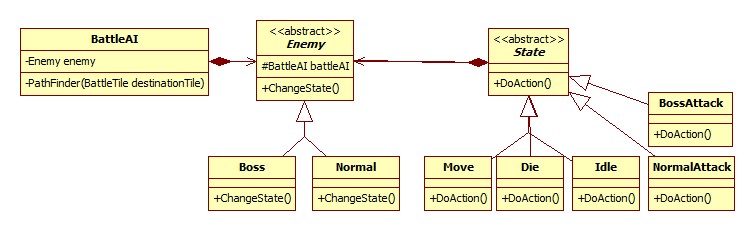
게임프로그래밍 포트폴리오

최대원

몬스터로 사용되는 Enemy 클래스의 구조는 아래와 같습니다.

  
Enemy : 몬스터들의 추상 클래스입니다. 생성될때 BattleAI 클래스를 생성시켜 자기 자신을 참조할 수 있게 합니다.

BattleAI : 몬스터의 구체적인 행동 제어를 책임지는 클래스입니다.

Boss : Enemy의 하위클래스 중에 보스급 몬스터 객체입니다.

Normal : Enemy의 하위클래스 중에 일반 몬스터의 객체입니다.

State : Enemy가 가질 수 있는 상태들의 추상클래스입니다.

포트폴리오에는 Boss의 ChangeState()와 BattleAI의 PathFinder()를 첨부했습니다.

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

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  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 | public override void ChangeState(EnemyStateType stateType)  {      switch (stateType)      {          case EnemyStateType.Idle:              currentState = new Idle(animator, rigidbodyComponent);              break;            case EnemyStateType.Die:              currentState = new Die(animator, DeadTimer, TimeToReturn, new DeadTimerCallback(EndDead));              break;            case EnemyStateType.Damage:              currentState = new Damage(animator);              break;            case EnemyStateType.Attack1:              if (isAttackable[0])              {                  currentState = new Boss.Attack(animator, bossPhase, fireballLifeTimer,                      new TimerCallback(PushFireballTimer), TargetObject, transform, TimeToReturn,                      ObjectPoolManager.Instance.FireballQueue, FireBallSpeed,  LaunchedFireballList,                      (int)Power);                    isAttackable[0] = false;                  Pattern1Timer = GameTimeManager.Instance.PopTimer();                  Pattern1Timer.SetTimer(attack1Tick, false);                  Pattern1Timer.Callback = new TimerCallback(SetAttackable1);                  Pattern1Timer.StartTimer();              }              break;            case EnemyStateType.Attack2:              if (isAttackable[1])              {                  currentState = new Boss.UseSkill(animator, explode, explodeLifeTimer,                      Power, explodeLifeTime, new TimerCallback(UseSkillEnd), bossPhase);                    isAttackable[1] = false;                  Pattern1Timer = GameTimeManager.Instance.PopTimer();                  Pattern1Timer.SetTimer(attack2Tick, false);                  Pattern1Timer.Callback = new TimerCallback(SetAttackable2);                  Pattern1Timer.StartTimer();              }              break;            case EnemyStateType.Move:              currentState = new Move(animator, transform, rigidbodyComponent, destinationPoint,              currentPoint, MoveSpeed, currentTile);              break;            default:              break;          }          base.ChangeState(stateType);  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

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

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  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 | private void PathFinder(BattleTile destinationTile)  {      openSet.Clear();      closedSet.Clear();        currnetTile = enemy.GetCurrentTile(enemyPosition);      startTile = currnetTile;      startTile.ParentTileXCoord = currnetTile.TileCoordinate.x;      startTile.ParentTileZCoord = currnetTile.TileCoordinate.z;      endTile = destinationTile;        do      {          closedSet.Add(currnetTile);            for (int i = 0; i < coordX.Length; i++)          {              BattleTile battleTile = SearchAdjacentTiles(i);                if (battleTile != null)              {                  if (battleTile.isWall)                  {                      closedSet.Add(battleTile);                      continue;                  }                    battleTile.g = Mathf.Abs(startTile.TileCoordinate.x - battleTile.TileCoordinate.x)                       + Mathf.Abs(startTile.TileCoordinate.z - battleTile.TileCoordinate.z);                  battleTile.h = Mathf.Abs(battleTile.TileCoordinate.x - endTile.TileCoordinate.x)                       + Mathf.Abs(battleTile.TileCoordinate.z - endTile.TileCoordinate.z);                    int tempG = 0;                  if (battleTile.g < currnetTile.g)                  {                      tempG = currnetTile.g;                  }                  battleTile.f = battleTile.g + battleTile.h + tempG;                    if (!closedSet.Contains(battleTile) && !openSet.Contains(battleTile))                  {                      battleTile.ParentTileXCoord = currnetTile.TileCoordinate.x;                      battleTile.ParentTileZCoord = currnetTile.TileCoordinate.z;                      openSet.Add(battleTile);                  }                    if (openSet.Count > 1)                  {                      openSet.Sort(delegate (BattleTile a, BattleTile b)                      {                          if (a.f > b.f) return 1;                          else if (a.f < b.f) return -1;                          return 0;                      });                  }              }          }            if (openSet.Count > 0)          {              currnetTile = openSet[0];              openSet.Remove(currnetTile);          }          else return;      }      while (currnetTile != endTile);        pathTile = CreateParh(startTile);  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |