[게임 알고리즘]

1) 좌표계

- 위치를 표현하는데 사용하는 체계

- 데카르트 좌표계

좌표계 축의 방향에 따른 분류

- 왼손 좌표계 : Window 기반(directX)

- 오른손 좌표계 : 수학쪽, 오픈소스진영(OpenGL, 3D Max)

원점 기준에 따른 분류

- 로컬 좌표계 : 자신의 원점 기준으로 한 좌표계

- 월드 좌표계 : 월드의 원점을 기준으로 한 좌표계

- 월드 변환(World Transform) : 오브젝트를 월드에 배치하는 작업(위치, 회전, 스케일 적용)

스크린 좌표계

- 디스플레이 기준

- Unity는 원점을 왼쪽 아래로 삼음

랜더링 파이프라인

- 로컬변환->월드변환->카메라변환->투영변환->레스터라이즈

벡터(Vector)

- 위치와 벡터는 다르다.

- 더하기 : 각 x, y, z를 더해서 새로운 x,y,z,를 만든다.

- 곱하기 : 벡터 \* 벡터, 백터 \* 벡터, 벡터 \* 스칼라

외적(Cross Product): 두 벡터가 만드는 평면에 수직한 벡터를 생성한다.

내적(Dot Product) : 두 벡터의 사이각을 구할 수 있다.

행렬

- 단위행렬 : 다른 행렬에 곱해도 다른 행렬에 변화가 없는 행렬

비트연산

- 비트끼리 계산하는 것

- & : 둘다 1일때만 1, 아니면 0 [특정 비트에 1을 ]

- | : 둘 중 하나면 1이면 1, 아니면 0 [특정 비트에 1을 세팅하기 위해 사용]

- << : 레프트 쉬프트. 비트를 왼쪽으로 지정된 숫자만큼 옮기기, 한번 쉬프트할 때마다 수가 2배가 된다. | ex) 3 << 2 = 12

- >> : 라이트 쉬프트. 비트를 오른쪽으로 지정된 숫자만큼 옮기기, 한번 쉬프트 할 때마다 수가 ½배가 된다.

[다른 나누기 연산과는 달리 비트를 한칸 옮기는것이라 비교적 연산이 빠른편에 속함]

Frustum(사각 절두체)[사각뿔에서 상부 머리를 제거한 형태]

- 카메라가 3D 월드에서 실제로 촬영하는 영역

- 상부 머리의 평면부[near plane], 하부 바닥의 평면부[far plane]

- FOV[Field Of View] : 카메라 화각

Camera

- Position: 카메라의 위치

- LookAt : 카메라가 바라보는 방향

- Up : 카메라의 위쪽 방향

3D모델

- 정점(Vertex) : 점(위치)

- 간선(Edge) : 정점과 정점을 잇는 선

- 삼각형(Polygon) : 정점 3개로 만들어지는 삼각형

- 면(Face) : 삼각형이 모여서 만드는 평면

- 메시(Mesh) : 3D 모델링 하나 (전체 외형)

보간(Interporation)

- 시작값과 도착값과 전체 진행시간 중 어느 정도가 경과했는지를 알면 현재 쥐편, 영양 여햑

값형식(Value Type)

- struct

- int, float, bool

- 스택 메모리에 저장된다.

- 복사를 할 때 깊은 복사가 일어난다. (완전한 사본 생성)

참조형식(Reference Type)

- class

- string 등등

- 힙 메모리에 저장된다.

- 복사를 하라 때는 얕은 복사가 일어난다. (참조만 전달한다.)

Boxin(박싱)

- 값 타입을 참조형식으로 캐스팅하는 작업[가능하면 안되게끔 작업하는 것이 좋다, 성능저하의 가장 큰 원인]

// 배열, 리스트, 큐, 스택,

[arrayList 값형식이든 참조형식이든 가능하다(int, float, class 등등) -> 서술과정은 쉽지만 그만큼 프로그램에 부하를 많이 주게 된다.]

ArrayList.Add(1);

ArrayList.Add(10.0f); 등등

=> List<int> list = new List<int>(); // 제네릭 한 리스트. 특정 데이터 타입만 넣을 수 있는 리스트

자료구조

- 구성요소 : 노드

- [Add : 데이터 추가, Remove : 데이터 제거]

선형 자료구조

- 리스트 : 데이터 부분 + 링크 부분 [Head : 시작 노드, Tail : 마지막 노드, 그 이후 마지막 뒤에는 null]

- 스택 : [따로 노드를 사용하기 보단 배열을 사용] [Top : 데이터가 들어가고 나오는 위치][Push : Top에 데이터를 추가, Pop : Top에서 데이터를 꺼내기]

- 큐 : Circular Queue를 많이 사용하는 편, [Front : 가장 앞에 있는 데이터 위치, Rear : 가장 뒤에 있는 데이터 위치]

[EnQueue: Rear에 데이터를 추가, Dequeue : Rear에 있는 데이터를 꺼내기]

비선형 자료구조

- 트리(Tree) : 나무의 역방향처럼 형상화, 뿌리에서 가지를 쳐 퍼져 나가는 방식[Root : 트리의 시작지점, Leaf, Terminal : 트리의 자식이 없는 노드]

계층구조를 가진다. [Sibling : 같은 부모를 공유하는 것] /1 : n 의 구조를 가진다/(부모 하나당 여러 개의 자식)

[ex) B <- A -> C, A는 B와 C의 부모, A의 자식은 B와C]

- 이진 트리(Binary Tree)

= 자식을 최대 2개까지만 가질 수 있는 트리

- 이진 탐색 트리(Binary Search Tree)

= 탐색에 강점을 가지는 트리.

= 순서가 있는 데이터(키)를 가져야 한다.

= 키 값이 크면 오른쪽, 작으면 왼쪽 자식으로 추가된다.

= 트리의 밸런스가 깨지면 탐색 성능이 떨어진다.

- 힙(Heap)

= 우선순위 적용할 때

= 데이터를 추가할 때 자동으로 정렬된다.

= 최대 힙(Max hep), 최소 힙(Min heap)

= 최대 힙일 때 무조건 부모는 자식보다 크다.

= 최소 힙일 때 무조건 부모는 자식보다 작다.

= 완전 이진 트리의 형태를 가져야 한다.

- 그래프

알고리즘

- 문제를 해결하기 위한 절차.

- 시간복잡도 : 데이터 개수에 따른 처리시간

= 빅 오 표기법 : O(n), O(n^2), O(n\*logN), O(logN)

- 공간복잡도 : 알고리즘 처리에 필요한 메모리 공간 정도.[최근 추새에는 크게 신경쓰지 않음]

탐색 알고리즘

- 데이터를 찾는 알고리즘

- 선형 탐색 알고리즘

- 이진 탐색 트리

정렬 알고리즘

- 데이터를 순서대로 정리하는 알고리즘

해시 알고리즘

- 입력 데이터를 정해진 크기의 유니크한 데이터로 변경해주는 알고리즘

- 해시 결과를 다시 원본 데이터로 만들 수 없어야 한다.

- 해시 충돌 : 서로 다른 입력 값에 대해 같은 해시 값이 나오는 경우

길찾기 알고리즘

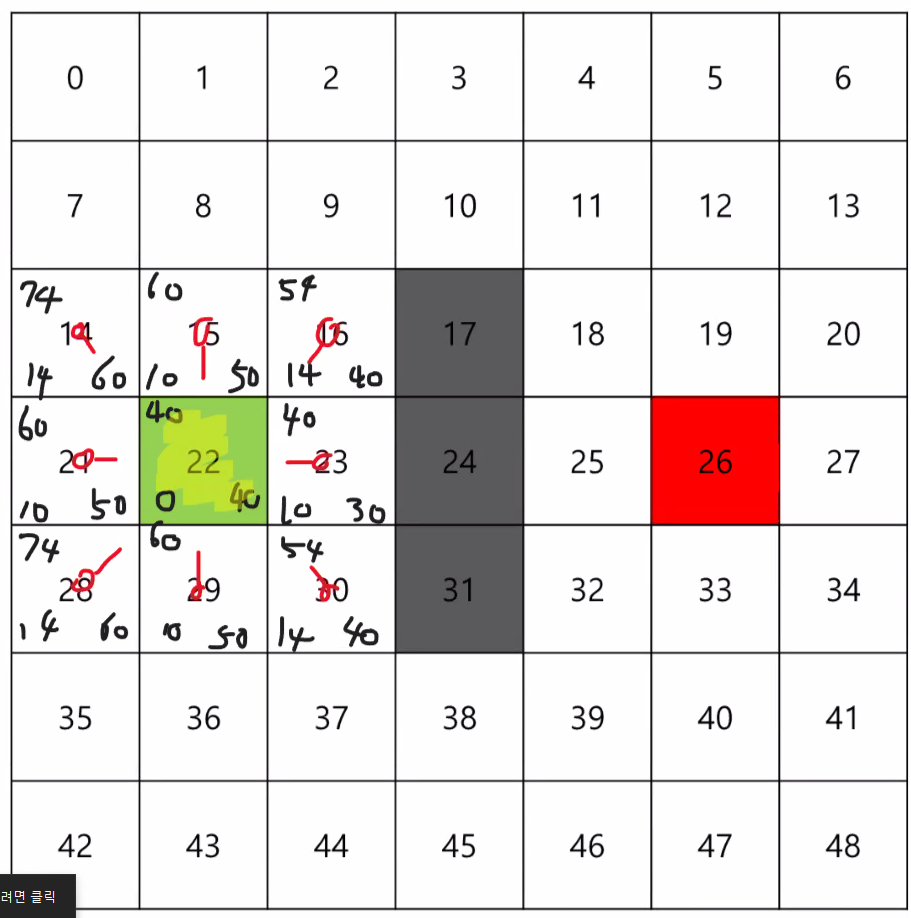
- 다익스트라 알고리즘

- AA^2 알고리즘

F(X) = G(X) + H(X)

G(X) : 목적지까지의 직선 거리(한 칸 당 10)

H(X) : 출발지로부터의 거리 (대각은 14, 직선은 10으로 계산)



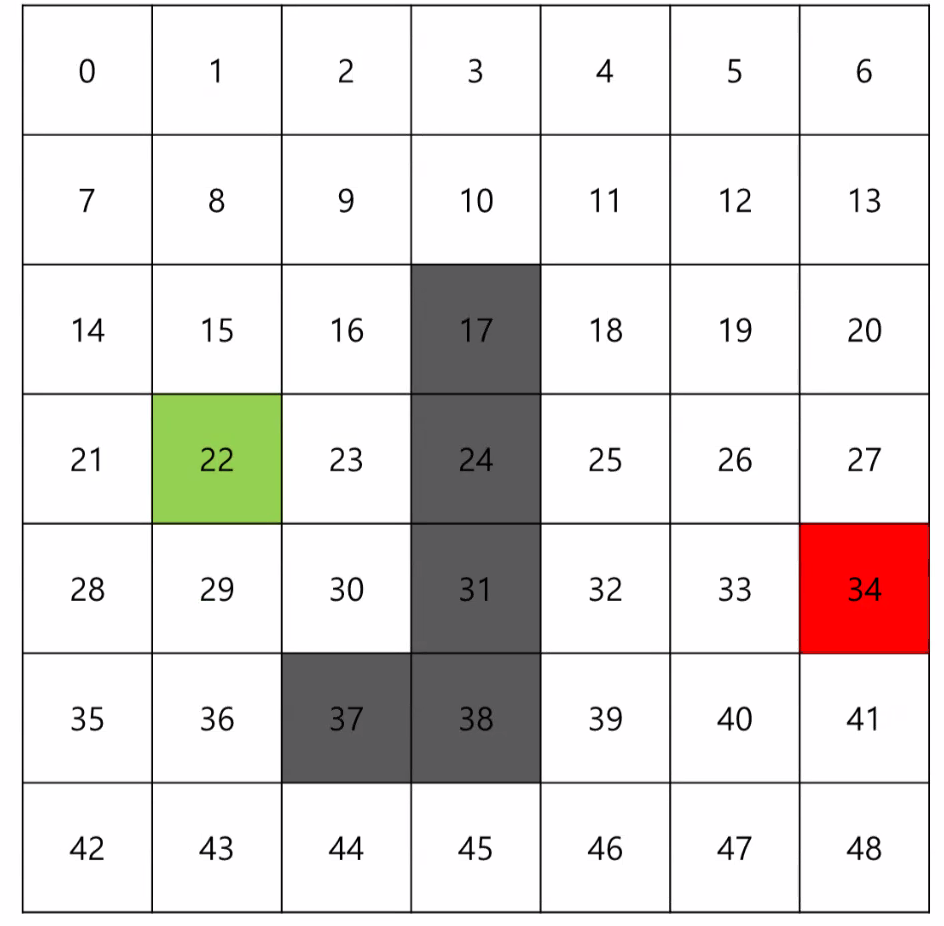
22 에서 26으로 가기 위한 과정

1. 출발지의 f값을 구한 후 close리스트로 넘긴다.
2. close리스트의 들어간 값의 주위에 있는 지점들의 f값들을 구한 후

open리스트에 추가한다.

1. open리스트에 추가된 값 중 가장 작은 값을 선정하여 위의 과정을 반복한다.

(위 과정에서 막힌 부분에 대해서는 open리스트에 포함하지 않는다.)



Open

7[114], 8[108->100], 42[114], 43[100], 1[128], 2[114], 3[108], 4[108], 5[108], 12[94], 18[94], 13[102], 20[88], 25[102], 26[88], 33[92]

Close

22[60], 30[54], 23[60], 29[60], 16[74], 28[74], 15[80], 21[80], 36[80], 9[94], 35[94], 10[94], 14[94], 11[94], 19[88], 27[82], 34[82]