**상태 기계 행위**

상태기계 : 각 행위가 하나의 상태를 가지는 것

상태 그 자체는 부분적으로 하나의 상태 기계만 정의한다. 그래서 상태 기계를 변경하거나 상태 기계 간 전이(transition).하는 방법을 결정하는 것이 중요하다. 각 상태는 진입하거나 빠져나갈 때 특정 행위를 수행한다. (몇 가지의 상태를 갖는지 계획하고 각 상태의 대한 전이를 정의한다.)

상태를 열거형으로 선언한다. (enum) – 간단하지만 확장성이 부족하다(상태가 많아질수록 update, enter, exit 함수가 많아지고 가독성 떨어짐) ( 여러 AI간에 기능을 혼합하고 일치시키는 것도 쉽지 않다.)

상태를 클래스를 사용해서 나타낸다.

상태 진입시 OnEnter, 상태 종료시 OnExit, AIState 인스턴스 포인터의 해시맵을 가져서 상태를 해시 맵에 추가시킨다.

상태 전이시 OnExit로 상태 종료 후 새로운 상태를 찾고 변경한 뒤 OnEnter로 들어간다. 상태를 찾지 못하면 에러메시지

**길찾기**

그래프를 이용( 노드, 엣지, 가중치 가 있다.)

너비 우선 탐색 : 에지에 가중치가 없거나 모든 에지가 양의 같은 가중치를 가질 경우 최단 경로를 찾는 것을 보장한다.

BFS 동안에 각 노드는 직전에 방문한 노드를 알아야 한다.(부모노드)

노드 맵을 큐로 사용하여 BFS를 구현한다.

BFS가 성공하면 outMap의 부모 포인터를 사용해서 경로 재구축이 가능하다.

목표노드의 부모노드 이면 목표에서 한칸, 부모노드의 부모노드이면 목표에서 두칸인 셈이다. 하지만 이는 목표점에서 시작점으로의 경로이다. 스택을 사용해서 반전시킬 수 도 있지만 탐색 자체를 반전시키면 된다.

휴리스틱 : 예상되는 결과를 근사하는 함수이다. (거리의 최소값)

H(x) 는 노드 x에서 목표 노드까지의 추정 비용을 뜻한다.

휴리스틱 함수는 노드 x에서 목표까지의 실제 비용보다 같거나 작으면 항상 허용이다. 실제보다 큰 경우가 있다면 허용되서는 안된다.(사용 x)

사각형 격자에서 휴리스틱을 계산하는 데는 두가지 방법이 있다.

첫번째는 대각선 이동은 유효하지 않다고 가정하는 것이다.

H(x) = |start.x – end.x| + |start.y – end.y|

두번째는 유클리드 거리이다. 표준 거리 공식을 사용한다.(일직선 경로)

H(x) =

두번째 휴리스틱 함수가 더 추천할만 하지만 첫번째가 계산이 효율적이다.

탐욕 최우선 탐색

다음에 어느 노드를 고려해야 하는지 결정하기 위해 휴리스틱 함수를 사용한다.

최단 경로를 보장해주지 않는다.

열린 집합에 있는 휴리스틱 비용이 가장 낮은 노드를 선택 후 인근 노드들을 열린 집합에 추가한 후에 현재 노드를 닫힌 집합으로 이동시킨 뒤 루프를 다시 반복한다.(닫힌 집합에 있는 노드는 조사하지 않는다.)

A\* 탐색(에이 스타)

최적의 경로를 찾기 위해서 몇 가지 조건을 충족해야 한다.

우선 시작점과 목표점 사이에 당연히 경로가 존재해야 한다. 또한 휴리스틱은 허용 가능해야 한다.(실제 비용 초과하면 안된다.) 마지막으로 모든 에지의 가중치는 0과 같거나 커야 한다.

GBFS 처럼 AStarScratch를 정의하지만 float 타입의 mActualFromStar가 차이점이다.

노드를 열린 집합으로 추가할 때 경로비용을 계산해야 하고 최소 비용의 노드를 선택할 때 에도 경로비용과 휴리스틱의 총합의 최소값이 가장 낮은 노드를 선택한다.

데이크스트라 알고리즘(균일비용탐색 이라고 부르기도 한다.)

시작 노드는 있지만 목표노드가 없다. 대신 그래프상의 시작 노드에서 모든 도달 가능한 노드까지의 거리를 계산한다.

길 따라가기

길 찾기 알고리즘이 경로를 생성하면 (일련의 점으로 추상화) AI는 경로를 따라 이동하기만 하면 된다.

기타 그래프 표현

경로 노드 : 게임 세계에서 AI가 이동할 수 있는 위치에 경로 노드를 배치한다.

결점 : AI가 노드 또는 에지의 위치로만 이동할 수 있다.

네비게이션 메시 : 각 블록 노드를 다각형으로 정의하여 몇 개의 다각형으로 게임세계의 전체를 나타낼 수 있다.

**게임 트리**

틱택토와 체스 같은 게임은 플레이어가 2명이며 각 플레이어는 차례를 번갈아가며 진행하고 서로 대결하는 구조이다. 게임은 게임 상태를 유지하는 것이 필요하며, 이 상태는 AI의 의사결정에 영향을 미친다. 이는 게임 트리라는 트리구조로 접근한다.

미니맥스

미니맥스 알고리즘은 2인용 게임 트리를 평가해서 현재 플레이어에 대한 최선의 이동을 결정한다. 미니맥스는 각 플레이어가 자신에게 가장 최선의 선택을 한다고 가정한다.

맥스플레이어(플레이어)는 자신의 점수를 최대화 시키려 하고 민플레이어는 맥스 플레이어의 점수를 최소화 시키는데 집중한다.

불완전 게임 트리 다루기

미니맥스 코드를 수정하면 불완전 게임 트리를 대응하는 것이 가능하다.

알고리즘이 사전에 트리를 생성하는 것이 아니라 실행시간 동안에 트리를 생성해야한다.

AI의 게임트리의 깊이를 제한해야한다.(게임 상태가 마지막이 아니어도 리프 노드처럼 다룬다)

정보에 입각한 결정을 내리기 위해 미니맥스는 종료되지 않은 상태의 품질을 근사화할 휴리스틱 요소가 필요하다. 하지만 이는 근사값일 뿐 정확하지 않다.

알파 베타 가지치기

평가하려는 노드의 수를 줄이는 미니맥스 알고리즘의 최적화 기법이다.

알파는 현재 레벨이나 그 이상에서 맥스플레이어에게 보장하는 최고 점수이다.(상한)

베타는 현재 레벨이나 그이상에서 민 플레이어에게 보장하는 최고 점수다.(하한)