알고리즘
설계 프로젝트

프로젝트 명	*PushBox*
팀 명	*GunBullDo*
문서 제목	결과보고서

Version	Final
Date	2019-12-05

팀원	송성재(팀장)	20153189
	신상훈	20153191
	김연수	20153159
	전하훈	20171697
	최운호	20171711
	최현인	20171716
지도교수	최준수 교수	

문서 정보 / 수정 내역

Filename	결과보고서-PushBox-GunBullDo.doc
원안작성자	신상훈
수정작업자	김연수 송성재 최현인 전하훈 최운호

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2019-10-21	신상훈	1.0	최초 작성	보고서 초안 및 틀 작성
2019-10-28	송성재	1.1	내용 추가	프로젝트 목표 등 작성
2019-11-02	김연수	1.2	내용 추가 및 수정	프로젝트 목표 수정 및 알고리즘(BFS) 작성
2019-11-09	최현인	1.3	내용 추가	알고리즘(DFS) 추가 작성 및 연구 내용 작성
2019-11-16	전하훈	1.4	내용 추가	알고리즘(A*)추가 작성 및 수행 내용 작성
2019-11-23	최운호	1.5	내용 추가	구현 코드 작성 및 수행 내용 작성
2019-11-29	공동	1.6	내용 추가&수정	보고서 검토 및 작성 마무리 작업
2019-12-04	**공동**	**Final**	**최종 검토**	**최종 검토 및 제출**

목 차

1 프로그램 실행방법 및 맵 형식

프로그램 실행방법

- $ make
- $ ./sokoban.out input.txt

맵 형식

- input.txt

```
7 4      // 열, 행
4111114
1325231
1320231
4111114
```

0 : 길 1 : 벽 2 : 박스 3 : 도착지점 4 : 빈공간 5 : 캐릭터

- 주석은 들어가면 안되며 맵의 형식은 중간에 공백이 들어가도 상관없다.

1. 방향키를 이용해서 맵을 클리어 할 수도 있다.
2. Z키를 누르면 이전 상태로 돌려 준다.
3. R키를 누르면 초기 상태로 돌려 준다.
4. S 키를 누르면 솔버 모드로 들어가서 중간에 입력을 할 수 없이 솔버 과정을 보여주고 게임이 클리어 된다.

2 프로젝트 목표

1. Push Box 게임을 C++ ncurses라이브러리를 이용해 유저인터페이스를 구현한다.
2. 유저인터페이스를 구현함에 있어 플레이어가 쉽게 인식할 수 있도록 오브젝트를 명확하게 구분한다.
3. 게임에 제공하는 모든 스테이지는 반드시 클리어가 가능한 맵으로 제공한다.
4. 플레이어는 Push Box 게임의 규칙이 허용하는 선에서 방향키를 이용해 게임을 진행 할 수 있다.

5. 플레이어는 자신이 이동한 횟수와 박스를 밀어낸 횟수를 인터페이스를 통해 인식할 수 있다.
6. 플레이어는 이전 상태로 돌아가길 원하는 경우 Z키를 이용해 바로 이전의 상태로 돌아갈 수 있다.
7. 플레이어는 자신이 원하는 경우 S 버튼을 눌러서 해당 스테이지의 해답을 볼 수 있다.

3번에서 언급한 클리어 가능한 스테이지란 다음과 같다.
1. 박스의 개수와 목적지의 개수는 같아야 한다.
2. 박스, 목적지, 캐릭터 3가지가 모두 맵 안에 존재해야 한다.
3. 캐릭터가 모든 박스를 목적지로 이동시킬 수 있어야 한다.
4. 위의 모든 조건이 동시에 충족해야 한다.

4번에서 언급한 Push Box 게임의 규칙은 다음과 같다.
1. 플레이어는 방향키를 이용해 자신의 위치를 가로 또는 세로로 이동할 수 있다.
2. 플레이어는 벽을 밀거나 통과해서 지나갈 수 없다.
3. 플레이어가 진행하는 방향에 박스가 있을 경우 박스를 밀면서 지나갈 수 있다. 단 박스가 옮겨질 위치에 벽 또는 박스가 막고 있지 않는 경우로 한정한다.
4. 게임의 클리어 조건은 모든 박스를 목적지로 이동 시키는 것 이다.

6번에서 언급한 이전 상태로 돌아가기는 다음과 같다.
1. 박스와 캐릭터가 현재 위치로 이동하기 바로 직전의 위치로 돌아간다.
2. 윈도우에 표시되는 박스와 캐릭터의 이동 횟수는 감소하지 않는다.
3. 이 기능은 해당 스테이지의 최초 상태까지 적용 가능하다. 이전의 스테이지로는 돌아갈 수 없다.

7번에서 언급한 해당 스테이지의 해답은 다음의 형태로 제공한다.
1. solve 윈도우에 스테이지를 클리어 할 수 있는 이동 순서를 텍스트 형태로 제공한다.
   제공된 텍스트대로 이동했을 시 반드시 스테이지는 클리어된다.
   만약 텍스트의 길이가 매우 길어 solve 윈도우를 초과할 경우 적정 길이만 유동적으로 제공한다.
1. solve 윈도우에 제공되는 텍스트는 다양한 알고리즘을 통해 구한 뒤 소모시간과 이동거리가 가장 효율적일 수 있는 알고리즘으로 구해진 텍스트를 제공한다.
2. 텍스트를 제공함과 동시에 캐릭터를 텍스트와 동일하게 이동시켜 가시적으로 플레이어가 확인할 수 있도록 한다.

3 프로그램 설계 및 수행 내용

3.1 프로그램 설계

A팀 -> UI 구성 및 게임 구현 , z 키를 누르면 이전 상황, r 키를 누르면 초기화를 하고 s를 누르면 Solver 모드로 전환 시킨다. Solver 모드시 Input String을 한 문자씩 Pop 하는 과정을 ui로 나타냄.

1. UI는 Window를 GameWindow, StatusWindow, PathWindow 세가지로 나눠서 구현한다.
    a. GameWindow : 해당 스테이지의 맵을 표현한다.
    b. StatusWindow : 해당 스테이지의 레벨, 현재 박스와 캐릭터의 이동 횟수를 표현한다.
    c. PathWindow : S키를 눌러 Solver모드로 진입했을때 해답 경로를 표현한다.
2. Z키를 누르면 이전 상태로 돌아가는데 이 기능은 박스와 캐릭터의 이동횟수에 변동을 주면 안되기 때문에 GameWindow에만 해당하는 기능이다.
3. S키를 눌러 Solver모드로 진입했을 경우 패스와 캐릭터의 이동을 가시적으로 보여줘야 하기 때문에 플레이어는 User모드로 진입하기 전까지 방향키를 사용할 수 없어야 한다, Solver모드를 시작한 스테이지가 종료되고 다음 스테이지로 넘어가면 자동으로 User모드로 시작한다.
4. Solver 모드로 진입 했을 경우 PathWindow에 패스를 보여주는데 해당 스테이지의 Path를 나타내는 String의 길이가 PathWindow의 크기를 넘어갈 경우 적정 길이의 Path만 보여주며 캐릭터가 Path를 따라 움직일때마다 남은 경로를 PathWindow에 갱신시킨다.

B팀 -> A팀에서 만든 UI 및 게임 형식에 맞춰서 Solver 알고리즘을 구현한다. Solver 알고리즘으로는 BFS, DFS, GBFS, A-Star 알고리즘을 사용하며 각 알고리즘에서 나온 경로중 최단 경로를 반환하는 Sub-routine을 구현한다.

3.2 수행내용

A팀 -> UI 구성 및 게임 구현 , z키를 누르면 이전 상황, r키를 누르면 초기화를 하고 s를 누르면 solver 모드로 전환 시킨다. solver 모드시 다음 입력값을 pop 하는 과정을 ui로 나타냄.

1. 게임을 실행할 때 argv로 파일의 경로를 넘겨준다.
2. Map을 불러와서 GameWindow에 표현하는 Function을 구현. Map은 InputSize와 0부터 5까지의 Number로 이루어진 File이다. 0은 이동할수 있는 Road, 1은 Wall, 2는 Box, 3은 Destination, 4는 벽 밖의 EmptySpace, 5는 Character로 설정이 돼있다. Map file을 읽어온 뒤 GameWindow에 위치에 맞게 배치를 해주며 색깔로 구분해준다. 벽은 흰색, 박스는 녹색, 목적지는 청색, 캐릭터는 &기호를 사용하여 플레이어가 쉽게 구분할 수 있도록 설정했다.
3. StatusWindow에 플레이어가 캐릭터를 이동시키면 이동횟수를 표현해주는 기능을 구현. 캐릭터의 이동 횟수는 플레이어가 캐릭터를 진행 가능 방향으로 이동시켰을 때에만 증가시키며, 박스 또한 캐릭터가 박스를 진행 가능 방향으로 이동시켰을 시에만 이동횟수를 증가시킨다.

4. 이전 상태로 이동하는 기능을 구현. 플레이어가 z키를 눌렀을 경우 이동 직전의 상태로 GameWindow를 갱신한다. GameWindow만 갱신하기 때문에 당연히 StatusWindow에 표현되는 이동횟수는 감소하지 않는다.
5. r키를 누르면 해당 스테이지를 리셋하는 기능을 구현. 모든 윈도우를 해당 스테이지의 초기 상태로 갱신한다. 모든 상태가 초기화 되기 때문에 플레이어는 리셋하기 전의 상태로 돌아갈 수 없다.
6. B팀의 Sub-routine을 추가해 Solver모드를 구현. B팀의 알고리즘으로 최적의 패스를 리턴받아 배열에 담아 저장한다. 저장된 배열은 PathWindow에 Path를 표현하는데 쓰이며 PathWindow를 넘어가지 않는 선에서 표현되며 Pop기능을 통해 현재 이동해야 하는 방향이 가장 앞에 나오도록 String을 감소시키며 갱신한다.

B팀 -> 기본적인 알고리즘 : is_goal : 게임 클리어 조건을 만족했는지 확인하는 함수를 구현한다. is_goal 함수는 목적지에 목적지임을 표현하는 문자가 없으면 목적지에 플레이어가 올라가 있는 것이 아닌지 확인하고 만약 플레이어가 서 있지 않다면, 맵 상에 상자가 있는지 없는지 확인한다. 위의 조건을 모두 만족할시 True를 리턴하고 하나라도 만족하지 못한다면, False를 리턴한다.

1. 현재 State를 계산한다 -> 현재 State를 계산한다는 뜻은 현재 노드에서 갈 수 있는 노드를 계산하고 해당 노드까지의 Cost를 계산한다. 이 알고리즘은 미리 gen_valid_state에서 맵을 받아오고 모든 노드에 대해 움직일 수 있는 노드를 계산한다. 움직일 수 있는 노드를 구하는 것은 미리 노드를 검사 한 뒤 해당 노드에서 목표지점까지의 Cost를 미리 계산한다. 이때 휴리스틱 알고리즘을 사용한다. 휴리스틱 알고리즘이란 어떤 노드 A에서 목적지까지의 거리를 계산하는 알고리즘이다. 단순히 유클리디안 거리를 이용해서 표현해도 되지만 해당 알고리즘을 사용할 경우 모든 노드를 탐색해야하는 경우가 생길 수 있다. 따라서 이를 응용하여 알고리즘을 제작한다. 이 휴리스틱 알고리즘을 사용하여 현재 State에서 갈 수 있는 노드를 계산할 수 있는 함수 gen_valid_state를 제작한다. 이 함수는 박스를 고려하여 박스가 불안정한 위치에 가는 것을 방지하고, 박스가 목적지까지 갈 수 있는 최단경로를 계산할 수 있다.
2. BFS : 노드에서 갈 수 있는 경로를 구하여 open_list에 넣고 현재 노드는 closed_list에 넣는다. 여기서 open_list는 탐색할 경로를 넣고, closed_list는 이미 탐색한 노드를 넣는다. 따라서 BFS는 open_list에 들어간 것들을 우선적으로 탐색한다. 이는 queue의 제일 앞에서 하나씩 pop 하는 형식으로 구현하였다. BFS알고리즘으로 나오는 경로는 최적의(최단) 경로가 아닐 확률이 높다. 왜냐하면 BFS로 탐색하는 도중에 is_goal을 만족할 경우 해당 경로를 출력하기 때문이다. 따라서 BFS알고리즘은 선택받지 못하는 경우가 많다.
3. DFS : DFS는 단순히 현재 노드에서 갈 수 있는 노드를 모두 stack에 쌓고 위에서부터 pop하는 형식으로 구현할 수 있다. 따라서 재귀적으로 구현하기 위해 stack 기법을 사용하였다. 현재 노드에서 갈 수 있는 노드를 push_back한 다음에 해당 노드로 건너가고 해당 노드에서 갈 수 있는 노드를 모두 push_back한 뒤 stack의 제일 상단부분을 pop한다. DFS도 마찬가지로 현재 노드에서의 cost만 계산하면서 진행하기 때문에 BFS처럼 출력되는 경로가 최단 경로가 아닐 확률이 높다. 따라서 DFS알고리즘도 선택받지 못하는 경우가 많다.
4. GBFS(Greedy Best First Search) : GBFS는 우선 순위 큐를 사용한다. 대기열에서 처음 상태를 취하고 목표 상태인 경우 종료한다. 목표 상태가 아닌 경우 유효한 모든 상태가 생성되고 휴리스틱 함수 기반으로 큐에 배치 시킨다. explored list는 무한 반복을 방지하는데 사용하고 모든 유효한 상태의 노드는 점수를 기준으로

큐에 배치 시킨다. 점수가 큐에 있는 노드와 동일한 새 노드는 동일한 점수를 가진 노드 목록의 끝에 배치 시켜 동일한 점수를 가진 노드에 추가한다.

5. A-star : A-star 알고리즘은 시작 노드와 목적지 노드를 분명하게 지정해 이 두 노드 간의 최단 경로를 파악할 수 있다. 휴리스틱 추정값을 기반으로 하고 있으며 초기 상태에서 노드까지의 총 비용을 휴리스틱 함수 점수에 추가하고 휴리스틱 추정값을 개선하며 진행한다.
6. 휴리스틱 함수는 상자, 목표 및 플레이어의 거리를 고려하지 않고 상자의 상태에 따라 점수를 부여한다. 점수는 0부터 시작하며 상자가 "안전하지 않은 위치"에 있는지 확인한다. "안전하지 않은 위치"란 목적지가 아닌 코너와 목적지가 라인 끝에 없는 벽을 말한다. 즉, "안전하지 않은 위치"인 코너에 있는 상자와 벽에 붙은 상자에 +1000 점을 부여한다.

따라서, 휴리스틱 함수로 상자에 점수를 부여하고 A* 알고리즘을 통해 시작노드와 목적지 노드간의 점수가 낮아지도록 개선하며 진행한다.

4 최종보고서 본문

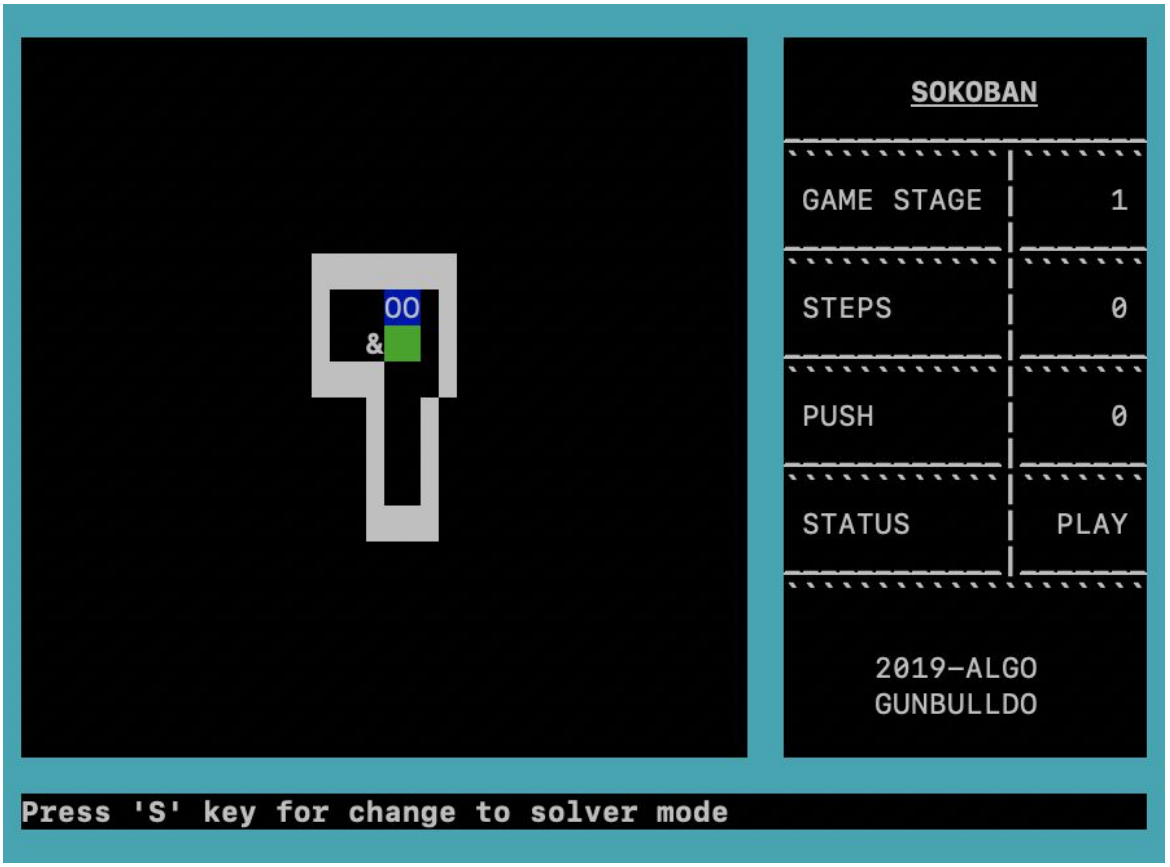
<Game 소스코드>
첨부파일 대처

<Solver 소스코드>
첨부파일 대처

<Main 소스 코드>
첨부파일 대처

5 자기평가

현재 운용 중인 게임과 견주어 봤을 때, 기능적인 부분은 거의 완벽하게 똑같음을 확인 할 수 있었다. GUI 또한 레트로의 감성을 살려 잘 구현하였고, solver 모드로 전환 시 지나 가는 경로를 가시적으로 보여주었으며 Path 윈도우에 팝 기능을 이용해 남은 경로를 표현해 줌으로서 전광판 같은 시각적인 효과를 줄 수 있었다. 또한 캐릭터를 움직인 횟수와 박스를 밀어낸 횟수 등을 키 입력을 받을 때 마다 갱신하여 보여줌으로써 플레이어가 현재 게임의 진행 상태를 알 수 있도록 잘 표현했다.



알고리즘을 통한 해답 경로를 얻는 시간동안 화면이 잠시 멈춰있기 때문에 플레이어가 혼란스럽지 않도록 로딩화면을 구현했으며 윈도우를 세개로 나눔으로써 게임 이용자의 가독성을 더 높일 수 있었던 것 같다. 현재 맥북에서 실행 시켰을때 박스가 초록색이라 합쳐지는 현상으로 보이지만 ubuntu 에서는 확연하게 다른 반응을 볼 수 있었다. 또한 A_BLINK 라는 기능 또한 맥북에서는 작동하지만 ubuntu 에서는 작동하지 않음을 확인 할 수 있었다. 이런 기능등을 고려하면서 최대한 ubuntu 환경에 맞춰서 개발하였다. 이렇게 팀원들의 서로 다른 개발환경으로 인해 merge 과정에서 어려움을 겪었기 때문에 개발환경 통일의 필요성을 알게 됐으며 ncurses뿐만 아니라 다른 GUI 모듈을 통해서도 개발을 할 수 있는 기회를 얻는다면 좀 더 퀄리티 있는 프로젝트를 만들어 보고 싶다. 이번 프로젝트는 알고리즘 구현과 제한 적인 GUI 에 맞춰 최대한 가독성있게 만든 것에 대해 만족감을 얻고 있다.