**개미 왕국**

* 환경 : 콘솔 윈도우, C++, 결과 – 콘솔 창 및 txt 출력
* 주제 : 개미들이 태어나고, 살아가고, 죽는 삶을 표현
* 구현할 내용
* 시간
* 1초를 하루로 생각함
* 공간
* 2차원 공간으로 10 x 10 셀을 가짐
* 모든 개체 (개미, 먹이)는 하나의 셀을 가짐
* 개미
* 3등급의 계급을 가짐
* 여왕개미 ( 약 5~10년 )
* 수개미 ( 약 6개월 )
* 일개미 ( 약 1년 )
* 생태
* 태어남
* 태어나면서 기본 생명일이 정해짐
* 여왕개미
* 사망 후 바로 5일 후 태어남
* 랜덤 위치에서 생성
* 이동 없음
* 모든 개미는 여왕 개미 옆에서 태어남
* 공간 3/4 상태인 경우 출산하지 않음
* 수개미
* 5일당 한마리 태어남
* 일개미
* 10일당 한마리 태어남
* 삶
* 식사
* 하루 소요
* 먹은 후 5일 경과시 생명력 1 추가 감소
* 식사시 먹이는 1 감소
* 휴식
* 먹은 후 하루 소요
* 이동
* 가장 가까운 먹이를 찾아 하루에 하나의 셀 이동 가능
* 먹이가 바로 옆에 있는 경우 이동하지 않음
* 죽음
* 10일 이상 굶은 경우 (+- 1)
* 나이가 들어 수명이 다한 경우
* 과다 개체 수
* 공간이 절반이상 찬 경우 하루에 2씩 생명 감소
* 공간이 2/3이상 찬 경우 하루에 3씩 생명 감소
* 먹이
* 크기
* 10 ~ 100 사이 랜덤한 크기를 가짐
* 생성, 소멸
* 크기가 0이 되면 소멸
* ( 10 ~ 15 ) 일 간격으로 생성됨
* 공간 3/4 상태인 경우 생성하지 않음
* 구현할 내용에 오류 또는 개선한 경우 - 오류(개선) 내용과 변경 처리에 대한 설명 문서

1. 시간 구현 :

#include <unistd.h>

...

while(...)

...

sleep(1);

멀티 스레드에서 자주 쓰이는 sleep() 함수를 이용하였다.

2. 공간 구현 :

10 \* 10 의 크기를 가진 2차원 배열을 사용하였다. 빈 공간, 개미, 먹이 등 최소한 세가지를 표현 할 수 있어야 하기 때문에 클래스를 만들어 그 클래스 형식을 가진 배열로 공간을 만들었다.

"cell\_class.h" 에서

property 는 그 오브젝트의 성질을 나타낸다. 0 부터 4까지 순서대로 빈 공간, 여왕개미, 숫개미, 일개미, 먹이를 의미한다.

life 는 개미의 수명을 나타낸다.

hunger 은 개미의 배고픔 수치를 나타낸다.

health 는 개미의 체력, 또는 남은 먹이를 나타낸다.

travel\_date 는 개미가 이동한 날짜를 의미하며 이미 이동한 개미가 여러번 이동하지 않도록 한다.

rest 는 개미가 식사를 하고 쉬는 것을 의미한다.

"cell\_class.cpp" 에서

생성자를 만들었다. property 값을 생성자에 입력하면 그 오브젝트가 가져야 할 성격에 맞게 생성된다.

이를 이용하여 Cell class 오브젝트를 원소로 하는 10 \* 10 크기의 2차원 배열을 만들 수 있었다.

vector< vector<Cell> > kingdom;

3. 개미

1) 3가지 등급

-property 값이 따라 계급이 나뉜다.

-property = 1 : 여왕개미 / 생성자에서 수명은 약 5년~10년으로(1800초 ~ 3600초), 체력은 100~200 사이로 설정된다. "aQ" 로 표현한다.

-property = 2 : 숫개미 / 생성자에서 수명은 약 6개월으로(180초), 체력은 10~20 사이로 설정 된다. "aM" 으로 표현한다

-property = 3 : 일개미 / 생성자에서 수명은 약 1년으로(360초), 체력은 25~50 사이로 설정된 다. "aW" 로 표현한다.

"cell\_class.cpp" 에서

Cell::Cell(int property\_value)

{

srand(time(nullptr));

if(property\_value==0)//empty

{

life=0;

health=0;

}

else if(property\_value==1)//Queen

{

life=1800+rand()%1800;

health=100+rand()%100;

}

else if(property\_value==2)//male

{

life=180;

health=10+rand()%10;

}

else if(property\_value==3)//worker

{

life=360;

health=25+rand()%25;

}

......

}

2) 생태

-여왕개미

main.cpp 의 main() 에서

여왕이 이미 존재하는지를 판별한다.

여왕이 부재중이라면, 여왕이 나타날 시기가 되었는지를 판별한다.

새로운 여왕을 등극할 때가 되었고, 왕국이 포화상태가 아니라면, 왕국 내 무작위 빈 공간에 여왕이 나타난다. 그 후, 여왕의 위치를 기억하고 여왕이 존재함을 기억하고, 왕국에 한 자리를 차지 했음을 저장한다.

주석으로 표기하였음

if(!queen\_ext)

{

......

}

-숫개미

숫개미가 태어날 시기가 되었는지를 판별한다.

탄생 시기가 되었고, 왕국이 포화상태가 아니며, 여왕이 존재한다면

여왕개미가 있는 자리 바로 주위 빈 자리에서 탄생하고, 왕국에 한 자리를 차지 했음을 저장한다.

-일개미

일개미가 태어날 시기가 되었는지를 판별한다.

탄생 시기가 되었고, 왕국이 포화상태가 아니며, 여왕이 존재한다면

여왕개미가 있는 자리 바로 주위 빈 자리에서 탄생하고, 왕국에 한 자리를 차지 했음을 저장한다.

매일(초) 마다 모든 배열을 순차적으로 읽으며 해당 원소에 따라 식사, 휴식, 이동, 죽음, 체력 계산 등이 이루어 진다.

개미들은 공통적으로 우선 수명이 다하거나, 체력이 0 이 되면 사망한다. 생성자에 0 을 입력하는 것으로 원소를 초기화 하여 죽음을 의미한다. 그리고 왕국에 한 자리가 비었음을 저장한다.

아직 살아있다면, 식사를 하고 휴식을 완료한 상태인지 판별한다. 휴식이 끝났다면 자기 주위로 1칸 거리 내에 먹이가 있는지 보고, 있다면 먹이를 먹으며, 없다면 배고픔이 1 증가한다. 이때, 섭취한 먹이가 모두 떨어지면 먹이가 있던 원소의 위치 또한 초기화되어 빈 칸이 된다. 이후 먹이를 먹었다면 소화를 해야하는 오브젝트라고 객체 내부에 저장한다. 먹이를 먹지 않았고, 배고픔 수치가 일정 이상이 되면 체력이 추가로 감소하거나 사망한다. 추가로, 왕국이 포화상태라면 추가로 체력이 감소한다.

여왕의 경우에는 추가적으로 죽음을 판별할때 여왕의 존재여부 또한 갱신되며, 숫개미와 일개미는 주위에 먹이가 없는 경우 가장 가까운 먹이를 찾아 이동하게 되고, 자세한 내용은 후술한다.

3) 먹이

생성자에 4 를 입력하면 먹이가 생성되며, 10~100 사이 무작위 재고를 가지고, 10~15일 주기로 생성되며, 포화상태에는 생성되지 않는다.

4. 개미의 이동

개미의 가장 가까운 거리의 최단거리 이동을 구현하기 위해 A\* path finding 을 구현하였다.

A\* path finding 이란

(출발점부터 방문한 위치까지의 거리) + (방문한 위치부터 도착지점까지 예상 거리) 를 판별하여 가장 작은 값을 가진 방문장소 들을 따라 이동하는 방법이다.

이때, (출발점부터 방문한 위치까지의 거리)를 gcost 라고 하고,

(방문한 위치부터 도착지점까지 예상 거리) 를 hcost 라고 하고,

gcost + hcost = fcost 라고 부르겠다.

"path\_finder.cpp" 의 A\_star\_pathfind() 에서

방문 가능한 공간과 불가능한 공간을 표현한 2차원 행렬과, 출발점 도착점 위치를 저장한 1차원 행렬을 입력받는다.

2차원 행렬의 각 원소들을 알맞게 표현하기 위해 가져야 할 정보가 있다.

1. 방문 여부

2. 출발점에서 방문지점까지 거리

3. 방문지점에서 도착점까지 거리

4. 현재 위치

5. 부모 방문지점 위치

이며, 이들을 "vector<int> col;" 에 입력하여 "vector< vector<int> > neighbors;" 에 저장한다.

위 정보들에 접근하기 쉽도록 "map< vector<int>, vector<int> > info;" 튜플을 만들었다. 현재 위치를 입력하면 위 정보들을 얻을 수 있다.

방문지점이 도착지점과 같아질때 까지 반복한다.

현재 방문지점의 모든 주위, 즉 이웃들을 조사한다. 이웃이 행렬 범위 내에 존재하고, 방문 가능한 지점이라면

처음 조사하는 이웃이라면, 이웃의 위 정보를 조사하여 저장한다.

이미 조사한 이웃이고, 출발지점으로부터 이웃까지 이동한 거리가 방문점에서 이웃으로 가는 거리보다 큰 경우

그 이웃의 gcost 를 수정하고, 부모를 방문지점으로 갱신한다.

조사한 이웃들의 fcost 에 따라 오름차순으로 정렬한다.

방문하지 않은 이웃들중에 가장 작은 fcost 를 찾는다.

만약 모든 이웃을 방문했다면, 출발점에서 도착점에 도달 할 방법이 없다는 뜻이므로, 경로 없음을 반환한다.

그 fcost 를 가진 이웃중에 가장 작은 hcost 를 가진 이웃을 찾는다.

그 이웃에게 방문하고, 반복문의 처음으로 되돌아간다.

방문지점이 도착지점과 같아져서 길 찾기를 완료 했다면,

튜플을 이용해 모든 방문지점들이 가진 부모 지점을 찾을 수 있다.

부모지점이 출발지점이 될 때까지 반복하며 경로에 대한 정보를 저장한다.

구한 경로를 반환한다.

5. 느낀점 및 개선할 점

스스로 길 찾기에 약해서 시작하게 된 첫번째 프로젝트이다. 여러 길 찾기 알고리즘 중에 A\*pathfinding 을 선택 한 이유는 내 능력으로 구현하기 가장 적합한 이론이라 판단했기 때문이다.

이론 자체는 간단했다. 처음에는 hyuristic cost, 즉 예상 거리의 의미를 이해하기 힘들었지만, 말 그대로 장애물의 영향 없이 방문지점으로부터 도착지점까지의 거리라는 것을 알고 구현에 들어갈 수 있었다.

이웃들이 가져야 할 정보를 저장하는데에는 2차원 배열을 사용하였다. 보편적으로 que, stack, linked list 등을 사용한다고 했지만, 내게 가장 익숙한 방법을 사용하게 되었다.

튜플이 정말 큰 역할을 해주었다. 덕분에 이웃들의 정보에 쉽게 접근하게 되어 이번 프로젝트를 완성 할 수 있게 하였다.

분명 이것보다 더 좋은 방법이 있었는지도 모른다. 특히 이 개미왕국의 크기가 겨우 10 \* 10 크기여서 이웃 찾기, 가장 가까운 먹이 찾기 등에 O(n^2) 의 방법을 타협하여 사용하기도 하였다. 이후 이 프로젝트를 보완한다면 속도 방면에서 개선해 보고 싶다.

개미왕국의 여러 오류들을 고쳐나가면서, 내가 사용하려고 만든 변수들 관리를 꼼꼼히 해야 한다는 것을 느꼈다. 고려해야할 변수가 많아지면서 까먹는 경우가 수없이 많이 생겼고, 이로인해 정말 예상 할 수 없었던 오류들을 경험 할 수 있었다.

마지막으로 개미왕국에 남은 오류가 하나 있는데, 개미들이 가끔 먹이를 못 찾아가는 경우가 있다는 것이다. 아예 못찾는게 아니고, 며칠 지나고 나서 뒤늦게 먹이를 찾으러 가는 현상이었다. 움직이지 못하는 경우로는 경로가 불가능하거나, 소화시킬때 뿐인데 어째서 뒤늦게 움직이는지 파악 할 수 없었다. 이후에 원인을 알게 된다면 이를 추가로 해결 하고 싶다.