텍스트이(가) 표시된 사진

자동 생성된 설명

**CONTENTS**

1. 프로젝트 분석
2. 작업 환경
3. 분석 설계
4. 결과 및 고찰
5. 역할 분담
6. 프로젝트 소감
7. **프로젝트 분석**

* Agent 설정
* Default heading : E
* Default action : Go Forward
* 보유 화살 개수 : 3
* 죽은 경우 : (1,1)에서 재시작. (이전까지 경험한 map에 대한 state 유지)
* 성격 : Reflex agent with state
* Map 설정
* Agent Map, RealWorld Map
* 크기 : 6x6 (각 모서리는 Wall 설정)
* safe zone : (1,1), (1,2), (2,1)
* Wumpus, Pit 생성 : 15% 생성확률, random 생성
* 금 위치 : (3,4)
* Agent Map은 모든 칸이 unknown. RealWorld Map에서 Agent 위치에서의 인식되는 사항들을 Agent Map으로 가져옴
* Percept
* Stench (Wumpus 주변 악취)
* Breeze (Pit 주변 바람)
* Glitter (금의 반짝임)
* Bump (벽에 부딪힘)
* Scream (헤딩방향에 존재하는 Wumpus 퇴치)

|  |  |
| --- | --- |
| 센서를 통해 받아들인 Percept | Percept에 대한 Reasoning |
| Stench | 주변에 Wumpus가 있음  Agent의 헤딩방향에 Wumpus의 존재를  Agent가 경험을 통해 알고 있는지 확인 |
| Breeze | 주변에 Pit가 있음  Agent의 헤딩방향에 Pit의 존재를  Agent가 경험을 통해 알고 있는지 확인 |
| Glitter | Agent의 현재 위치에 금이 있음 |
| Bump | 헤딩방향에 벽이 있음 |
| Scream | Agent의 헤딩방향에 있는  Wumpus를 퇴치함  (Agent의 헤딩방향에 더 이상  Wumpus가 존재하지 않음) |

* Reasoning

|  |  |  |
| --- | --- | --- |
| Percept | Reasoning 결과 | Action |
| Stench | Agent가 헤딩방향에  Wumpus 존재를 알고 있음 /  모르고 있음 | 알고 있는 경우 :  헤딩방향으로 Shoot  모르고 있는 경우 : Go Forward |
| Breeze | Agent가 헤딩방향에  Pit 존재를 알고 있음 / 모르고 있음 | 알고 있는 경우 :  Turn Left (Default설정)  모르고 있는 경우 : Go Forward |
| Glitter | 현재 위치에 금이 있음 | 금을 Grab |
| Bump | 헤딩방향에 벽이 있음 | Turn Left (Default설정) |
| Scream | Agent의 헤딩방향에 있는  Wumpus를 퇴치함 | Agent가 알고 있는 Map상황에서  현재 헤딩방향에  Wumpus 존재하지 않음 |

* Action : 헤딩방향으로 전진, 좌측으로 방향 변경, 우측으로 방향 변경, 금 잡기, 화살 쏘기, 금을 획득하고 처음 위치로 돌아와 동굴 빠져나가기의 행동이 있다.

1. **작업 환경**

* Visual Studio 2019
* 사용 언어 : C언어

1. **분석 설계**

* **1)프로그램 구현 설계**

프로그램 로직을 최대한 Reflex agent with state에 맞게 설계를 하려 했다.

1. **환경 인식**
2. **State 업데이트**
3. **액션**

크게 이 세과정으로 나누어 보았다.

에이전트가 현재 환경을 인식하고, 인식한 것을 토대로 State를 업데이트해 유지해 나가고, 액션을 하는 것이다. 프로그램을 구현할 때 최대한 이 세과정에 맞게 구현을 했다..

* **2)구현 코드 설명**
* Agent 설정
* struct agent\_state

텍스트이(가) 표시된 사진

자동 생성된 설명

에이전트가 인식하고 있는 그리드 맵, 센서로 받은 percept 내용, 현재위치 Heading, 마지막 액션, 화살 개수, 금 보유여부, 생존여부를 구조체로 만들어 관리하기 최대한 수월하게 만들어보았다.

* Map 설정
* struct WorldEnviron



Map에서 일어날 수 있는 모든 환경, 인지되는 사항을 모두 구조체로 표현했다.

열거형으로 정의를 하려다, 구조체에서 true or false 로 나타내는게 코드를 구현하는데 간편할거라 판단했다.

* Map Set함수들



텍스트이(가) 표시된 사진

자동 생성된 설명

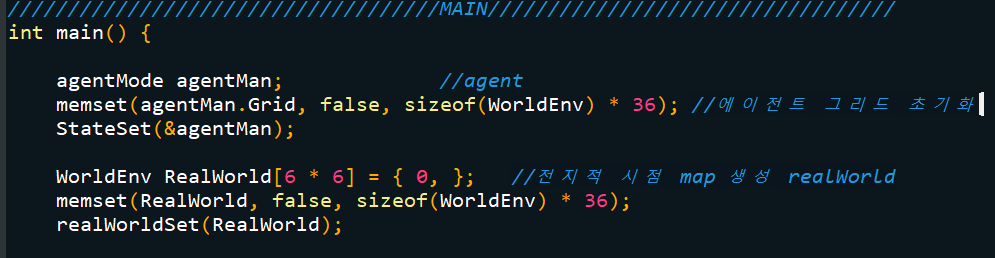
위 함수들은 웜퍼스 팀프로젝트 참고코드에 있는 함수들을 참고한 함수들이다.

유용한 함수들에 세번째 인자(WorldEnv\* Grid)를 넣어, 각각 함수들이 인자로 특정 맵을 받게해, 해당 Map에서 그리드의 상태 세팅시켜주게 하였다.

이제 본격적으로 c언어로 짠 코드 중, main 부분을 보며 어떻게 설계했는지 살펴보겠다.

|  |
| --- |
| ////////////////////////////////////MAIN//////////////////////////////////  int main() {  **agentMode agentMan; //agent**  **memset(agentMan.Grid, false, sizeof(WorldEnv) \* 36);**  **//에이전트 그리드 초기화**  **StateSet(&agentMan);**  **WorldEnv RealWorld[6 \* 6] = { 0, }; //전지적 시점 map 생성 realWorld**  **memset(RealWorld, false, sizeof(WorldEnv) \* 36);**  **realWorldSet(RealWorld);**  while (1)  {  char test[100];  ///////1.환경인식  **percept(&agentMan, RealWorld);**  if (!checkAlive(&agentMan))//에이전트 생존여부 체크  {  StateSet(&agentMan);  puts("\n\n=================================================");  puts("================Agent is dead====================");  puts("=================================================\n\n");  puts("Enter any key and retry");  gets\_s(test, 3);  continue;  }  ////2.환경업데이트  **UpdateState(&agentMan);** //에이전트가 인식하고있는 것을 토대로 에이전트가 가지고있는 맵을 업데이트해줌.  ////// 그리드 맵 출력  puts("\n====RealWorld Map========");  showMap(RealWorld, 9, 9);  puts("\n=======Agent Map=========");  showMap(agentMan.Grid, agentMan.x\_pos, agentMan.y\_pos);  if (!agentMan.cur\_percept.bumps)  printInfo(&agentMan);  //////////3. 액션  **agentMan.LastAction = ChooseAction(&agentMan);**    if (agentMan.cur\_percept.bumps)  printInfo(&agentMan);    gets\_s(test, 3);  }  } |

* Main 전체 코드
* **Agent Map과 RealWorld Map 생성**



먼저, 메인에서 동굴을 탐험할 Agent와, 전지적 시점 MAP, 즉 실제동굴세계를 6X6 배열로 만들어주었다.

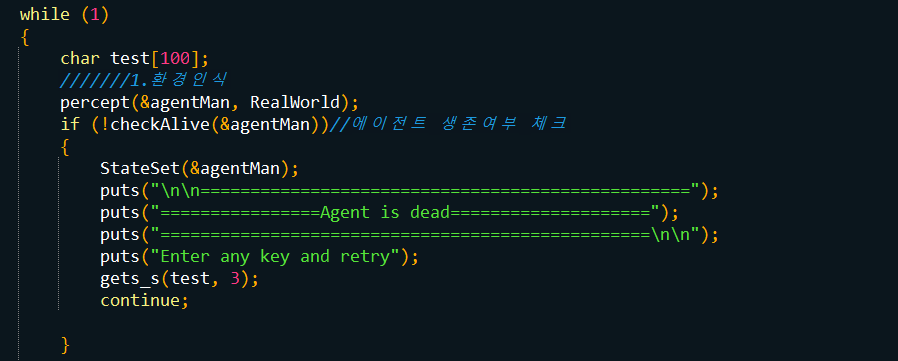
그리고, memset 함수를 통해 생성된 배열들을 모두 false 값으로 초기화를 해주었다.

그런 다음, StateSet() 함수를 통해, Agent를 초기상태로 초기화 해주는 작업을 진행했다.

* **StateSet(agentMode\* agentSt):** 시작 위치 (1,1), Default heading E, 화살 개수 3, 금 보유여부 false 등 에이전트의 상태를 초기화해주는 간단한 함수이다**.**

또, RealWorld[] (전지적시점 맵)도 초기화를 해주어야하니. realWorldSet()함수를 통해 초기화를 해주는 작업을 했다.

* **realWorldSet(WorldEnv\* RealWorld):** 각 모서리 Wall설정, safe zone 설정, 금 위치 3,4로 임의 생성, Wumpus와 Pit 랜덤 생성을 하는 간단한 함수이다.



이제 while문으로 들어간다. (char test는 while문을 한번 돌때마다 gets\_s 입력을 받아, 한번씩 멈추기위한 용도로 만든 임시버퍼이다.)

첫 번째로, 에이전트가 주어진 환경을 인식해야하니, percept() 함수를 실행하게된다.

* **void percept(agentMode\* agent, WorldEnv\* RealWorld):** 인자로 agent 변수의 주소와 월드맵의 주소가 들어간다. 메인에서는 미리 선언된 agentMan의 주소와 RealwWorld(전지적 시점 맵)의 주소를 넣어주게된다.

이 함수에서는 에이전트 구조체에서, cur\_percept, 즉, 현재 환경을 인지할 수 있도록 해주는 함수이다.

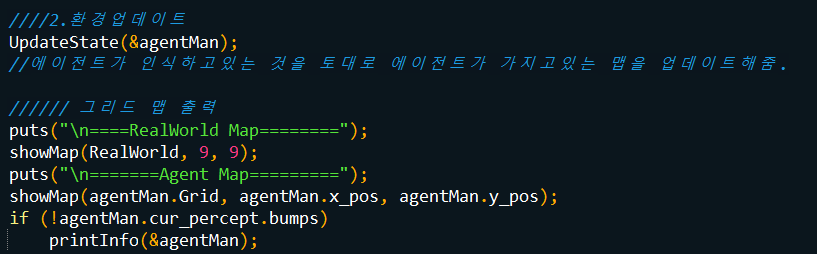
agent->cur\_percept = RealWorld[GridSize \* x + y]; 이런식으로 RealWorld에서 현 위치에 대한 상황을 에이전트가 현재 인지하고 있는 상황변수에 넣어 줌으로써 현재 환경을 인지할 수 있도록 해주었다.

추가로 화살을 쏘았을 때, 웜퍼스의 비명, 벽에 부딪힘, 골드소유여부도 이 함수에서 인지한다

이렇게 percept가 끝나면, checkAlive 함수를 통해 에이전트의 생존여부를 체크한다.

* **bool checkAlive(agentMode\* agent):** 에이전트가 현재있는 위치랑, RealWorld의 웜퍼스, 구덩이 위치랑 비교를 해, 생존 여부를 판단하고, 에이전트가 인식하고 있는 맵에도 해당 위치에 웜퍼스 또는 구덩이 위치를 업데이트해준다.

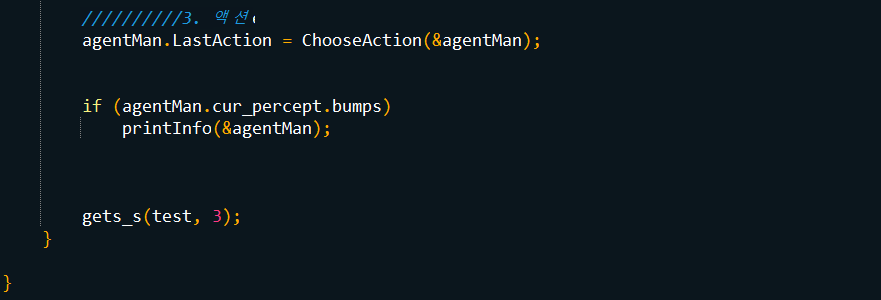
에이전트가 죽었다면, StateSet 함수를 통해 초기상태로(1,1) 돌아가 다시 시작하게 된다. 물론 에이전트가 죽을때까지 인식해왔던 각 맵위치에 대한 state는 그대로 유지된다.



percept와 생존여부판단이 끝나면, UpdateState를 진행하게된다. (물론 에이전트가 죽었을땐 continue를 통해 남은 while문 과정들을 모두 생략한다.)

* **void UpdateState(agentMode\* agentSt):** State를 업데이트해주는 함수이다. 이 함수에서는 에이전트가 현재 인식하고 있는 상황(cur\_percept)을 참고로 여러가지 state를 업데이트한다.
  + 벽이랑 충돌을 했으면, 해당위치가 벽이라고 맵에 업데이트를 한다.
  + 1,1위치 일 때 금을 가지고 있으면 목표달성.
  + 금을 가지고 있지 않을 때, 현재 위치가 반짝이면 현재 위치에 골드가 있다는 것을 업데이트.
  + 악취를 인식했으면, 맵 현재 위치엔 악취나는 곳이라고 업데이트.
  + 미풍을 인식했으면 맵 현재 위치엔 미풍이 부는 곳이라고 업데이트
  + 미풍, 구덩이, 악취, 웜퍼스 모두 없으면 안전한 곳이라고 업데이트

UpdateState함수가 끝나면, 사용자가 현재 상황을 볼 수 있게 에이전트 맵, RealWorld 맵을 showMap() 함수를 통해 출력시킨다. showMap()은 말그대로 해당 맵을 출력시키는 것이라 자세한 함수분석은 생략하도록 하겠다. printInfo() 함수 역시 헤딩방향이랑 골드 소유여부, 화살개수, 현재위치를 출력시키는 간단한 함수이다.

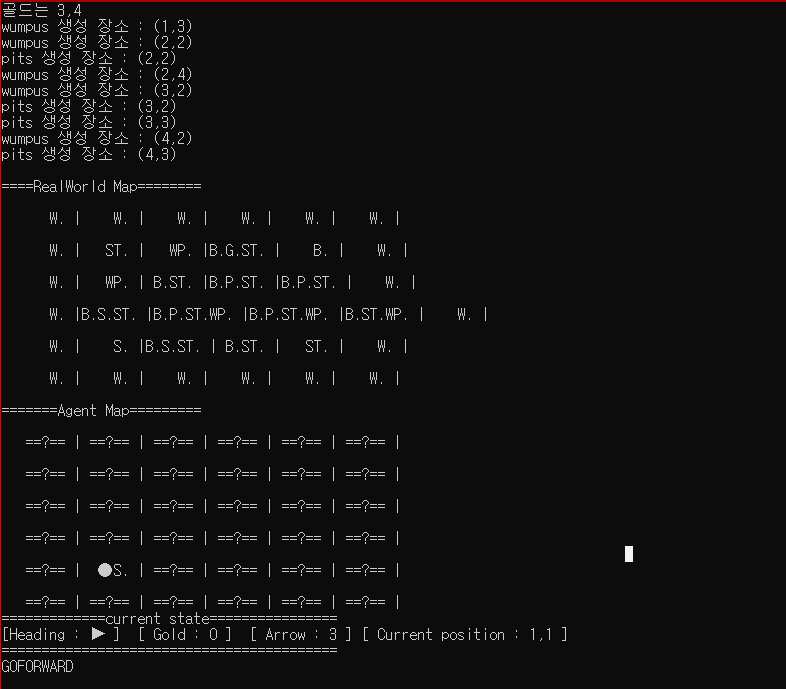


마지막으로 Action 부분이다. ChooseAction()함수의 리턴 값을 에이전트의 마지막 액션 값에 집어넣게 된다. ChooseAction()함수의 리턴 값은 actionList이다.

|  |
| --- |
| typedef enum { NONE, TURN\_LEFT, SHOOT, GOFORWARD, TURN\_RIGHT, GRAB, CLIMB } actionList; |

* **actionList ChooseAction(agentMode\* agent)** : 이 함수에서 에이전트가 취할 action을 판단한다. 이 함수 역시 에이전트가 현재 인식하고 있는 환경(cur\_percept)을 참고해서 action을 판단하게 된다.
  + 목표를 달성했을 때 : CLIMB를 통해 동굴을 빠져나간다.
  + 골드를 인식했을 때 : GRAB을 통해 골드를 잡는다.
  + 미풍과 악취가 동시에 날 때 : 현재 헤딩 방향을 참고해, 헤딩 방향에 구덩이는 없고 웜퍼스만 있다면 화살을 쏘고, 그게 아니라 헤딩 방향에 웜퍼스 존재여부 상관없이 구덩이가 있으면 왼쪽으로 헤딩을 바꾸고, 앞서말한 해당사항이 없으면 앞으로 전진한다.
  + 미풍만 날 때 : 이 역시 현재 헤딩 방향에 구덩이가 없으면 전진하고, 있다면 왼쪽으로 헤딩방향을 바꾼다.
  + 악취만 날 때 : 현재 헤딩 방향에 웜퍼스가 있다면 화살을 쏘고, 그렇지 않으면 전진한다.
  + 벽에 부딪혔을 때 : 벽에 부딪히면 왼쪽으로 헤딩방향을 바꾸는 것으로 설정했다. 그치만 골드를 못찾고 계속 벽에 부딪히면서 빙빙 도는 상황이 발생해, 헤딩 방향을 바꾸었는데도 벽에 10번이상 부딪히면 반대방향으로 헤딩방향을 바꾸는 것으로 설정했다. (왼쪽 10번, 오른쪽 10번, 왼쪽 10번….)
  + 위에 해당상황이 없을 때 : 앞으로 전진한다.

1. **결과 및 고찰**



**(초기실행화면)**

초기엔 에이전트가 벽에 부딪히면 왼쪽으로만 헤딩을 전환하게 했는데, 이렇게하니 골드를 찾지못하고 벽에만 부딪히면서 빙빙 도는 현상이 발생해, action()함수에서 해당 헤딩 방향으로 벽에 10번 부딪힐때마다 반대 방향으로 헤딩을 전환하게 **(ex: 왼쪽으로 전환하게 했는데 10번 부딪히면 앞으론 오른쪽으로 전환하게 설정)** 했다. 그 결과, 만족스럽게 빙빙 도는 상황에서 벗어나 골드를 찾아갈 수 있게 되었지만, 이보다 더 좋은 방법, 더 합리적인 아이디어가 있었으면 어땠을까 하는 생각도 들었다.

이렇게 전체적으로 결과는 잘 작동되었지만 예상치 못한 문제들과 아쉬웠던 점을 들어보겠다.

* **wumpus와 pits의 잦은 발생**

생각보다 0.15의 확률이 높은 것 같았다. 위 결과처럼 wumpus와 pits가 매우 많이 발생하는 상황이 자주 발생하곤 했다. 그래서인지 pits가 많이 발생했을 경우, 에이전트가 벽과 pits를 피해 골드위치까지 가지 못하고 계속 빙빙 도는 상황이 발생할 수도 있었다.

* **에이전트가 죽었을 때, State Update 문제, ( 무엇 때문에 죽었는지 인식 )**

이 부분이 로직에서 제일 아쉬운 부분이었다. 역시 발표 때 교수님께 질문을 받은 문제이기도 하다.

초기엔 에이전트가 죽었을 때, 죽기 직전 에이전트가 인식한 환경 (cur\_percept)를 참조해, 마지막으로 미풍을 느꼈는데 죽었으면 구덩이 때문에 죽었고, 마찬가지로 악취를 느꼈는데 죽었으면 웜퍼스 때문에 죽었다고 인식을 하게 시도를 했다.

그렇게 진행하다 보니 에이전트가 미풍과 악취를 동시에 인식한 뒤 죽으면 에이전트맵상 죽은 위치에 구덩이와 웜퍼스가 둘 다 기록되는 상황이 발생하게 된다. 실제 RealWorld 맵에선 해당 위치에 웜퍼스 또는 구덩이 둘 중 하나만 있었는데 말이다.

이렇게 미풍과 악취를 동시에 인식한 뒤 죽었을 때, 잘못된 state를 업데이트하는 것을 막고자 checkAlive() 함수에서 에이전트가 죽는 순간 환경을 그대로 인식해 에이전트 맵에 state를 업데이트하게 만들었다.

다시 생각해보면, 액션를 판단하는 함수에서, 현재 헤딩 방향 다음 칸에 웜퍼스가 있어야만 화살을 쏘는게 아니라 악취가 나기만 하면 일단 화살을 쏘게하면 어땠을까 하는 아쉬움이 있다. 그렇게하면 미풍과 악취가 동시에 인식된 상황에서 화살을 쏨으로써 해당방향 다음 칸에 웜퍼스의 존재여부를 파악할 수 있기 때문이다. 물론 웜퍼스가 많이 생성되어 악취가 많이나는 맵에선 화살 3개로 부족한 경우가 매우 높은 확률로 발생될 문제가 있다.

결과적으로 이 부분이 가장 아쉬운 부분이긴 하다.

1. **역할 분담**

* 정지용 : main함수 구성 논의, 프로젝트 개발
* 고다영 : 도메인 분석, main함수 구성 논의, 버그 탐색, PPTㆍ보고서 작성
* 김태진 : main함수 구성 논의, 인터페이스 개발, 버그 탐색
* 이승희 : 도메인 분석, main함수 구성 논의, 버그 탐색, 발표

1. **프로젝트 소감**

* 정지용 : 생각보다 고려해야할 경우의 수가 다양해서 시행착오를 겪었지만, 이렇게 팀프로젝트로 에이전트 구현을 위한 과정들이 정말 보람찼고, 그 중에서도 구현을 위한 서로 팀원들간 아이디어, 의견들을 적극적으로 공유하는 것이 좋았다. 다들 팀워크가 정말 잘맞은 것 같아 정말 뿌듯했다.
* 고다영 : 이론으로 배운 Wumpus World를 구현하기란 생각만큼 간단하지 않았다. Agent가 사람처럼 특정 상태에 대해 기억하고 올바른 행동을 이끌어 낸다는 것이 코드로 구현하려면 복잡한 과정이었다. 하지만 과정 중에서 어려운 부분이나 수정이 필요한 부분은 팀이 다함께 논의하며 좋은 아이디어를 반영하여 끝마칠 수 있었다. 어려운 프로젝트였지만 그만큼 많이 배울 수 있었고, 좋은 팀원들과 함께하여 즐거웠다.
* 김태진 : 개발을 맡아 인터페이스에 대해 고민을 많이해보았고, 생각한 것과는 다르게 인터페이스가 잘 해결이 되지않았지만, 팀원과 함께하면서 에이전트 구현을 생각해보았던게 보람찼다. 회의를 통해서 예외 상황들을 찾는게 재밌었고 정말 재밌게 팀 프로젝트를 했다.
* 이승희 : 이번 팀프로젝트에서 자료 조사를 맡아 함수를 분석하면서 일어날 수 있는 상황과 변수에 대해 잘 고려하여 해결해 나갈 방법을 생각해야 한다는 것을 다시 한 번 알게 되었다. 그리고 그러한 문제를 해결하기 위해서는 다 같이 문제에 대한 아이디어를 나누는 것이 중요하다는 것을 배웠다. 이런 모든 점에서 뜻 깊은 팀 프로젝트였다고 생각한다.