**소프트웨어개발도구및환경실습**

**최종 프로젝트 보고서**

20231515

컴퓨터공학과

김다은

|  |
| --- |
| <목차>   1. 프로젝트 목표 2. 게임 실행 환경에 대한 설명 3. 각 변수에 대한 설명 4. 각 함수에 대한 설명 5. 프로젝트 전체 플로우차트 + 자료구조 및 알고리즘 + 시간/공간복잡도   5-1. 프로젝트 전체 플로우차트  5-2. 사용된 자료구조  5-3. 사용된 알고리즘  5-4. 시간/공간복잡도 고려   1. 프로젝트 실행 결과 캡쳐 2. 느낀 점 및 개선사항 |

**1. 프로젝트 목표 및 게임 설명**

게임 명: Delivery Game

<프로젝트 목표>

목적지까지 최소한의 Time과 Power를 소비해서 도달하여 The Best Delivery Man이 되는 것을 목표로 하는 Delivery Game 구현

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

<게임 규칙>

여러분은 매우 중요한 택배를 배달해야 하는 택배기사이다. 0번째 도시에서 시작해서 9번째 도시까지 최대한 적은 Power와 Time를 소모하여 배달을 완료하는 것이 목표이다. 그러나 각 도시들 간의 경로는 한정적이다. 도시간의 경로를 지도를 통해 확인하고 가장 효율적인 경로로 이동해서 배달을 완료해야 높은 점수를 받을 수 있다. 제한시간이 끝나거나 목적지에 도달하면 게임이 종료된다.

이때, 한 번 방문했던 도시는 재방문이 가능하며, 지도 상으로 경로가 존재하지 않는 도시 사이에는 이동이 불가능하다. 최종 스코어는 사용한 Power와 Time를 기반으로 산출된다.

<지도 표시 정보>

지도에는 0~9까지의 도시들이 나타나 있다. 각 선 위에 쓰여진 숫자는 해당 경로로 배달 기사가 이동할 때 소모하게 되는 Power이다. 또한 텍스트로 소모되고 있는 Time이 나타난다.

스크린샷, 텍스트, 도표이(가) 표시된 사진

자동 생성된 설명

* 아직 한 번도 거쳐가지 않은 도시: 파란색
* 적어도 한 번 거쳐간 도시: 노란색
* 직전에 지나간 도시: 주황색
* 현재 선택한 도시: 빨간색

<조작법>

* 0~9까지의 숫자를 눌러 이동하고 싶은 도시를 선택한다.
* Space Bar를 눌러 해당 도시로 이동하고 Power를 소모한다.

**2. 실험 환경에 대한 설명**

<data 파일 설정하기>

`projectfile/bin/data`에 다음 3개 파일이 존재해야 한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

또한 visualStudio 2022에서 개발을 진행하였다.

**4. 각 변수에 대한 설명**

**텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

게임의 진행 상태를 크게 4가지로 나누어 저장한다.



Main Menu에 배달원 이미지를 저장하기 위한 변수이다.



게임 진행 도중 에러가 발생한 경우(두 도시 사이에 길이 없는데 이동하려고 하는 경우 등) 에러 메시지를 저장했다가 draw 함수에서 호출되어 출력된다.



플레이어가 선택한 map의 이름을 저장한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**5. 각 함수에 대한 설명**

**<초기 설정 역할을 하는 함수들>**

void setup();

프로그램이 실행될 때 호출되는 함수이다. 전역변수에 대한 초기화를 수행한다.

void initializeGraph();

각 노드(도시)의 지도에서의 위치를 지정한다.

void keyPressed(int key);

gameState 값을 기준으로 플레이어의 keyboard input을 읽어와 처리해주는 함수이다.

void readmap(string filename);

filename의 이름을 가지는 지도 파일을 읽어와 graph 자료구조에 저장한다.

**<화면 UI를 그리는 함수들>**

void draw();

gameState 값에 따라 아래 4개 함수를 호출하는 역할을 한다.

void drawInstructions();

gameState == INSTRUCTIONS인 경우 호출되는 함수이다. 게임 플레이 방법에 대한 설명이 있는 UI를 그린다.

void drawGraph();

graph 자료구조에 저장된 정보를 토대로 graph(지도)를 그린다.

void drawMenu();

gameState == MENU인 경우 호출되는 함수이다. 메뉴 화면 UI를 그린다.

void drawGame();

gameState == GAME인 경우 호출되는 함수이다. 게임 화면 UI를 그린다.

void drawResult();

gameState == RESULT인 경우 호출되는 함수이다. 게임이 종료되었을 때, 스코어와 안내 메세지를 포함한 UI를 그린다.

**<게임 진행 과정에 관여하는 함수들>**

void updateConsumedPower(int preCity, int curCity);

사용자가 다른 도시로 이동했을 때, 소모한 power 값을 알아내 consumedPower 변수의 값을 갱신한다.

bool checkValidSelect();

서로 이어져 있지 않은 도시를 이동하는 것은 불가능하다. 이 경우를 판별하기 위한 함수로 이동이 가능하면 true를 리턴하고 이동이 불가능하면 false를 에러 메시지와 함께 리턴한다.

void selectCity();

스페이스바가 눌렸을 때, 배달기사의 city 이동을 처리하는 함수이다. visitedCity 배열을 갱신하고 previousCity를 갱신한다. 만약 목적지에 도착한 경우 gameState를 RESULT로 설정한다.

**<최종 스코어 산출에 관여하는 함수들>**

void calculateWeightSum(); void readMap(string filename);

graph의 모든 간선의 weight를 합하여 저장하는 함수이다. 이 합은 나중에 최종 스코어를 계산할 때 활용된다.

int dijkstra(int start, int n);

priority queue를 이용한 다익스트라 알고리즘을 수행하는 함수이다. 출발 도시부터 목적지 도시까지 가는 경로 중 Power를 제일 적게 소모하는 경우의 Power 소모량을 리턴한다.

void updateScore();

gameState가 RESULT로 바뀌었을 때, 화면에 표시될 Score를 계산하는 함수이다. dijkstra 함수를 호출하여 최소 Power 소비량을 알아낸 뒤 플레이어의 Power 소비량과 비교한다. 또한 플레이 시간을 고려하여 Score를 update한다.

**4. 프로젝트 전체 플로우차트 + 자료구조 및 알고리즘 + 시간/공간 복잡도 상세 설명**

**4-1. 프로젝트 전체 Flow Chart**

텍스트, 도표, 평면도, 스케치이(가) 표시된 사진

자동 생성된 설명

gameState로 게임의 큰 틀이 잡힌다. 이후 키보드 인풋 값에 따라 각 함수를 실행하는 방식이다.

**4-2. 사용된 자료구조**

**4-2-1. GameState: 게임 상태를 구분하기 위한 자료구조**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Delivery Game의 경우 게임 상태를 위 4가지로 구분할 수 있다.

* MENU: 게임 시작 전 메인 메뉴가 표시되는 상태
* GAME: 게임이 진행중인 상태
* RESULT: 게임이 끝나고 점수가 표시되는 상태
* INSTRUCTIONS: 플레이 방법이 표시되는 상태

각 상태별로 draw 함수를 호출하고 key input을 처리하기 위해 enum을 이용해 선언하였다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**4-2-2. graph: 지도를 구성하기 위한 자료구조(가중치가 있는 그래프의 인접리스트 저장 형태)**



지도를 그리기 위한 각 city 사이에 이동하는 데 소모되는 Power값이 저장하기 위해 만든 자료구조이다. graph의 구조를 알기 쉽게 나타낸 도식이다.

u와 v1, v2, v3 … 도시를 연결하는 길의 Power 소모량이 weight라고 했을 때, graph는 다음과 같이 adjacent list 형태로 그래프를 저장한다. 이는 미로 실습시간에 다루었던 ‘가중치가 없는 그래프’에서 한 단계 발전하여 pair 자료구조를 이용해 연결된 두 노드의 정보와 함께 가중치(Power)값까지 함께 저장하고 있는 형태이다.

텍스트, 친필, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**4-2-3. nodePositions: Scity의 위치를 저장하기 위한 자료구조**



ofVec2f는 x, y 좌표를 가지는 2차원의 지점을 쉽게 저장할 수 있도록 해준다. 이를 통해 지도에서 각 city가 위치하는 좌표를 vector 배열로 저장하였다.

**4-2-4. priority queue: 다익스트라 알고리즘을 효율적으로 적용하기 위해 사용한 자료구조**



greater<> 규칙을 적용하여 최소 priority queue를 선언하였다. 여기에서 pair의 second는 해당 노드의 번호를 저장하고 있으며 first 요소는 해당 노드까지의 거리를 저장하고 있다.

**4-3. 사용된 알고리즘**

**4-3-1. 두 도시(노드)가 연결되었는지 판단하는 알고리즘(adjacent list 해석)**

두 도시가 연결되지 않았다면 배달원은 이동할 수 없다. 이를 판별하기 위한 알고리즘이다. 이는 adjacent list의 정보를 해석하여 return true, false 여부를 결정하다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**4-3-2. 다익스트라 알고리즘(최적의 Power 소비량을 찾아 최종 스코어 계산에 활용)**

게임이 종료되면 다익스트라 알고리즘을 이용해 출발지부터 도착지까지의 모든 경로 중 Power가 제일 적게 소모되는 경로를 구한다. 다익스트라 알고리즘을 통해 구한 최소 Power와 사용자가 사용한 Power를 비교하여 점수를 산출한다.

다익스트라를 효율적으로 구현하기 위해 priority queue를 사용하였다. 이 큐에는 greater<> 규칙이 적용되었는데 이는 top에 제일 작은 요소가 위치하게 된다는 것을 의미한다. 자세한 작동 방식은 아래 플로우차트로 표현하였다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 도표, 평행, 라인이(가) 표시된 사진

자동 생성된 설명

**4-4. 시간/공간복잡도에 대한 고려**

편의를 위해 graph자료구조가 나타내는 그래프의 도시의 개수가 v이고, 경로(간선)의 개수가 e라고 가정하고 서술하겠다.

**<adjacent list와 matrix>**

그래프를 저장하는 방식은 크게 adjacent matrix와 adjacent list가 있다. Adjacent matrix의 경우 두 노드 사이에 간선이 존재하는지 여부를 O(1)의 시간복잡도로 알아낼 수 있다. 그러나 Adjacent list에서는 최악의 경우 O(v)의 시간복잡도가 걸린다. 그러나 matrix의 경우 그래프 저장에 공간복잡도 O(v^2)가 필요하지만 list의 경우 O(v+e)의 공간복잡도가 필요하다.

시간복잡도와 공간복잡도 중 하나를 희생하는 선택이 필요했다. 배달 게임은 플레이어의 가독성을 위해 간선 수가 최대 간선 수보다 적게 배치되도록 개발하였다. 따라서 e가 작기 때문에 adjacent list를 사용하는 것이 효율적이라고 생각했다. 또한 맵의 도시가 늘어났을 때 공간복잡도 O(v^2)가 필요하게 되므로 adjacent list를 선택하는 것이 합리적이라고 판단했다.

**<다익스트라와 일반 탐색>**

게임의 최종 스코어 산출을 위해서는 목적지까지의 최소 Power 소모량을 파악해야 한다. 이때, 단순 중첩반복문이나 DFS, BFS를 반복하여 구하는 것이 가능하지만 시간복잡도 측면에서 비효율적이다. 시간복잡도 O(v^2)를 가지게 된다.

이러한 비효율성을 해소하고자 다익스트라 알고리즘을 사용하였다. 다익스트라 알고리즘의 경우 priority queue와 dist라는 추가적인 배열 메모리가 필요하지만, 시간복잡도 측면에서는 큰 이점이 있다. priority queue를 사용하는 다익스트라의 경우 adjacent list에 대해 queue를 갱신하는 데 log v의 시간복잡도가 소요되므로 총 시간복잡도 O((v+e)logv)를 가진다.

또한 게임의 경우 버퍼링은 큰 마이너스 요소이다. 따라서 조금 메모리를 더 사용하더라도 시간 효율성에 초점을 맞춰 개발하는 것이 옳다고 판단했다.

**5. 창의적 구현 항목에 대한 설명**

**5-1. 맵 파일(txt)를 읽어와서 사용한다.**

일일이 Hard-Coding으로 지도를 그리도록 지정한 것이 아니라, txt에 아래의 형태로 이어져 있는 두 도시의 번호와 소요되는 Power가 적힌 형태의 txt 파일을 읽어와 graph 자료구조에 저장하도록 하였다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

**5-2. 방문했던 도시들, 직전에 방문한 도시, 현재 선택된 도시를 알기 쉽게 표시**

스크린샷, 텍스트, 도표이(가) 표시된 사진

자동 생성된 설명

도시 정보를 색으로 나타내어 플레이어의 가독성을 높였다.

**6. 프로젝트 실행 결과 캡쳐**

<메인 화면>

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

<Instruction 화면>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<게임 플레이 화면>

TIME과 Consumed Power가 표시되는 모습, previous, current에 따라 city의 색이 다른 모습이다.

스크린샷, 텍스트, 도표이(가) 표시된 사진

자동 생성된 설명

서로 인접하지 않은 도시로 이동하려고 한 경우 에러메세지가 뜨는 모습이다.

텍스트, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

<게임오버 후 Result 화면>

텍스트, 스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

**7. 느낀 점 및 개선 사항**

처음에는 각 도시의 위치를 랜덤 배치하는 것으로 개발하였으나, 간선이 겹쳐져서 지도의 가독성에 문제가 생겨서 계획을 수정하였다. 나중에 수학적으로 간선이 겹쳐 보이지 않는 위치의 조건을 구한 뒤에 랜덤 배치한다면 더 다이나믹한 게임이 될 수 있을 것 같다.

혼자서 처음부터 끝까지 게임을 개발해 본 것은 이번이 처음이다. visual studio를 이렇게 만이 써본 것 또한 처음인데, 계속 오류와의 싸움을 하면서 debugging tool을 어떻게 사용해야 하는지 많이 익혔던 것 같다. 전까지는 계속 printf문을 코드 중간 중간에 넣으면서 삽질을 하면서 에러를 찾아냈던 기억이 있는데, debugging tool 사용법을 알고 나서 코딩 속도가 확실히 빨라졌 음을 느낄 수 있었다.