

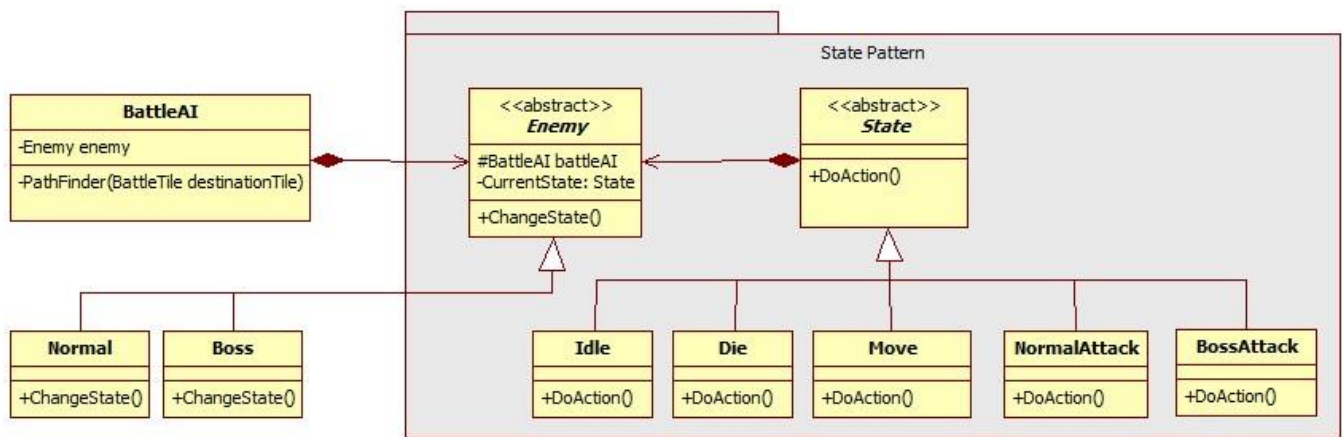
게임프로그래밍 포트폴리오

최대원

회사명	인하대학교 미래인재개발원 문화컨텐츠 프로그래밍 개발 과정		
프로젝트 명	The RedSettler	프로젝트 기간	18.07 ~ 18.08 (2 개월)
프로젝트 상용화	아니오	담당 역할	프레임워크 일부, 몬스터, 몬스터 AI
사용 툴 / 언어	C#, Unity 5.6.5f		
내용	<p>개발 인원 : 9 명 (기획 1 명 + 프로그램 8 명)</p> <p>게임 장르 : 보드 게임</p> <p>게임 소개 : 보드게임 카탄을 기반으로 턴제 보드 게임과 실시간 전투를 혼합하여 전략적인 운영과 실시간 전투 지원, 형상 관리 툴은 Github 와 Sourcetree 사용, 일정 관리 툴은 Jira 사용</p> <p>담당 업무 :</p> <ul style="list-style-type: none"> ● 몬스터와 몬스터의 AI 설계와 구현 ● 프레임워크의 타임 매니저 설계와 구현 		

디자인 패턴 : 상태 패턴(State Pattern)을 응용하여 몬스터로 사용되는 Enemy Class를 설계하였습니다.

상태 패턴은 하나의 클래스가 하나의 상태만을 갖게 하여 각 상태에 따라 런타임 중에 클래스의 행동을 변경하는 패턴입니다.



Enemy : 몬스터들의 추상 클래스입니다. 생성될 때 **BattleAI** 클래스를 생성시켜 자기 자신을 참조할 수 있게 합니다.

BattleAI : 몬스터의 구체적인 행동 제어를 책임지는 클래스입니다.

Boss : **Enemy**의 하위 클래스 중에 보스 몬스터 객체입니다.

Normal : **Enemy**의 하위 클래스 중에 일반 몬스터의 객체입니다.

State : **Enemy**가 가질 수 있는 상태들의 추상 클래스입니다.

포트폴리오에는 **Boss**의 **ChangeState()**와 **BattleAI**의 **PathFinder()**를 첨부했습니다.

ChangeState() : 보스 몬스터의 상태를 바꾸는 메서드입니다. 이 메서드는 BattleAI와 유니티의 애니메이션 이벤트를 통해 호출됩니다.

```
1 //상태를 변경하는 메서드
2 public override void ChangeState(EnemyStateType stateType)
3 {
4     switch (stateType)
5     {
6         //각 상태로 전이되어 행동을 취하기 위해서는 Enemy의 각종 요소들을 참조할 수 있어야 하는데
7         //상태 패턴에서는 각 상태가 캡슐화에 의해 객체를 알 수 없습니다.
8         //객체를 모르는 상황에서 객체의 행동을 제어하기 위해 제어에 필요한 요소들만 생성자를
9         //이용해 넘겨주는 방법을 선택했습니다.
10        case EnemyStateType.Idle:
11            currentState = new Idle(animotor, rigidbodyComponent);
12            break;
13
14        case EnemyStateType.Die:
15            currentState = new Die(animotor, DeadTimer, TimeToReturn, new DeadTimerCallback(EndDead));
16            break;
17
18        case EnemyStateType.Damage:
19            currentState = new Damage(animotor);
20            break;
21
22        case EnemyStateType.Attack1:
23            if (isAttackable[0])
24            {
25                currentState = new Boss.Attack(animotor, bossPhase, fireballLifeTimer,
26                    new TimerCallback(PushFireballTimer), TargetObject, transform, TimeToReturn,
27                    ObjectPoolManager.Instance.FireballQueue, FireBallSpeed, LaunchedFireballList,
28                    (int)Power);
29
30                isAttackable[0] = false;
31                //GameTimeManager : 게임의 시간 및 타이머를 관리하는 싱글톤 클래스입니다.
32                //공격을 연속으로 하지 못하도록 대기 시간을 타이머를 통해 지정했습니다.
33                //타이머 내부 Callback을 통해 시간이 다 되었을 때 호출할 메서드를 지정했습니다.
34                Pattern1Timer = GameTimeManager.Instance.PopTimer();
35                Pattern1Timer.SetTimer(attack1Tick, false);
36                Pattern1Timer.Callback = new TimerCallback(SetAttackable1);
37                Pattern1Timer.StartTimer();
38            }
39            break;
40
41        case EnemyStateType.Attack2:
42            if (isAttackable[1])
43            {
44                currentState = new Boss.UseSkill(animotor, explode, explodeLifeTimer,
45                    Power, explodeLifeTime, new TimerCallback(UseSkillEnd), bossPhase);
```

```

46
47     isAttackable[1] = false;
48     Pattern1Timer = GameManager.Instance.PopTimer();
49     Pattern1Timer.SetTimer(attack2Tick, false);
50     Pattern1Timer.Callback = new TimerCallback(SetAttackable2);
51     Pattern1Timer.StartTimer();
52 }
53 break;
54
55 case EnemyStateType.Move:
56     currentState = new Move(animator, transform, rigidbodyComponent, destinationPoint,
57     currentPoint, MoveSpeed, currentTile);
58     break;
59
60 default:
61     break;
62 }
63 base.ChangeState(stateType);
64 }

```

A*알고리즘 응용 : 길찾기 구현

PathFinder() : BattleAI의 길 찾기 메서드입니다. 육각 타일 기반의 레벨에서 작동할 수 있는 A*알고리즘을 적용했습니다.

```

1  //A* 알고리즘
2  private void PathFinder(BattleTile destinationTile)
3  {
4      //openSet : 이동 가능한 노드들의 후보들이 들어있는 리스트
5      //closedSet : 이동 불가능한 노드들의 후보들이 들어있는 리스트
6      openSet.Clear();
7      closedSet.Clear();
8
9      //시작 타일과 목적지 타일을 설정
10     currnetTile = enemy.GetCurrentTile(enemyPosition);
11     startTile = currnetTile;
12     startTile.ParentTileXCoord = currnetTile.TileCoordinate.x;
13     startTile.ParentTileZCoord = currnetTile.TileCoordinate.z;
14     endTile = destinationTile;
15
16     do
17     {
18         closedSet.Add(currnetTile);
19
20         for (int i = 0; i < coordX.Length; i++)
21         {
22             //인근 타일을 검색합니다.

```

```

23 BattleTile battleTile = SearchAdjacentTiles(i);
24
25 if (battleTile != null)
26 {
27     if (battleTile.isWall)
28     {
29         closedSet.Add(battleTile);
30         continue;
31     }
32
33     //인근 타일의 경로 비용(g)과 휴리스틱(h)을 맨해튼 거리로 계산합니다.
34     battleTile.g = Mathf.Abs(startTile.TileCoordinate.x - battleTile.TileCoordinate.x)
35         + Mathf.Abs(startTile.TileCoordinate.z - battleTile.TileCoordinate.z);
36     battleTile.h = Mathf.Abs(battleTile.TileCoordinate.x - endTile.TileCoordinate.x)
37         + Mathf.Abs(battleTile.TileCoordinate.z - endTile.TileCoordinate.z);
38
39     //해당 타일의 경로 비용이 현재 타일의 경로 비용보다 적은 경우
40     //이 타일이 반대 방향으로 가는 길일수도 있기 때문에
41     //가중치(현재 노드의 g비용)를 더해 줍니다.
42     int tempG = 0;
43     if (battleTile.g < currnetTile.g)
44     {
45         tempG = currnetTile.g;
46     }
47     battleTile.f = battleTile.g + battleTile.h + tempG;
48
49     //인접 타일 중 이동 가능한 후보가 되는 타일들은 오픈셋에 추가합니다.
50     //이때 현재 타일이 이 인접 타일들의 부모 노드가 됩니다.
51     if (!closedSet.Contains(battleTile) && !openSet.Contains(battleTile))
52     {
53         battleTile.ParentTileXCoord = currnetTile.TileCoordinate.x;
54         battleTile.ParentTileZCoord = currnetTile.TileCoordinate.z;
55         openSet.Add(battleTile);
56     }
57
58     //오픈셋에 추가함과 동시에 바로 비용 합계(f)를 기준으로 오름차순 정렬합니다.
59     if (openSet.Count > 1)
60     {
61         openSet.Sort(delegate (BattleTile a, BattleTile b)
62         {
63             if (a.f > b.f) return 1;
64             else if (a.f < b.f) return -1;
65             return 0;
66         });
67     }
68 }
69 }
70

```

```
71 //인접 타일들 모두 탐색했다면 오픈셋에서 가장 적은 비용을 가진 노드를 골라냅니다.
72 //이 노드는 다음 이동 할 타일이 됩니다.
73 if (openSet.Count > 0)
74 {
75     currnetTile = openSet[0];
76     openSet.Remove(currnetTile);
77 }
78 else return;
79 }
80 while (currnetTile != endTile);
81 //목적지에 도착하면 while문이 종료됩니다.
82 //마지막 노드로부터 각 노드의 부모 노드를 따라 최단 경로를 구성합니다.
83 pathTile = CreateParh(startTile);
84 }
```