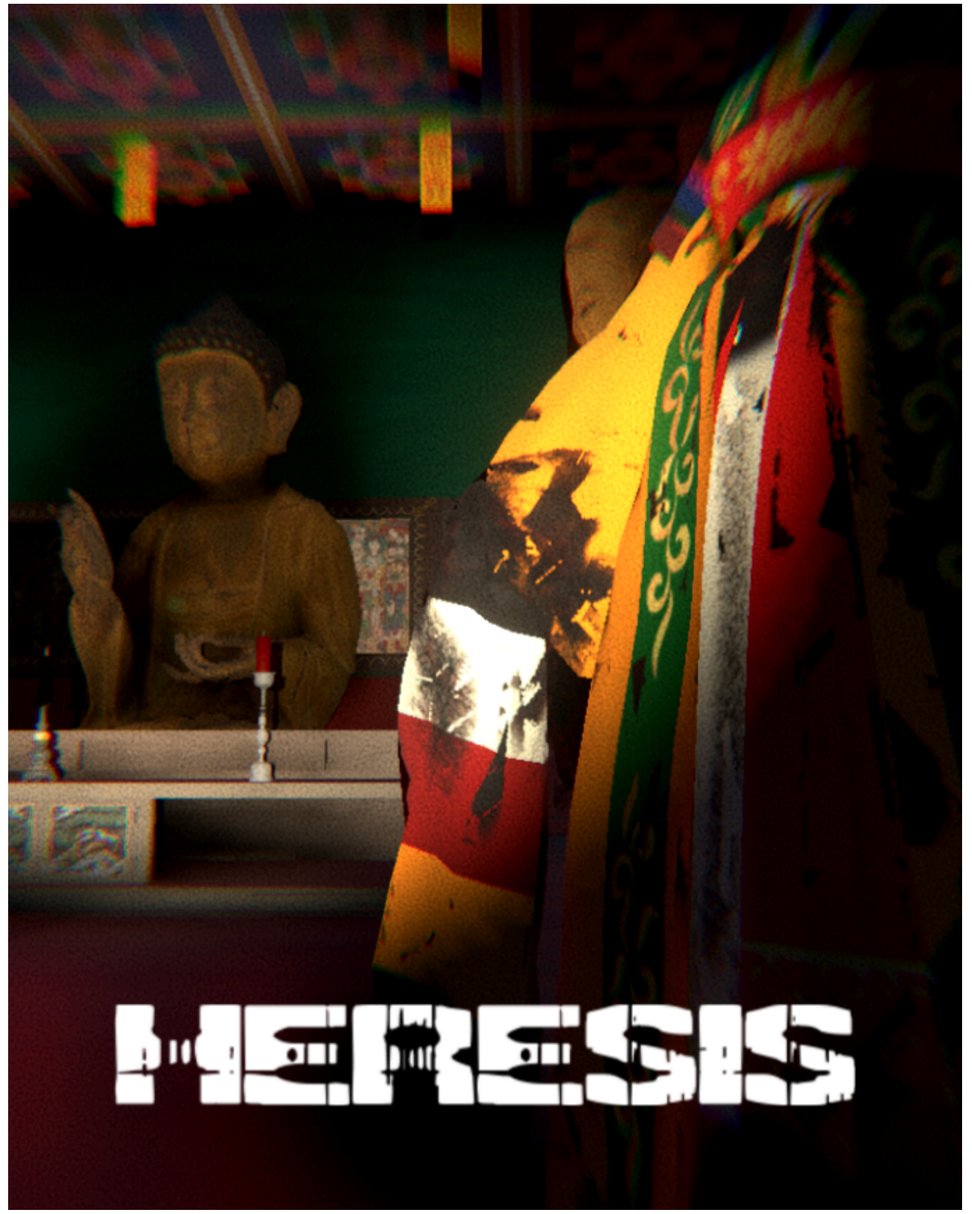
담당 업무

[클라이언트 개발]

- 에너미 AI 개발
  - ○ 행동 트리와 AI Navigation을 사용한 에너미 의사 결정 로직 구현
  - ○ Mixamo, Animator를 사용한 에너미 FSM 구현
- 유저 경험 향상 요소 개발
  - ○ Perlin Noise 알고리즘을 활용한 피격 시 카메라 진동 효과 구현
  - ○ 마우스 클릭으로 입력 받은 패스워드를 처리하는 도어락 UGUI 구현
- 시네마틱 컷신 개발
  - ○ Timeline, Cinemachine을 사용한 인트로, 이벤트 컷신 구현
  - ○ Dolly Camera와 Virtual Camera를 조합하여 자연스러운 공간 전환 효과 구현

[프로젝트 관리]

- SteamWorks 파트너 등록 및 Steam 출시 전담
- Git Repository 및 PR 관리, 사용법 공유
- 출시 후 피드백 기반 1차 업데이트 전담

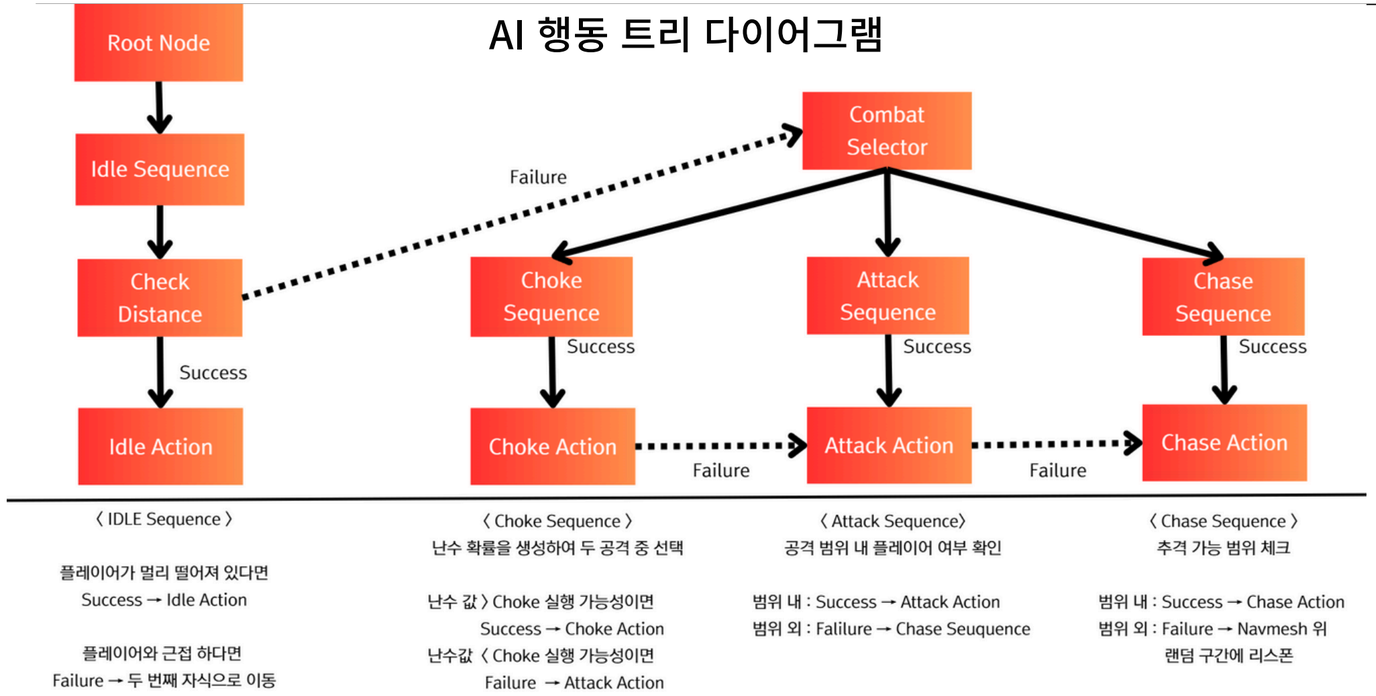


GitHub

Steam

Behavior Tree와 AI Navigation을 사용한 에너미 AI 구현

보스 에너미의 의사 결정 알고리즘을 Behavior Tree 기반으로 구현하였습니다.



[의도]

- Behavior Tree로 에너미의 행동을 노드 단위로 분리
- 공격, 추격, 대기 상태를 독립적으로 구성
- 트리 구조로 표현하였기 때문에 “어떤 행동이 언제 실행되었는지” 명확한 추적 가능 → 유지 보수 / 디버깅 용이

[동작 원리]

SequenceNode

모든 자식 노드를 순차적으로 실행.

▼

SelectorNode

자식 노드 중 하나라도 성공하면 Success

모든 자식이 실패하면 Failure를 반환.

Behavior Tree - 문제 해결 사례

플레이어 추격에 실패한 후 IDLE 상태가 무한 반복되는 문제를 해결한 사례입니다.

[원인 분석]

- 플레이어 추격 가능 범위 이탈 ▶ 에너미는 랜덤 위치로 이동 후 플레이어를 기다립니다.
- 기존 코드 : NavMesh.SetDestination()으로 랜덤 위치로 이동하며 Failure를 반환
  → 부모 노드가 추적 실패로 인식하고 IDLE 상태로 전환됩니다.

```
public override Status Evaluate()//플레이어와의 거리 확인
{
    float distanceToPlayer = Vector3.Distance(shaman.transform.position, shaman.player.position);

    if(distanceToPlayer > respawnDistance)//너무 멀어지면 랜덤위치로 이동하여 대기하도록 한다.
    {
        NavMeshHit hit;
        Vector3 randomDirection = Random.insideUnitSphere * respawnDistance;//네비메시 위의 원형 랜덤 구간 * 거리
        randomDirection+=shaman.player.position;//Navmesh위의 새로운 위치로 이동.

        if(NavMesh.SamplePosition(randomDirection, out hit, respawnDistance, NavMesh.AllAreas))
        {
            shaman.agent.SetDestination(hit.position);//랜덤위치가 존재할 시 해당 장소로 이하여 대기.
            hasRoared = false; // 포효 상태 초기화
            return Status.Failure;
        }
    }
```

[해결]

```
public override Status Evaluate()
{
    //플레이어와의 거리 확인
    float distanceToPlayer = Vector3.Distance(shaman.transform.position, shaman.player.position);

    if(distanceToPlayer > respawnDistance)//너무 멀어지면 리스폰되거나, 그 자리에서 IDLE상태가 된다.
    {
        int maxAttempts = 10;//랜덤 위치를 찾기 위한 시도 횟수에 제한을 둔다.
        int attempts = 0;
        NavMeshHit hit;

        while(attempts < maxAttempts)//시도 제한 횟수까지 반복하며 리스폰 가능 위치를 찾는다.
        {
            Vector3 randomDirection = randomDirection.insideUnitSphere * respawnDistance;
            randomDirection += shaman.player.position;

            //SamplePosition 메서드를 사용하여 최대 respawnDistance 까지 탐색하며 랜덤 위치를 찾는다.
            if(NavMesh.SamplePosition(randomDirection, out hit, respawnDistance, NavMesh.AllAreas))
            {
                shaman.transform.position = hit.position; // 에너미 위치를 랜덤 위치로 리스폰.
                shaman.agent.ResetPath(); // 기존 경로 초기화
                hasRoared = false; // 포효 상태 초기화

                return Status.Success; // 리스폰 후 Idle 상태로 전환
            }
            attempts++;
        }
        shaman.agent.ResetPath(); // 이동 중지
        hasRoared = false; // 포효 상태 초기화
        return Status.Success; //유효한 위치를 찾지 못한 경우, 현재 위치에서 Idle 상태 유지
    }
    shaman.agent.SetDestination(shaman.player.position);  //플레이어 추적
```

- 생성된 랜덤 위치를 Position값에 대입하여 에너미를 리스폰(ReSpawn) ▶ IDLE상태로 전환 ▶ 플레이어 재 접근 시 정상적으로 Action이 수행됩니다.
- 이를 위해 Failure 대신 Success를 반환하여야 SequenceNode에서 올바른 상태 전환이 가능합니다.
- 맵의 구조를 고려하여 위치 탐색에 실패할 경우를 대비 ▶ 탐색 횟수 제한을 두어 유효 위치를 찾지 못하면, 현재 위치에서 IDLE상태로 전환될 수 있도록 ResetPath() 호출 후 Success를 반환합니다.

던전 키우기

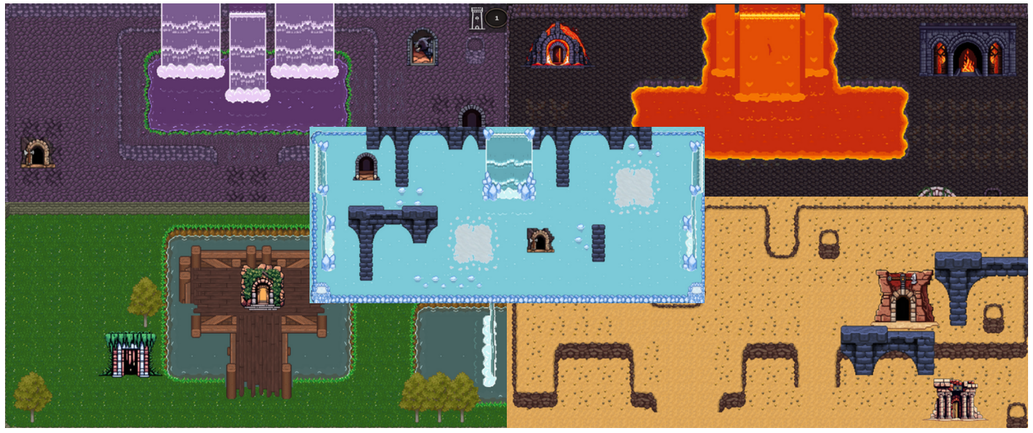
#2D #픽셀 아트 #방치형 RPG #모바일 #Google Play 출시작

Unity | C# | Git

2025.04 출시 예정

GitHub

Google Play에서 다운로드



담당 업무

[클라이언트 개발]

- 데이터 기반 정보 관리 시스템 개발
  - ○ Scriptable Object를 활용한 인 게임 데이터 관리 구현
  - ○ 디자인 패턴 기반 퀘스트 시스템 구현
- 2D 픽셀 아트 컨셉의 전투 시스템 개발
  - ○ Tilemap을 사용한 전투 환경 구현
  - ○ Tilemap 환경 내 유닛 배치 알고리즘 구현
  - ○ 방치형 자동 전투 시스템 구현
- UGUI를 사용한 UI 개발
  - ○ 퀘스트 / 인벤토리 / 대장간 등 게임 내 전체 UI 구현
- 서드파티 SDK 활용
  - ○ 뒤끝(The Backend) SDK를 활용한 비속어 필터링 기능 구현
  - ○ SPUM을 활용한 2D 픽셀 캐릭터 제작
  - ○ DOTween을 사용한 UI 애니메이션 효과 구현

[프로젝트 관리]

- Google Play 출시 전담

Tilemap 환경 내 유닛 배치 알고리즘 구현

방치형 전투를 위한 유닛 스폰 및 자동배치 기능을 구현하였습니다.

Scene 활성화 시 기본 배치

▼

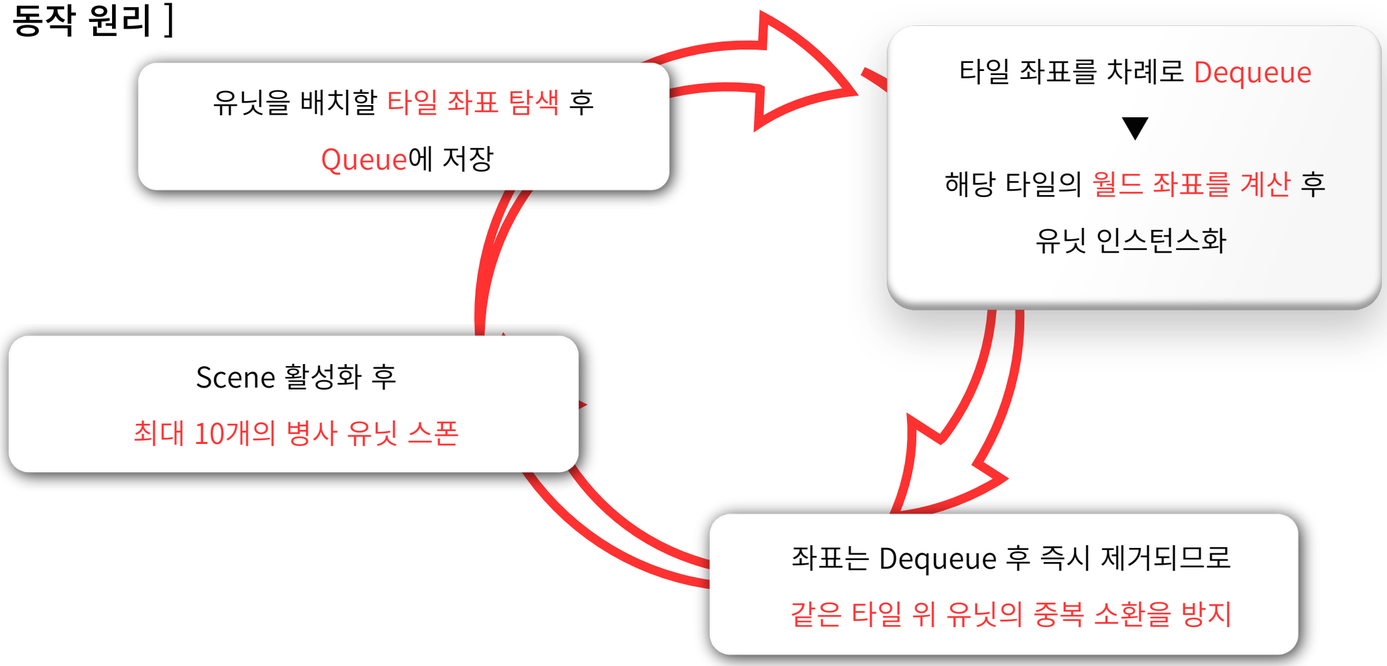
자동 배치 실행 결과

[의도]

- 터치 입력으로 유닛을 수동 / 자동 배치
- 영웅 1명 / 플레이어 1명 / 병사 10명을 사용한 방치형 자동전투
- 자동 배치 클릭 시 공격력이 높은 영웅 유닛을 선두에, 병사 유닛을 후위에 배치
- 타일의 Cell 좌표를 Queue에 넣어 관리하고, 수동 / 자동 배치 Layer를 분리하여 유닛의 좌표 이동 수행

  → 인스턴스 관리 효율성 증가

[동작 원리]



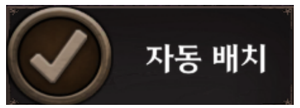
유닛 배치 알고리즘 - 문제 해결 사례

[자동 배치 ↔ 배치 초기화] 반복 시 타일 저장 큐 무한 증가 현상을 해결한 사례입니다.

[문제 파악]

전체 25칸의 타일 중 23칸에 병사 배치 가능 ▶ [자동 배치] 후 유닛을 기본 위치로 되돌리기 위해 [초기화] 클릭 ▶ 유닛이 랜덤한 좌표에 위치하게 되는 오류 발생

자동배치 1회 실행 시



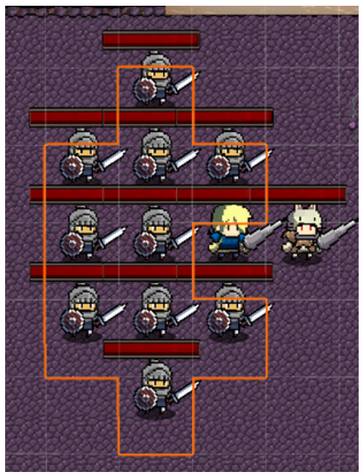

```
[22:52:06] availableTilesForSoldiers.Count = 10
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

자동 배치용 Layer의 좌표 저장

▼

```
[22:52:06] availableTilesForSoldiers.Count = 0
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

해당 좌표를 Dequeue하여 병사 배치 수행



초기화 1회 실행 시


```
[22:52:30] availableTilesForSoldiers.Count = 1
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:52:30] availableTilesForSoldiers.Count = 2
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
.........
[22:52:30] availableTilesForSoldiers.Count = 23
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

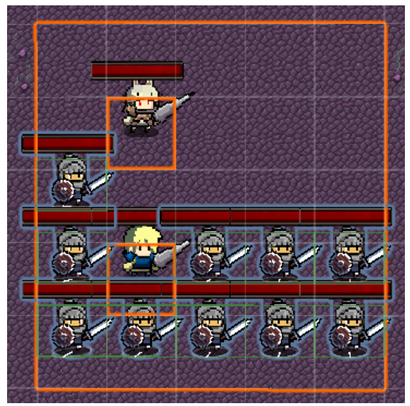
병사 배치 가능한 23칸의 좌표 저장

▼

```
[22:52:30] availableTilesForSoldiers.Count = 13
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

23칸 중 10칸에 병사를 배치.

Queue 내 13개 좌표 잔여.



자동배치 2회 실행 시

```
[22:53:35] availableTilesForSoldiers.Count = 14
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
.........
[22:53:35] availableTilesForSoldiers.Count = 22
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:53:35] availableTilesForSoldiers.Count = 23
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

13개의 좌표가 남아 있는 Queue에

10개 좌표가 더 추가

▼

```
[22:53:35] availableTilesForSoldiers.Count = 13
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

해당 좌표를 Dequeue하여 병사 배치 수행

Queue 내 13개 좌표 잔여

초기화 2회 실행 시

```
[22:54:00] availableTilesForSoldiers.Count = 14
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 15
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
.....
[22:54:00] availableTilesForSoldiers.Count = 36
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] Find Position for AutoPlace Completed.
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 35
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 34
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 33
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 32
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 31
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 30
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 29
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 28
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 27
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] availableTilesForSoldiers.Count = 26
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
[22:54:00] AutoPlace Completed.
0x00007ffb4c4d3ffd (Unity) StackWalker::ShowCallstack
```

13개의 좌표가 남아있는 Queue에

23개 좌표가 더 추가

▼

저장된 좌표가 누적되고 있던 것이 원인.

유닛 배치 알고리즘 - 문제 해결 사례

[자동 배치 ↔ 배치 초기화] 반복 시 타일 저장 큐 무한 증가 현상을 해결한 사례입니다.

[원인 분석]

- 병사 유닛 배치를 위해 할당된 타일 수는 23칸
- 자동 배치 ▶ 스폰 좌표 탐색 과정에서 10개의 좌표가 큐에 추가되고, 이를 Dequeue하여 유닛을 배치.
- 초기화 ▶ 23칸 좌표가 큐에 추가 후 10칸에 병사 유닛 배치. 이 때 큐에는 13개 원소 잔여.
- [자동 배치 ↔ 초기화] 반복 ▶ 13개 좌표가 채워진 큐에 다시 23개 좌표가 Enqueue → 10개 Dequeue 반복
- 결과적으로 초기화가 실행될 때마다 큐 공간이 13개씩 증가.

```
private void FindSoldiersPosition(Tilemap tilemap)//병사 유닛들의 스폰 장소를 찾는다.
{
    //BoundsInt는 주로 타일맵 내의 유효한 cell 영역을 나타내는 데 사용. 각 cell은 하나의 tile을 나타낸다.
    BoundsInt bounds = tilemap.cellBounds;
    foreach(Vector3Int pos in bounds.allPositionsWithin)//자동배치용 병사타일에서 배치 가능한 위치를 찾는다.
    {
        if(tilemap.HasTile(pos))
        {
            availableTilesForSoldiers.Enqueue(pos);//타일목록 큐에 삽입.
            Debug.Log($"availableTilesForSoldiers.Count = {availableTilesForSoldiers.Count}");
        }
    }
    if(availableTilesForSoldiers.Count == 0 )
    {
        Debug.LogWarning("No Available Tiles found in soldierPlaceLayer");
    }
```

BoundsInt 클래스의 동작 원리를 이해하여 해결

Tilemap의 Cell 영역을 나타내는 BoundsInt의 allPositionWithin 메서드

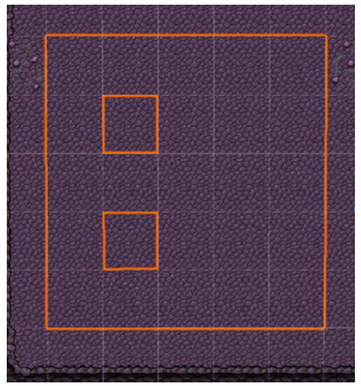
▼

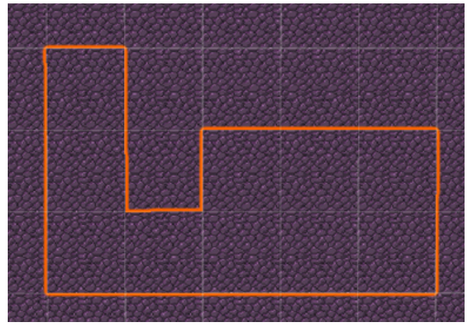
Layer의 좌 하단에서 시작하여 우 상단까지 Vector3Int 좌표를 탐색합니다.

따라서 초기화 스폰 지점은 항상 좌 하단부터 10칸이 됩니다.

▼

좌 하단부터 10칸의 초기화용 Layer를 분리하여 관리하면 해결





기존 Layer를 10칸으로 줄이지 않고 새 Layer를 생성한 이유?

- 총 25칸의 좌표가 모두 저장된 Layer가 존재해야 드래그를 통한 수동 배치가 가능하기 때문입니다.
- 만약 기존 23칸 Layer를 10칸으로 변경할 경우, 드래그 수동 배치가 가능한 Tile은 12칸으로 제한됩니다.

DOTween 라이브러리를 활용한 UI 연출 및 모듈화

서드 파티 라이브러리 함수를 모듈화 하여 개발 편리성을 높였습니다.

DOTween

- Unity에서 오브젝트의 자연스러운 변환을 지원
- 매우 간단하고 직관적인 문법
- Unity에 특화된 퍼포먼스
- 코루틴을 사용하지 않아 GC 발생 감소 가능

UI 호출 시 크기가 점점 커지며 활성화되는 팝업(PopUp) 연출

게임 진행 시간을 제한하는 타이머 기능

DOTween 사용 시, DG.Tweening 라이브러리를 선언하고 초기화, 시퀀스 연결 등 선행 작업이 필요

▼

오브젝트 연출 함수를 모아놓은 정적 클래스를 생성하여 선행 작업 없이 코드 한 줄만 작성하여 사용하도록 개선

▼

```
public static class DOTWeenUIAnimation,
public static void PopupAnimationInUI(GameObject obj, Vector3 beforeEndValue, float beforeDuration, Vector3 aftertAndValue, float afterDuration)
{
    //UI 팝업이 훅 커졌다가 원래 크기로 돌아오는 효과 연출을 위한 메서드. UI의 w, h값이 동일하지 않을 때는, ui의 스케일 값을 Vector3로 매개변수에 넣는다.
    var seq = DOTween.Sequence();
    seq.Append(obj.transform.DOScale(beforeEndValue, beforeDuration));// 훅 커지는 효과
    seq.Append(obj.transform.DOScale(aftertAndValue, afterDuration));// 다시 원상태로 되돌아오는 효과.
    //ui의 로컬스케일이 너무 커서, 훅 커지는 효과를 쓰면 오히려 눈이 아플 경우 ==> 이럴 때는 beforeEndValue를 UI크기보다 작게 줄이고, beforeDuration을 아주 짧게
}
```

▼

```
DOTWeenUIAnimation.PopupAnimationInUI(dutyPanel, dutyPanel.transform.localScale * 0.2f, 0.001f, dutyPanel.transform.localScale, 0.2f);
```

→ 최적화 + 코드 가독성 + 유지 보수성 향상

SoundCheckEditor

Unity 기반 Freesound API 통합 사운드 검색 도구

Unity **C#** **HTTP** **Freesound API**

GitHub

MADE WITH Unity® Asset Store

담당 업무

[클라이언트 개발]

- Unity 에디터 확장 기능 개발
  - ○ 검색어 입력, 검색 결과 목록, 미리듣기 버튼, 페이지네이션 구현
- 외부 API 통합 및 HTTP 통신
  - ○ API 요청 및 응답 처리 기능 구현
  - ○ 사운드 미리듣기를 위한 오디오 스트리밍 기능 구현
- 사용자 경험 개선
  - ○ HTTP 요청 진행도 표시기 구현
  - ○ HttpRequestException 및 JsonException 에러 핸들링

[프로젝트 관리]

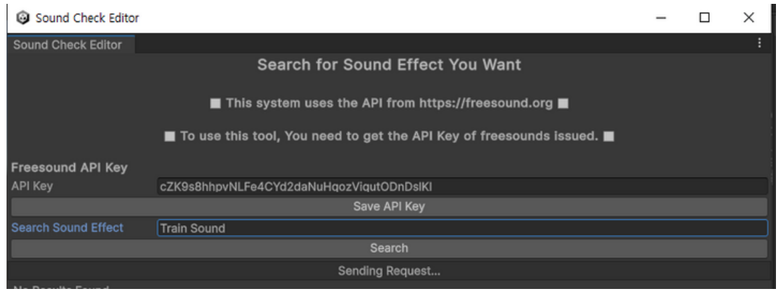
- Unity Asset Store 퍼블리싱

개발 목적

- Unity의 EditorWindow 시스템을 활용하여 개발자 툴 제작 역량을 쌓고자 하였습니다.
- 엔진 바깥에서 사운드 리소스를 탐색하며 불편함을 겪었던 경험에서 아이디어를 얻었습니다.
- 게임 개발 실무에 바로 사용할 수 있는 효율적인 기능 구현을 목표로 하였습니다.

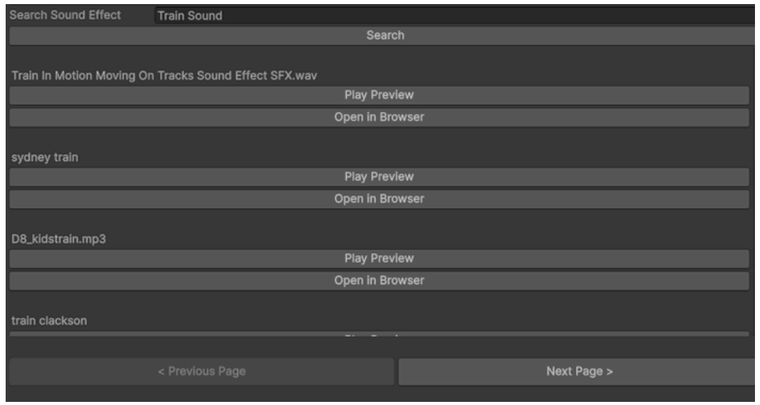
API 요청 및 응답 처리 기능 구현

(1) 사용자 검색어 인코딩 및 요청 전송 기능을 구현하였습니다.



Unity 에디터에서 사운드 검색

검색어 쿼리 인코딩 후 Freesound 서버로 요청




```
searchQuery = EditorGUILayout.TextField("Search Sound Effect", searchQuery);// 검색어 입력

private async void SearchSound(string query)// Freesound의 api키와 검색 쿼리를 사용하여 api요청을 보내 사운드를 검색. 입력한 데이터에 따라 동적으로 URL 생성,호출
{
    CheckApiKeyVaildation();//API KEY 유효성 검사 수행.
    if(string.IsNullOrEmpty(query))//쿼리가 비어있다면면
    {
        EditorUtility.DisplayDialog("Empty Query", "Please enter Query in blank.", "OK");
        return;
    }
    string encodedQuery = HttpUtility.UrlEncode(query);
    /* 쿼리는 URL형태로 전송되므로, 특수문자나 공백이 포함될 경우를 대비한 URL인코딩이 필요.
       C#의 System.Web에 있는 HttpUtility.UrlEncode를 사용. 이를 통해 서버와의 통신 오류, 검색 결과 누락, 보안 문제 등 방지.
       저작권이 자유로운 cc0라이선스를 필터링, api응답에 프리뷰 url을 포함하여 요청하기 위해 프리뷰 파라미터 추가*/

    string url =  $"https://freesound.org/apiv2/search/text/?query={encodedQuery}&filter=license:(\"Creative Commons 0\")&fields=id,name,previews,username";

    var request = new HttpRequestMessage(HttpMethod.Get, url);// HttpRequestMessage 객체 생성 및 인증 헤더 설정
    request.Headers.Add("Authorization", $"Token {apiKey}");
    await FetchSoundData(request); // 주어진 url로 freesound 데이터를 검색하는 메서드. 페이지 이동 시 사용
}
```

[동작 원리]

- OnGUI()에서 API에 HTTP 요청을 보낼 쿼리를 입력 받습니다.
- 쿼리에 특수문자나 공백이 포함되어 있을 경우 오류가 발생할 수 있으므로 HttpUtility.UrlEncode로 인코딩합니다.
- 상업적 이용에 제한이 없는 CC0 라이선스 사운드만 필터링 하여 사운드의 ID, 이름, 미리듣기 소스, 업로더 명을 쿼리와 함께 URL에 삽입합니다.
- HttpRequestMessage 객체를 생성하고 인증 정보를 헤더에 추가한 후, 비동기 요청을 위해 FetchSoundData()에 전달합니다.

API 요청 및 응답 처리 기능 구현

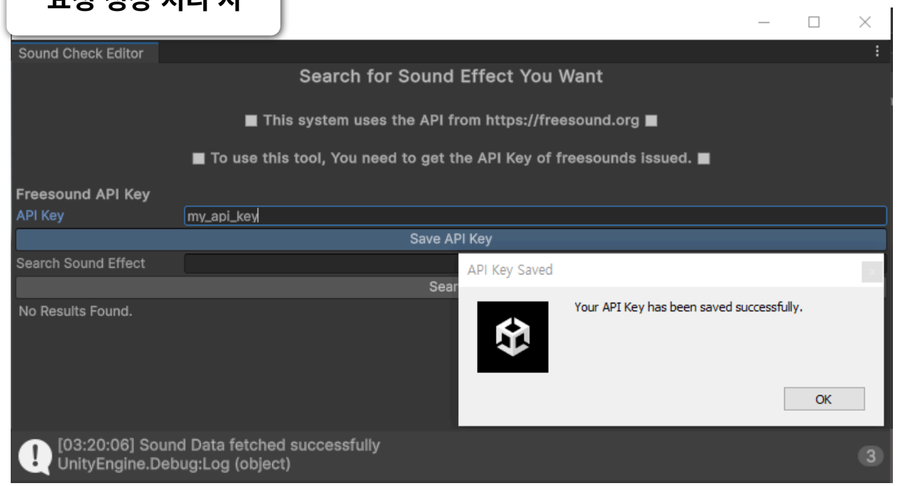
(2) HTTP 요청 전송 및 응답 처리, 에러 핸들링 기능을 구현하였습니다.

```
private void HandleApiError(HttpResponseMessage response)//api키 에러 핸들링을 위한 메서드
{
    string errorMessage = $"API Error : {response.ReasonPhrase}";

    switch (response.StatusCode)//인증오류 시
    {
        case System.Net.HttpStatusCode.Unauthorized:
            errorMessage = "Unauthorized: Your API Key might be invalid or expired.";
            break;
        case System.Net.HttpStatusCode.Forbidden:
            errorMessage = "Forbidden: You do not have permission to access this resource.";
            break;
    }

    Debug.LogError(errorMessage);
    EditorUtility.DisplayDialog("API Error", errorMessage, "OK");
}
private async Task FetchSoundData(HttpRequestMessage request) //주어진 URL로 freesound 데이터를 비동기적으로 가져오는 메서드.
{
    try
    {   isSearching = true;//검색 시작. HTTP 요청 전후로 진행상태를 업데이트.
        progress = 0.0f;
        progressMessage = "Sendig Request...";

        HttpResponseMessage response = await client.SendAsync(request);//비동기적으로 HTTP GET 요청을 보낸 후 응답을 받는다. 생성한 URL을 사용
        progress = 0.5f;
        progressMessage = "Receiving data...";

        if (response.IsSuccessStatusCode) // 요청이 성공했는지 확인
        {
            string jsonResponse = await response.Content.ReadAsStringAsync();//응답의 JSON 문자열을 읽는다.
            progress = 0.8f;
            progressMessage = "Parsing data...";

            soundResults = ParseSoundResults(jsonResponse);//JSON 문자열을 파싱하여 검색 결과 리스트를 업데이트
            progress = 1f;
            progressMessage = "Complete!";
        }
        else
        {
            HandleApiError(response);
        }
    }
    catch (HttpRequestException e)//요청 실패 시
    {
        Debug.LogError($"Error fetching sound data: {e.Message}");
        EditorUtility.DisplayDialog("API Request Error", $"Error fetching sound data: {e.Message}", "OK");
    }
    finally
    {
        isSearching = false; // 검색 종료
        progress = 0f;
        progressMessage = "";
    }
}
```

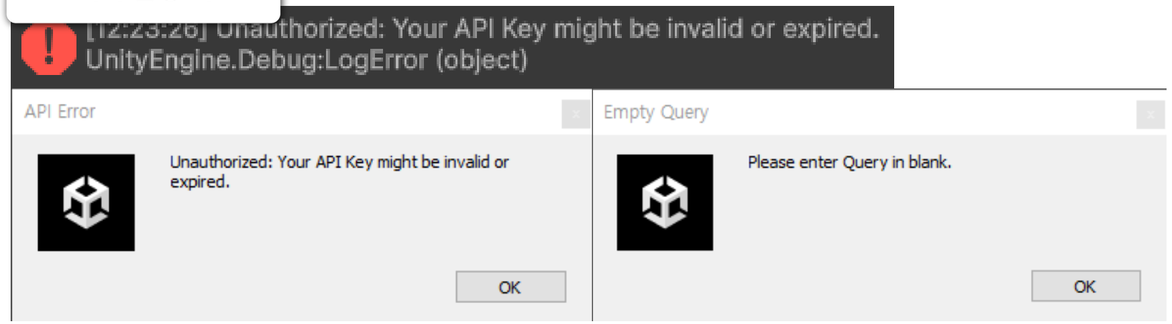
[의도]

- API에 HTTP 요청을 보내고 응답 데이터를 받아와 검색 결과를 처리합니다.
- 비동기적으로 동작하여 에디터 중지 없이 작업을 수행합니다.
- 진행 상태를 실시간으로 업데이트하여 사용자에게 피드백을 제공합니다.

요청 정상 처리 시



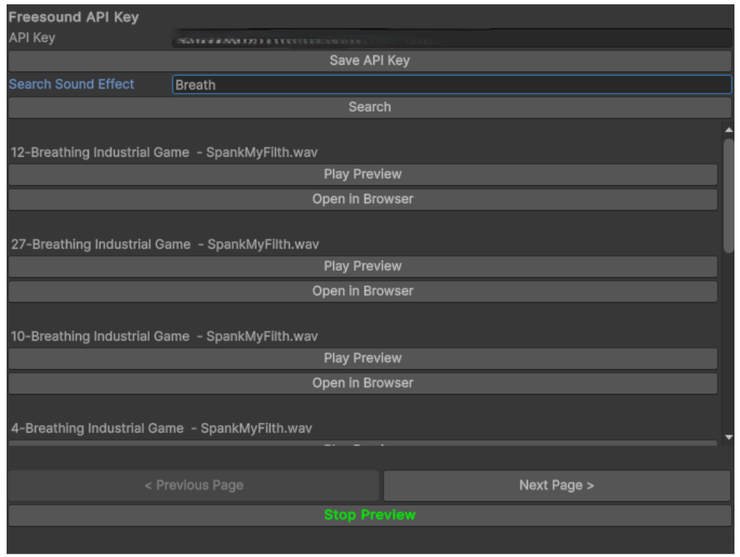
에러 발생 시



API 요청 및 응답 처리 기능 구현

(3) JSON 데이터 파싱 및 검색 결과 처리 기능을 구현하였습니다.

```
private List<SoundResult> ParseSoundResults(string json)//freesound API 응답을 파싱하여 검색 결과를 반환하는 메서드
{
    if (string.IsNullOrEmpty(json))//api 응답 데이터 유효성 검증 --> json파싱 전에 api 응답이 empty인 경우를 방지
    {
        Debug.LogError("API response is empty.");
        EditorUtility.DisplayDialog("API Error", "The API response is empty.", "OK");
        return new List<SoundResult>();
    }
    try
    {
        //JSON 문자열을 FreesoundSearchResult 객체로 변환.
        var searchResults = JsonConvert.DeserializeObject<FreesoundSearchResult>(json);
        if (searchResults == null || searchResults.results == null || searchResults?.results == null)
        {
            Debug.LogError("No search results found or invalid JSON structure.");// 검색 결과가 없거나 json 구조가 유효하지 않을 경우 출력
            return new List<SoundResult>();//빈 리스트를 반환토록 한다.
        }
        //다음페이지와 이전 페이지의 URL 설정
        nextPageUrl = CleanUrl(searchResults.next);
        prevPageUrl = CleanUrl(searchResults.previous);

        List<SoundResult> results = new List<SoundResult>();

        foreach (var sound in searchResults.results)// 검색 결과 리스트를 순회하여 SoundResult 객체로 변환 후 리스트에 추가
        {
            results.Add(new SoundResult
            {
                id = sound.id,
                name = sound.name,
                license = sound.license,
                username = sound.username,
                previews = sound.previews
            });
        }
        return results;//변환된 검색 결과 리스트 반환
    }
```



에디터 내 사운드 재생

검색 결과 페이지 뷰

다운로드 URL 이동

[의도]

- API로부터 받은 JSON 응답을 Unity에서 사용 가능하도록 역직렬화 합니다.
- JSON 데이터에서 사운드 데이터 및 페이지 데이터를 추출합니다.

[동작 원리]

- DeserializeObject()를 사용하여 JSON 문자열을 FreesoundSearchResult 객체로 변환합니다.
- 페이지네이션을 위해 next, previous 필드 추출 후, 불필요한 부분은 제거합니다.
- 검색 결과 리스트를 순회하며 각 사운드 정보를 SoundResult 객체로 변환합니다.

[문제 해결 과정]

- FreesoundSearchResult 클래스는 네스팅된 사용자 정의타입 리스트를 포함하고 있기에 JsonUtility 라이브러리 사용 시 오류가 발생합니다.
- 이를 해결하기 위해 .NET Framework 기반의 NewtonSoft.json 라이브러리를 사용하여 유연한 역직렬화를 수행했습니다.
- 또한 Assembly Definition 파일에 NewtonSoft.json 참조 추가하여 사용자가 라이브러리 추가 없이 바로 사용 가능하도록 조치하였습니다.

```
var searchResults = JsonUtility.FromJson<FreesoundSearchResult>(json);
```

```
[01:24:08] No search results found or invalid JSON structure.
UnityEngine.Debug:LogError (object)
```

```
using Newtonsoft.Json;
var searchResults = JsonConvert.DeserializeObject<FreesoundSearchResult>(json);
```

사운드 미리듣기를 위한 오디오 스트리밍 기능 구현

URL으로 다운로드한 오디오를 에디터에서 재생하는 기능을 구현하였습니다.

```
private async void PlayPreview(string previewUrl)//목록에 출력된 사운드 리소스의 미리듣기 기능을 구현하는 메서드.
{
    CheckApiKeyVaildation();//API 키 유효성 검사
    CheckUrlValidation(previewUrl);//URL 유효성 검사
    try
    {
        if(audioSource.isPlaying)//현재 미리듣기 중일 경우 중지.
        {
            StopPreview();
        }
        //UnityWebRequestMultimedia.GetAudioClip을 사용하여 미리듣기 오디오 클립을 다운로드.
        using(UnityWebRequest www = UnityWebRequestMultimedia.GetAudioClip(previewUrl, audioType:AudioType.MPEG))
        {
            // Freesound API는 미리듣기 요청에도 API 키를 필요로 하므로, 헤더에 API 키를 추가
            www.SetRequestHeader("Authorization", $"Token {apiKey}");

            var operation = www.SendWebRequest();//비동기 오디오클립 다운로드.

            while(!operation.isDone)//다운로드 완료 시 까지 대기.
            {
                await Task.Yield();//비동기 대기.
            }

            if(www.result == UnityWebRequest.Result.ConnectionError || www.result == UnityWebRequest.Result.ProtocolError)
            {
                Debug.LogError($"Error loading audio : {www.error}");// 다운로드 중 오류 발생 시 에러 출력.
                EditorUtility.DisplayDialog("Audio Error", $"Failed to load audio : {www.error}", "OK");
            }
            else//다운로드 성공 시
            {
                currentAudioClip = DownloadHandlerAudioClip.GetContent(www);//다운로드된 클립을 가져온다.

                if(currentAudioClip != null)//오디오 클립이 정상적으로 로드되었다면
                {
                    if(audioSource.isPlaying)// 현재 오디오 재생 중일 경우.
                    {
                        audioSource.Stop();// StopPreview()는 currentAudioClip도 null로 초기화시키므로, 여기서는 호출X
                    }
                    audioSource.clip = currentAudioClip;
                    audioSource.Play();
                }
            }
        }
    }
```

[의도]

- 미리듣기 기능을 Unity 에디터 내에서 구현하여 사용자가 사운드를 미리듣기할 수 있도록 합니다.

[동작 원리]

- 현재 오디오 재생 여부를 확인하여 중복 재생을 방지합니다.
- UnityWebRequestMultimedia.GetAudioClip()을 사용하여 미리듣기 오디오를 다운로드합니다.
- www.SendWebRequest()를 호출하여 다운로드를 시작하고, await Task.Yield()를 사용하여 다운로드 완료 시 까지 비동기적으로 대기합니다.
- 다운로드 성공 시 DownloadHandlerAudioClip.GetContent(www)를 사용하여 오디오 클립을 가져옵니다.