

Survivor

2024. 10. 20 배성훈



게임 설명

- 장르 : RTS 생존 게임
- 개발 인원 : 1인
- 제작 기간 : 2023. 08. 01 ~ 2024. 12. 31

사용한 툴



유니티
2021. 3.11f



Visual Studio 2022
C# 9.0

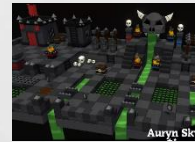


포토샵

에셋



BE5



Dungeon



PolygonFantasy



MeshIntChicken

목차

주요 기법

- 오브젝트 풀링
- 싱글톤 객체
- 상태 패턴

주요 클래스

- Selectable
- Command
- PlayerManager
- Mission
- Minimap
- ActionManager
- UIManager

피드백

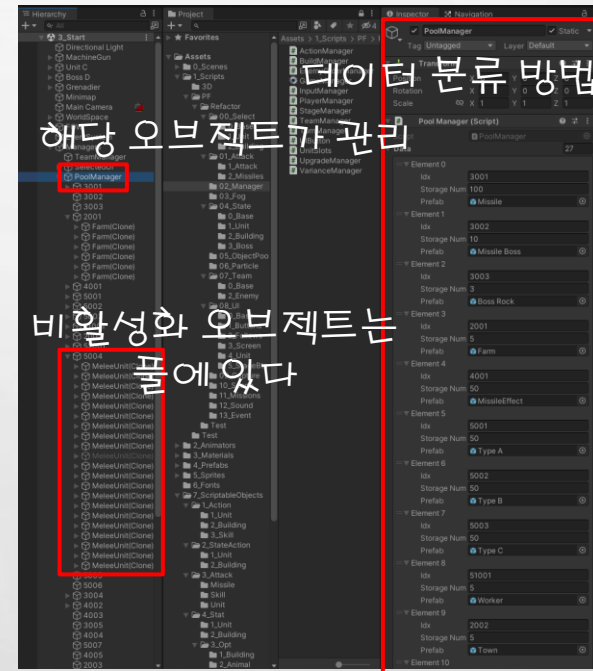
- 최적화
- FogWar
- 클래스 설계

주요 기법

오브젝트 풀링, 싱글톤 객체, 상태 패턴

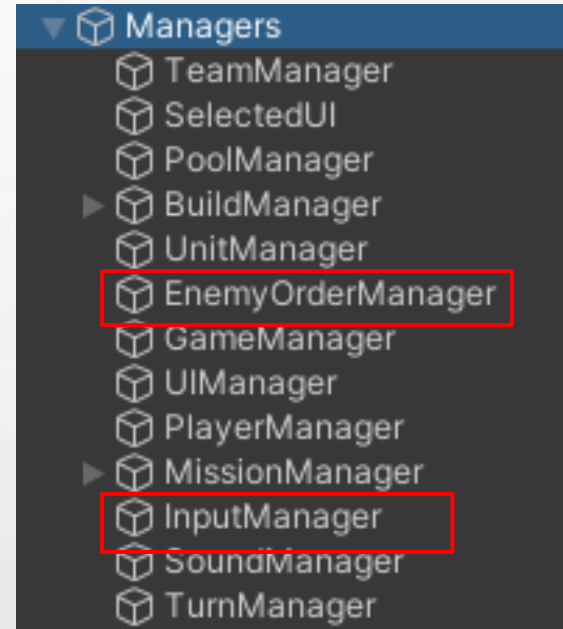
오브젝트 풀링

- 미사일, 유닛, 건물, 명령과 같이 자주 쓰는 클래스는 new 연산이 비싸기에 오브젝트 풀링 적용
- 유닛, 건물, 미사일은 PoolManager에서 관리
- 명령은 Command 클래스의 static 변수로 관리



싱글톤 객체

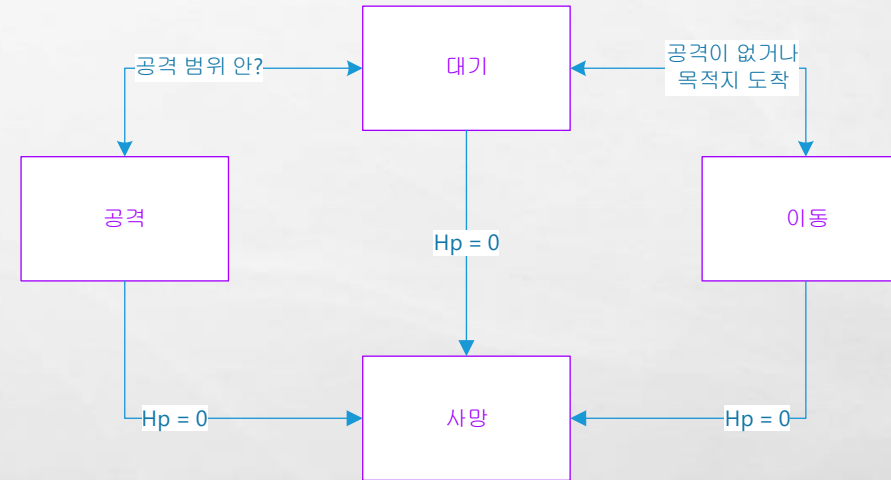
- 유닛을 행동을 일괄 실행시키는
ActionManager처럼 많은 객체가
참조하는 경우 싱글톤 객체
- UIManager처럼 UI 기능들처럼
특정 기능들을 모아 실행해야 하는 경우
싱글톤 객체



다음처럼 복속되거나
참조 필요 없으면
Manager중 싱글톤 선언 안
한 것도 있다

상태 패턴

- 유닛의 상태를 알기 쉽게 관리하기 위해 상태에 맞춰 행동을 진행하는 상태 패턴 적용
- 유닛 뿐만 아니라 입력에 따라 버튼의 적용 및 변환도 상태 패턴으로 적용
- 상태와 행동을 이어주는 것은 enum 타입으로 선언



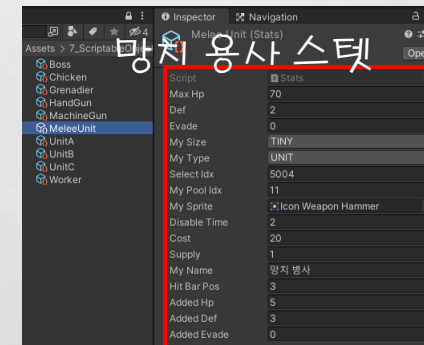
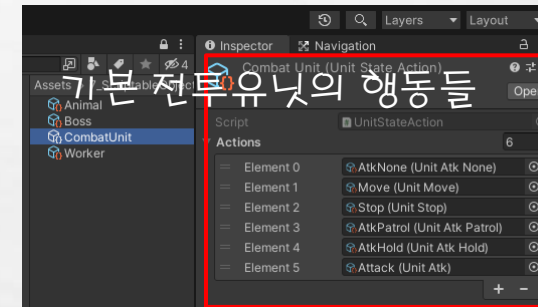
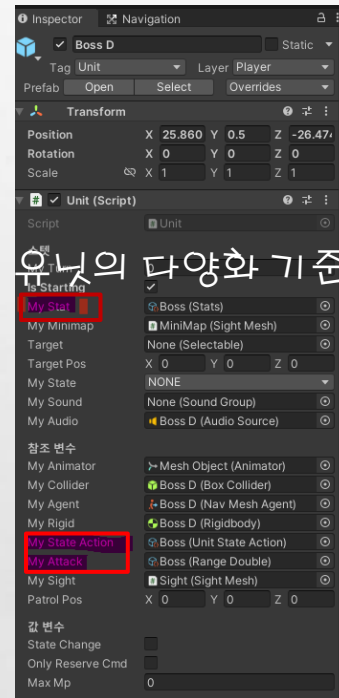
유닛의 자동 행동은
FSM 알고리즘

주요 클래스

Selectable, Command, PlayerManager, Mission, Minimap, ActionManager, UIManager

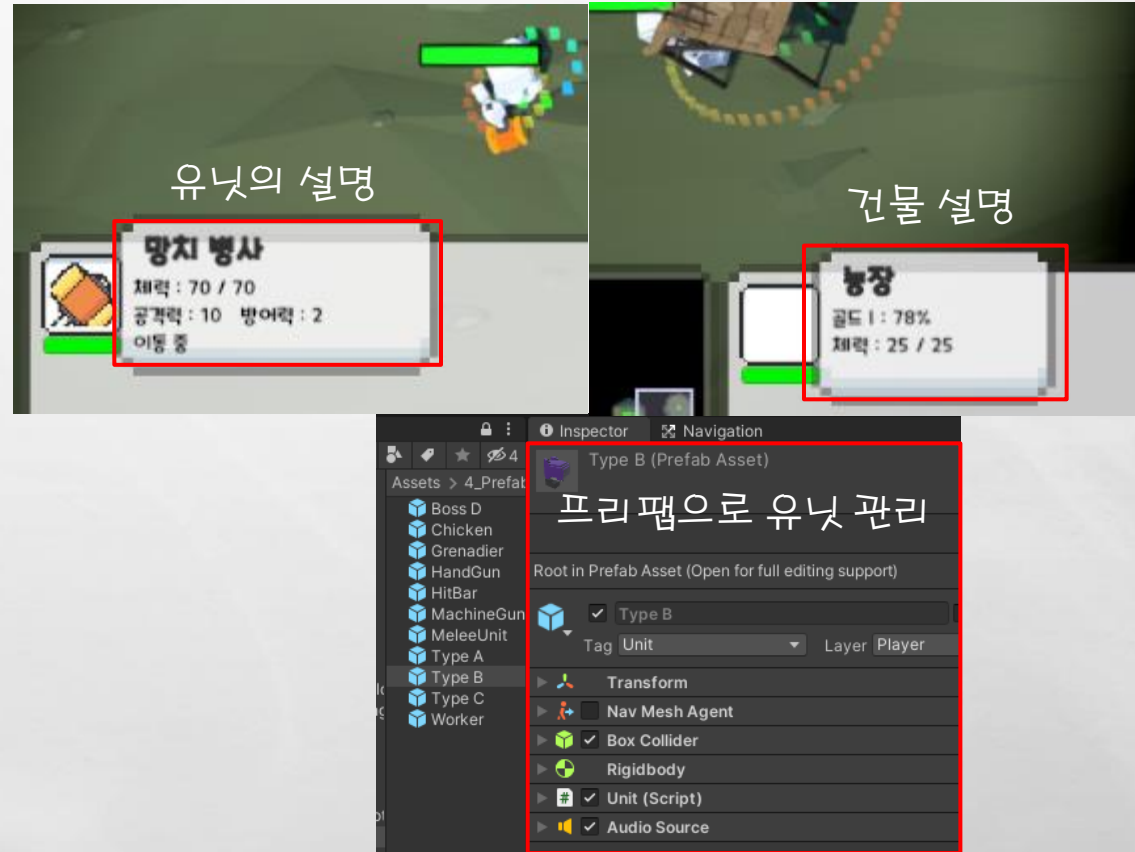
Selectable

- 초기에는 유닛, 건물 상관없이 Selectable 클래스를 이용
- 행동, 공격 방법, 스택으로 다양한 유닛, 건물을 구현할 의도
- 행동은 상태 패턴으로 관리하고 스크립트 오브젝트로 구현



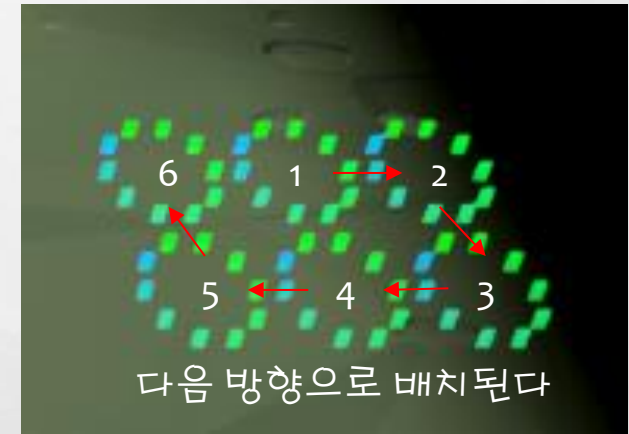
Selectable

- 상태 설명과 같이
유닛과 건물의 차이 존재
- 유닛과 건물을 Prefab으로 관리하고,
외형 3D 오브젝트도 함께 저장하고
Prefab으로 풀링한다
- 유닛과 건물을 각각 구분



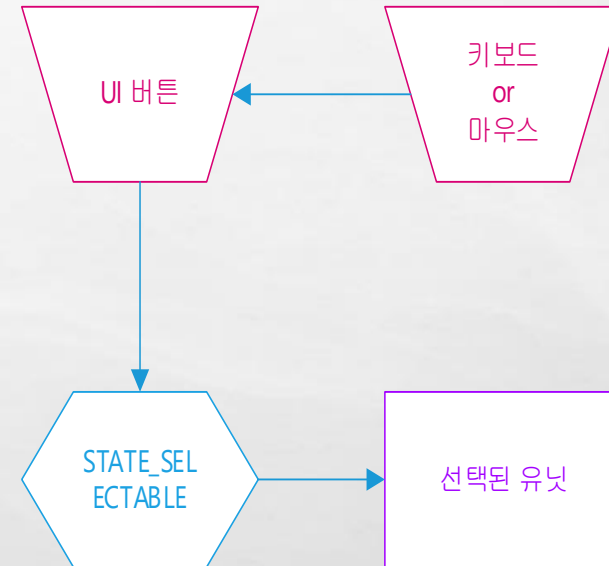
Command

- 예약 명령이 존재해 명령 클래스를 따로 분할
- 다수의 유닛을 명령할 수 있어 한 지점에 이동하라는 명령을 받으면 나선 모양으로 배치 설정



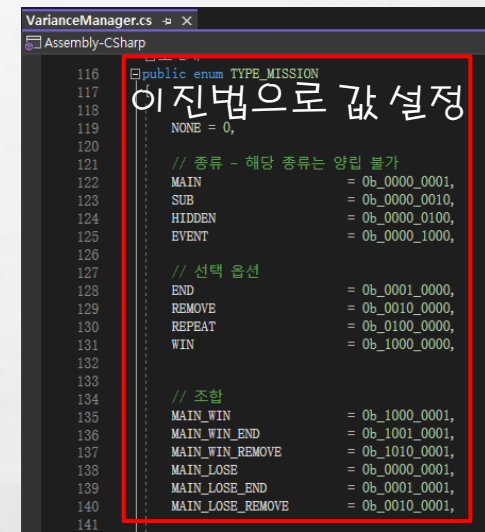
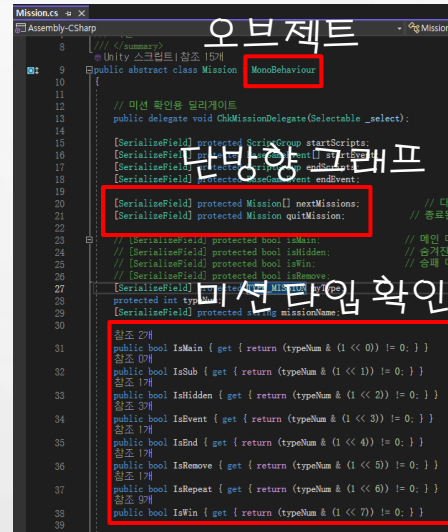
PlayerManager

- PlayerManager 유닛 선택과 선택된 유닛에 명령 주는 클래스
- 선택된 유닛에 따라 버튼이 다르게 표시되어야 하므로 버튼 정보는 PlayerManager에서 관리



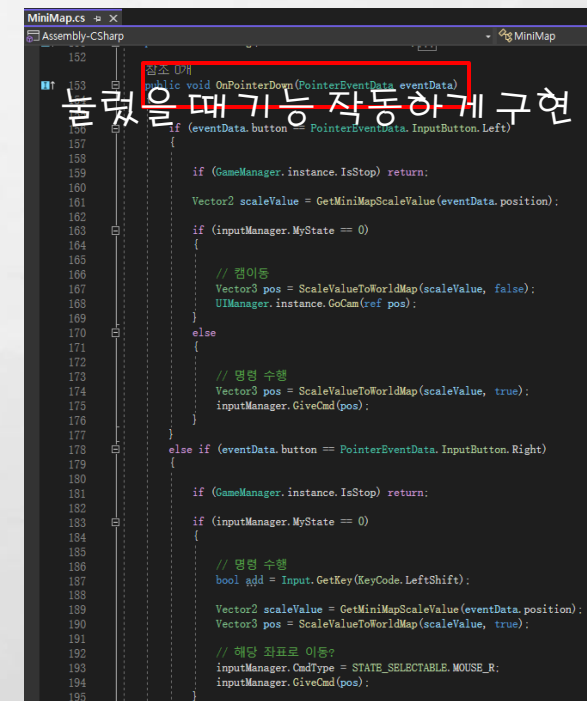
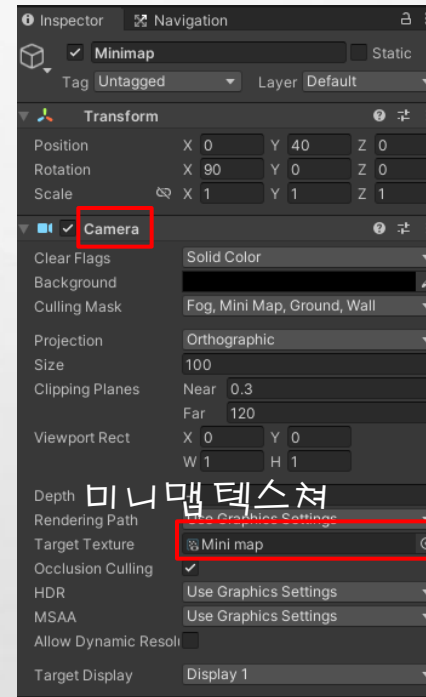
Mission

- 메인, 서브, 히든 미션 구분이나 반복, 단발성 미션 구분 같이 미션의 구분은 비트마asking으로 구분
- 해당 좌표로 이동하는 미션으로 게임 오브젝트를 상속받아 확인
- 미션은 단방향 그래프 형식



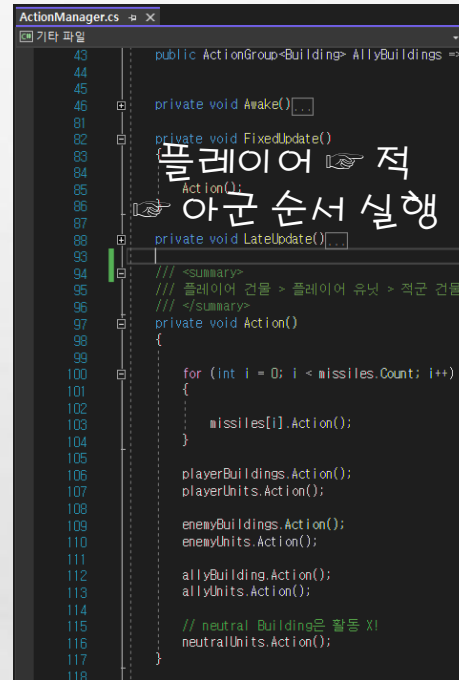
Minimap

- 미니맵은 따로 카메라 텍스처로 구현
- 미니맵 UI는 Image를 이용하고
UnityEngine.EventSystems 안의
IPointerDown 인터페이스 상속으로
누르면 바로 기능 작동하게 구현
- 미니맵을 통한 월드 좌표 계산은
비례식을 이용한 월드 좌표 계산



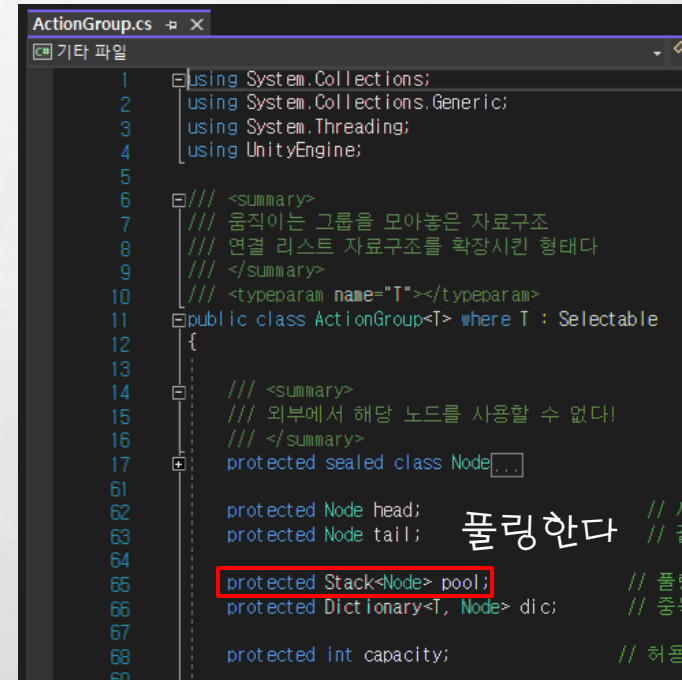
ActionManager

- 유닛 개개인이 FixedUpdate 메서드를 보유하면 느리기에 ActionManager에서 일괄적으로 FixedUpdate 실행
- ActionGroup은 유닛들을 일괄 행동 시키는 자료구조로 초기에는 리스트로 하다가 유닛의 넣고 빼는 경우를 고려해 이중 연결 리스트 형태로 구현



```
43 public ActionGroup<Building> AllyBuildings => a
44
45 private void Awake()
46
47 private void FixedUpdate()
48     Action();
49
50 private void LateUpdate()
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
```

플레이어 적
아군 순서 실행

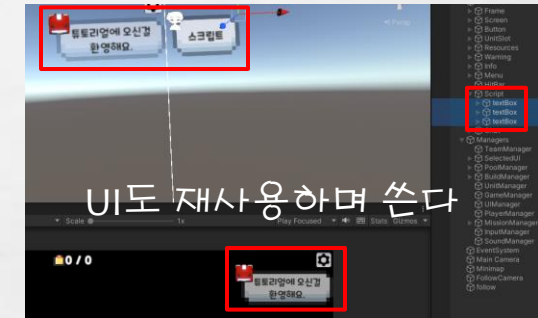
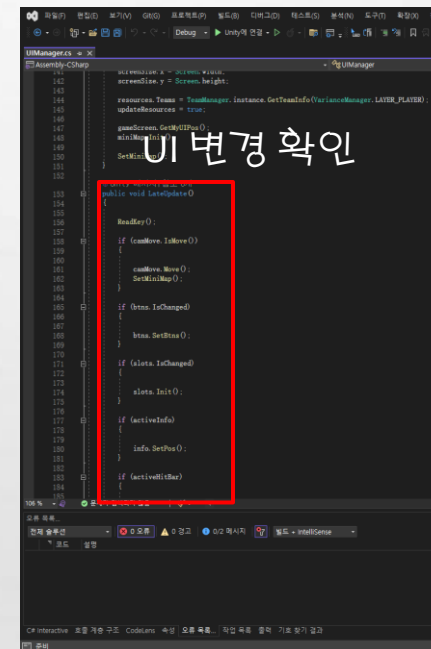


```
1 using System.Collections;
2 using System.Collections.Generic;
3 using System.Threading;
4 using UnityEngine;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

풀링한다

UIManager

- UI 변화가 필요한 부분을 LateUpdate에서 감지하고 변화 시켜주는 클래스
- 화면 이동, 버튼 정보 갱신, 선택 유닛 갱신, 알림 글 등 모든 UI를 관리

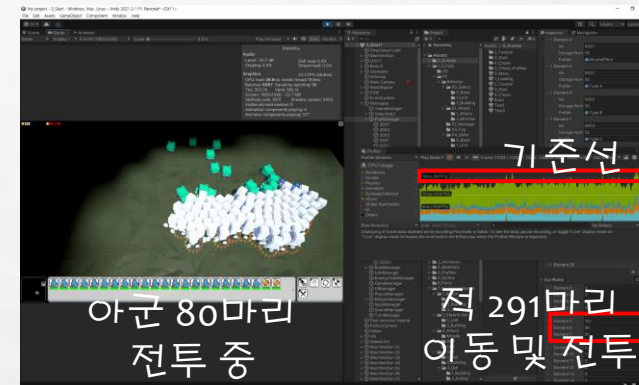


피드백

최적화, FogWar, 클래스의 설계

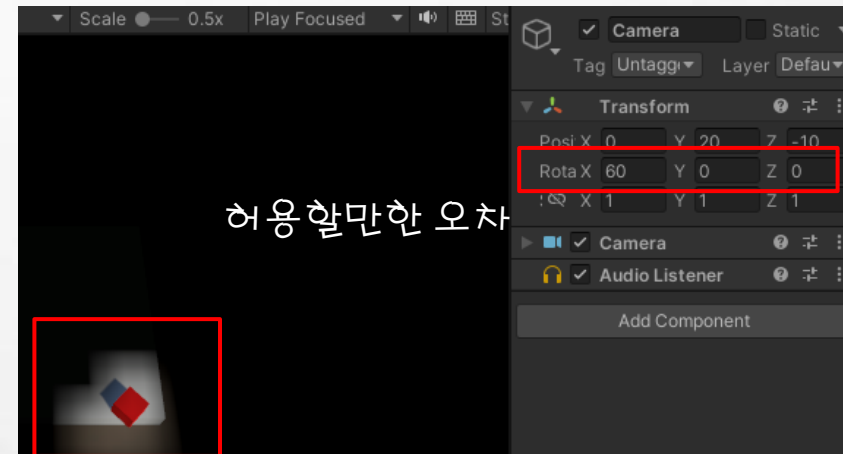
최적화

- 최적화의 기준은 200마리 이상의 유닛을 놓고 프로파일러에서 others를 제외하고 체력바 없이 60FPS 안에 실행되는지 여부
- 클래스들의 재활용 (오브젝트 풀링)
- 필요한 레이어끼리만 물리 충돌 감지
- UI들의 구분 지어 그룹화해서 드로우콜 줄이기



FogWar

- 전장의 안개(FogWar) 부분은
정사영을 기반으로 구현
- 카메라를 이용한 방법과
Shader를 이용한 방법이 있었다
☞ 카메라를 이용한 방법은 기울어질 경우
어색하게 보이는 경우가 존재했다
쉐이더 기초지식을 익혀 쉐이더 방법 적용



클래스의 설계

- 큰 틀만 잡고 코드를 작성했다
기존 InputManager의 경우
PlayerManager의 기능이 있었다
현재는 키 입력 감지만 한다
- 코드의 길이가 길어지고,
유닛의 행동 분석에
시간이 많이 걸렸다

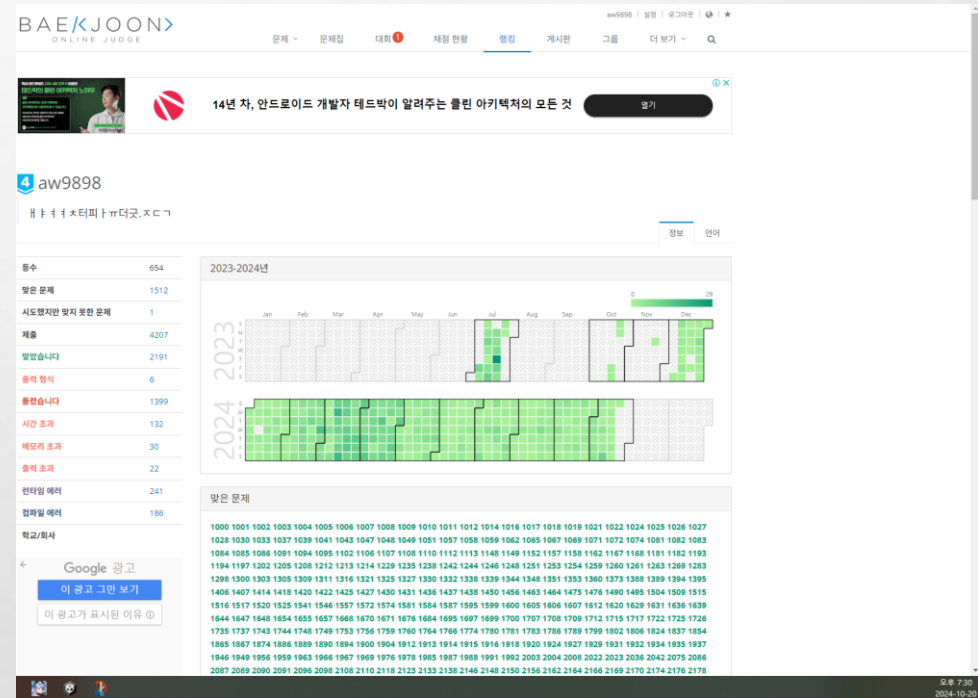
InputManager.cs

```
71 + public int MyState
72 + {
73 +
74 +     set
75 +     {
76 +
77 +         if (curGroup.GetSize() == 0
78 +             || (keys & (1 << value)) == 0)
79 +         {
80 +
81 +             // 유닛이 없거나 입력 받을 수 없으면 @으로 강제 초기화 하고 종료
82 +             myState = STATE_KEY.NONE;
83 +             IsActionUI = true;
84 +             return;
85 +         }
86 +
87 +         // 유닛이 존재하고 입력받을 수 있으므로 상태 변경
88 +         myState = (STATE_KEY) (myState | (1 << value));
89 +         isDrag = false;
90 +
91 +         // 상태에 따른 기능 수행
92 +         if (curGroup.ChkCommand(value))
93 +         {
94 +
95 +             // 좌표가 필요한 경우
96 +             GiveCommand(Input.GetKey(KeyCode.LeftShift));
97 +         }
98 +
99 +         {
100 +
101 +             // 좌표가 필요한 경우
102 +             IsActionUI = false;
103 +         }
104 +     }
105 +     get { return (int)myState; }
106 + }
```

InputManager에서
명령도 전달

클래스의 설계

- 특정 기능들을 하나씩 분할 해보고
비슷한클래스끼리는 묶어주며
클래스를 명확하게 정의해갔다
- 다양한 경험이 클래스를 명확하게
설계하는게 도움이 됨을 알고,
백준, 프로그래머스 알고리즘을 풀며
다양한 상황을 해결해 왔다



감사합니다

튜토리얼 플레이 영상: <https://blog.naver.com/tryingpop/223625795475>

게임 플레이 영상: <https://blog.naver.com/tryingpop/223626557033>