프로그램 컴파일을 위해 필요한 파일

Visual studio에서 구동할 소스 코드와 헤더 파일, 솔루션 파일.

프로그램 컴파일 환경 & 컴파일 방법

Windows 10, Visual Studio community 2017

해당 환경에서 visual studio를 실행하여 컴파일하시면 됩니다.

프로그램을 테스트하기 위한 입력값 & 출력값.

입력값

프로그램 파일 실행 후, cmd창에 입력이 요구될 때마다 선택지에 따라 입력하면 됩니다.

기본적으로 보드게임 부루마블의 턴 진행에 필요한 정보를 입력한다고 생각하면 됩니다.

입력 및 프로그램 진행 순서는 다음과 같습니다.

Sequence 1. 플레이어 명수 입력(초기화)

Sequence 2. 원하는 동작 실행(1~4. 1의 경우 주사위를 굴려 턴을 진행)

Sequence 3. 도착한 지점에 따라 연산 진행 + 땅 매입/건물 건설 등의 선택지 선택.

Sequence 4. 다음 플레이어에 대해 < Sequence 2.> 부터 반복.

출력값

게임 진행상황을 한눈에 볼 수 있는 보드판과, 플레이어가 선택값을 입력해야 할 때마다 선택지를 텍스트로 출력합니다. 또한, 출력한 것에 대한 연산 결과 역시 cmd창에 표시됩니다.

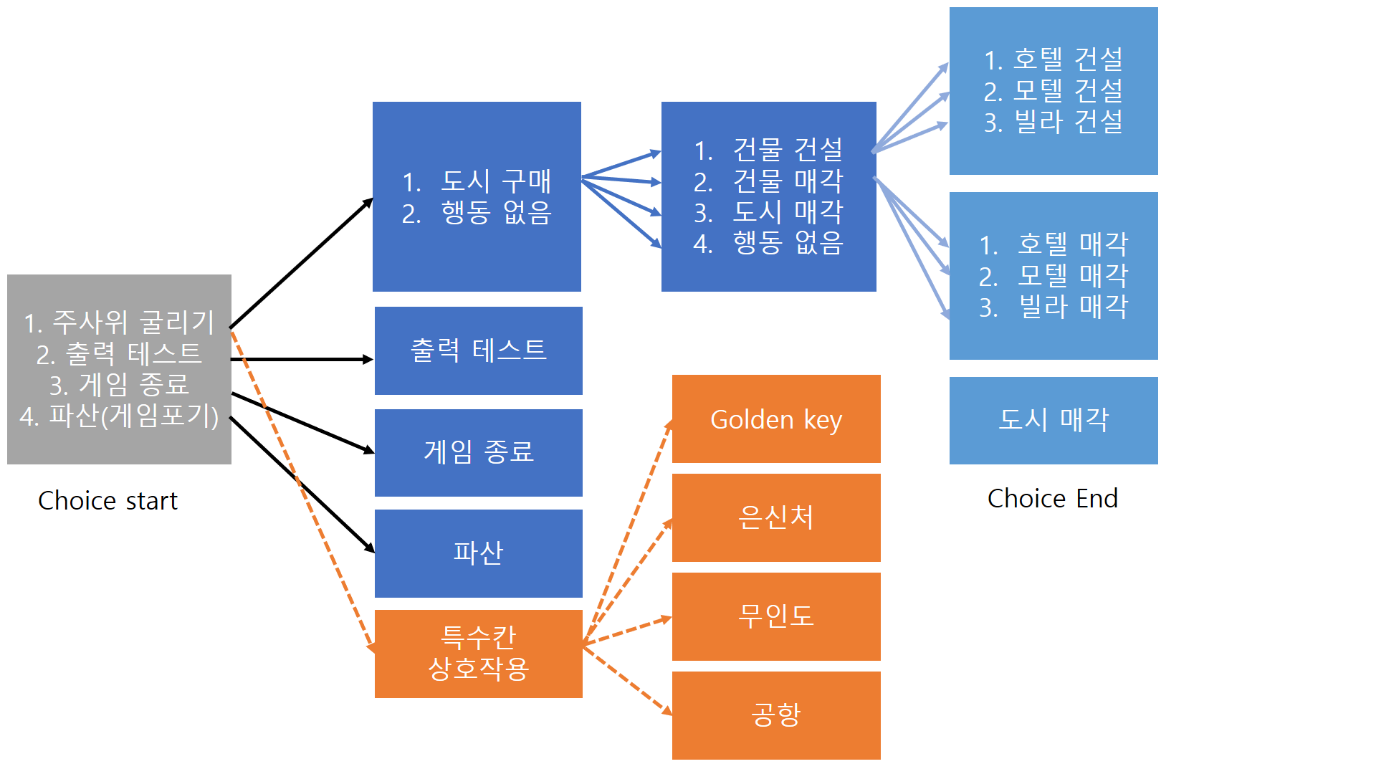
**프로그램명 : Bluemarble by c++**

**작성하고자 한 프로그램의 기능.**

본 프로그램은 한국판 모노폴리라고도 불리는, 부루마불 보드게임을 c++ 문법으로 구현해낸 프로그램이다. 부루마불의 기본적인 룰을 프로그램을 통해서 구현함과 동시에, 각 칸과 플레이어를 클래스로 생각하여 부루마블이 진행되며 발생하는 연산을 클래스 간의 상호작용으로 구현하고자 하였다. 또한, 부루마블의 다양한 특수규칙들 역시 특수한 클래스를 선언하여 기존의 부루마블과 최대한 비슷한 동작을 할 수 있도록 하였다.

**프로그램이 받는 입력**

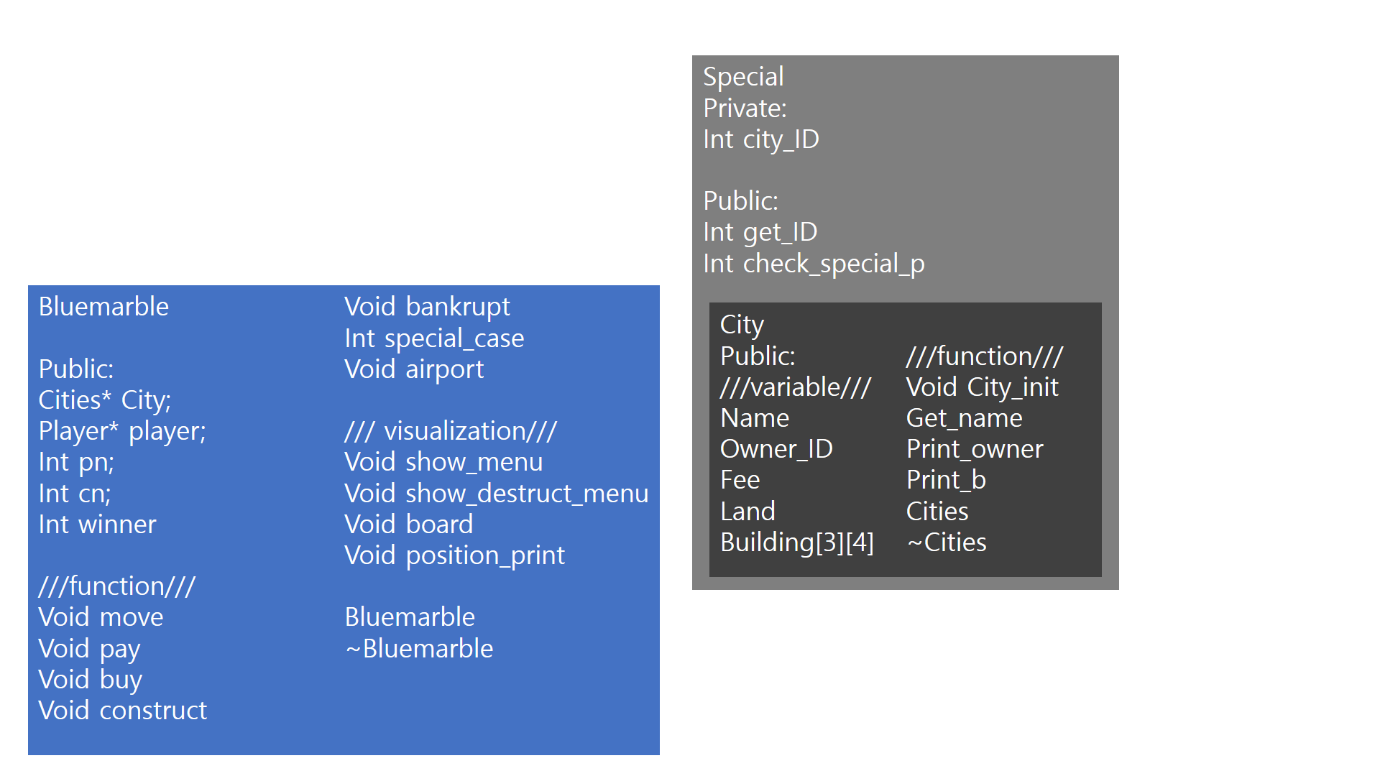
본 프로그램은 실행과 동시에 게임에 참여할 인원수를 입력으로 받으며, 이후로는 부루마블의 턴 진행에 필요한 선택지들에 대한 결정을 입력으로 받게 된다. 가령 Player 1이 턴을 진행하고자 한다면 기본 선택지 1~4 중에 1을 선택하여 주사위를 굴려야 하며, 또한 나온 주사위의 눈 만큼 진행하여 새로운 칸에 도착하면 해당 칸의 성격에 따라 또다른 선택지를 보고 입력을 진행해야 한다. 이를 간단하게 Directed Graph로 나타내면 다음과 같다.



<Figure 01. 선택지에 대한 directed graph>

Figure 01에서 실선은 플레이어가 선택한 결과에 따른 분기를, 점선은 플레이어의 선택에 의해 강제로 진행되는 분기를 뜻한다. 위 그래프에 따르면 각 플레이어는 한 턴에 최대 4번, 최소 1번의 선택을 해야 함을 알 수 있다.

**프로그램 구조에 대한 설명**

본 프로그램은 크게 네 가지의 클래스를 통해 진행된다. 칸의 정보를 저장하기 위한 City class, 플레이어의 정보를 저장하기 위한 Player 클래스, 특수칸의 정보를 저장하기 위한 Special class, 그리고 모든 진행상황을 총괄하는 Bluemarble class가 있다. 프로그램 상에서 입력과 그에 따른 연산은 대게 cin과 switch문을 통해서 이루어지며, class간의 상호작용은 main문에서 동작하는 함수를 통해 구현하였다. Class의 특징과 연관 관계는 다음 Figure에 나타나 있다.

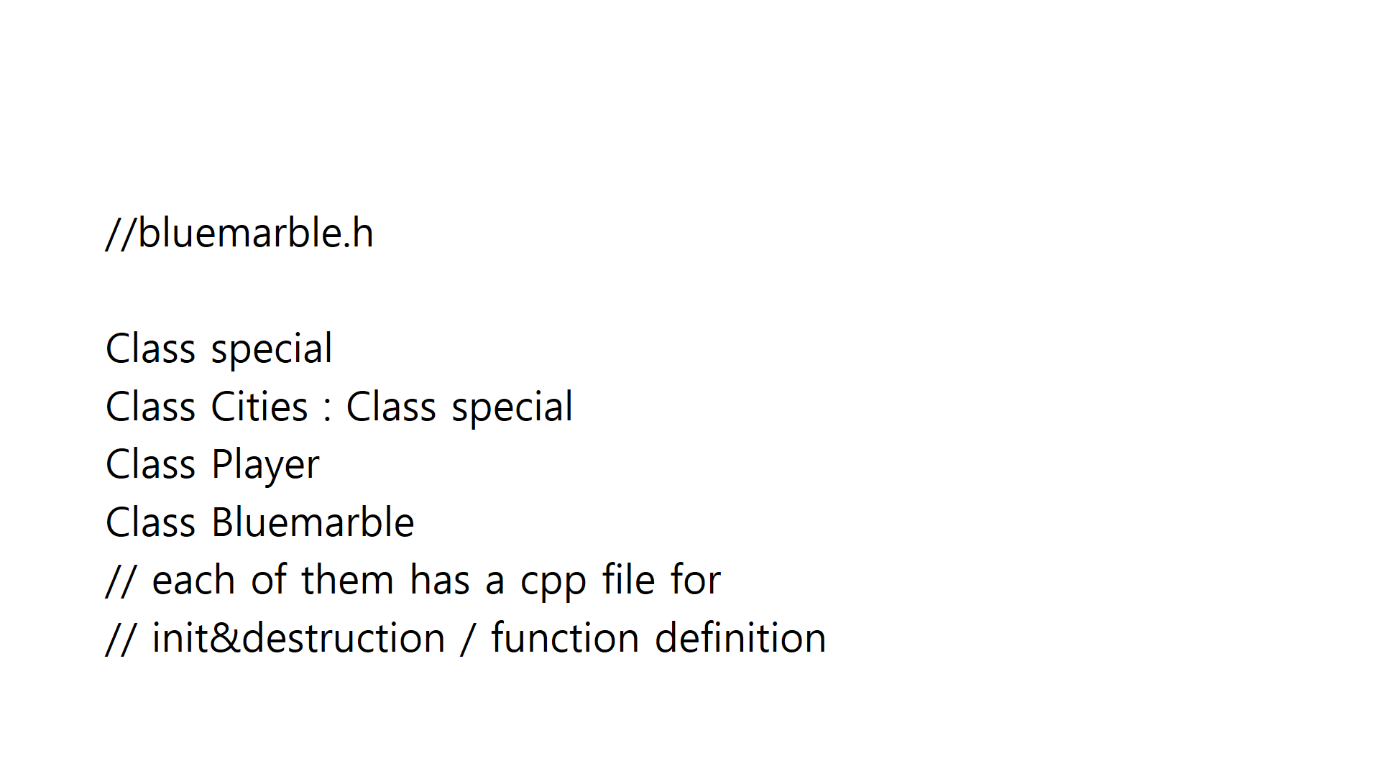


<Figure 02. Relationship and specification of each classes>

각 클래스에 속한 멤버 변수들에 대한 자세한 설명은 Appendix 1.에 자세히 나타나 있다. 본문에서는 기본적인 클래스 간의 관계만을 다루겠다. Bluemarble 클래스에서 player와 city를 클래스 안의 클래스로 선언하며, 블루마블 내부에 선언된 다양한 함수를 통해 각 클래스들에 대한 연산을 진행하는 구조이다.

**주요 알고리즘에 대한 설명**

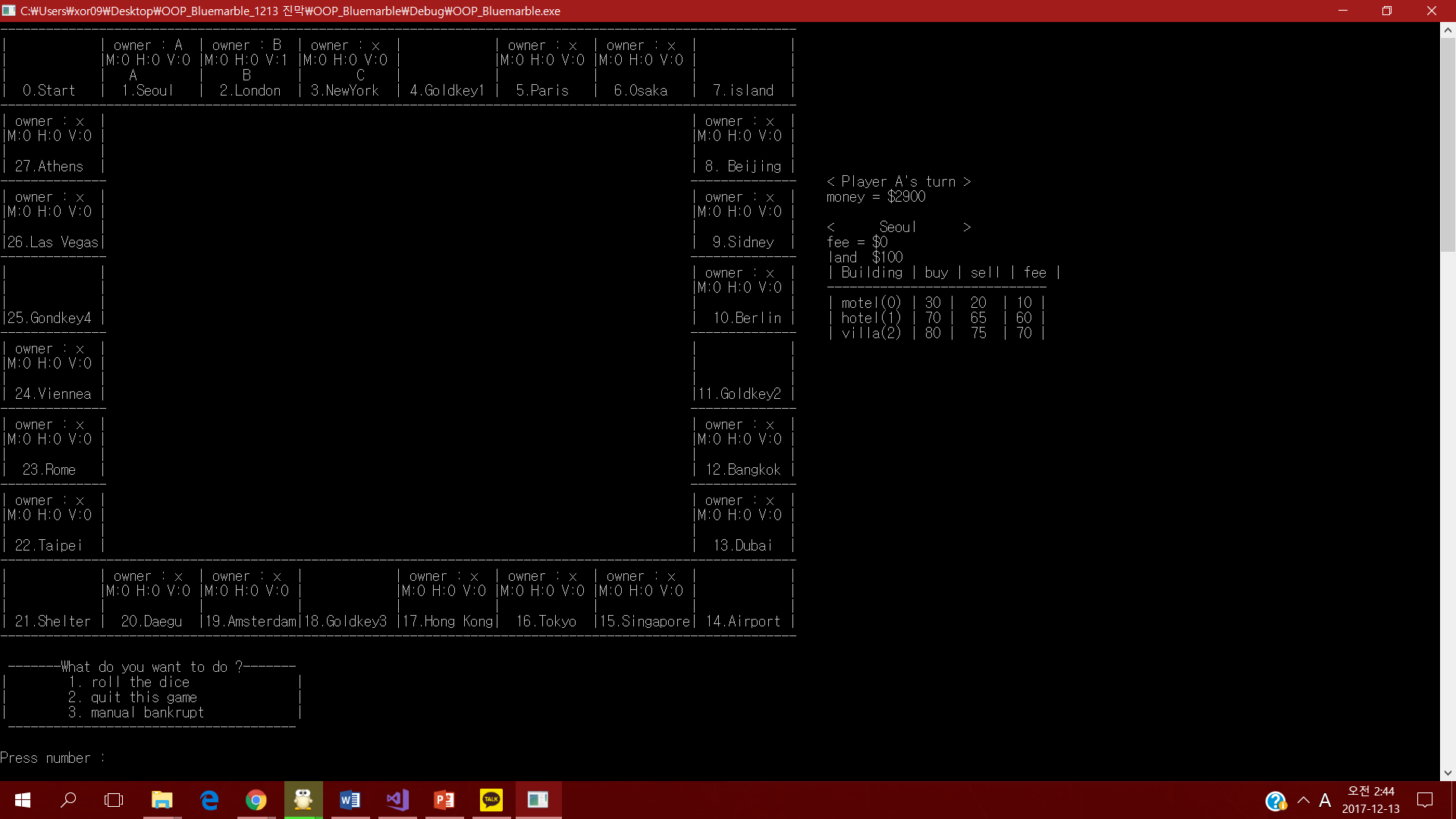
프로그램의 진행 상황을 간략히 표현한 알고리즘은 다음과 같다.



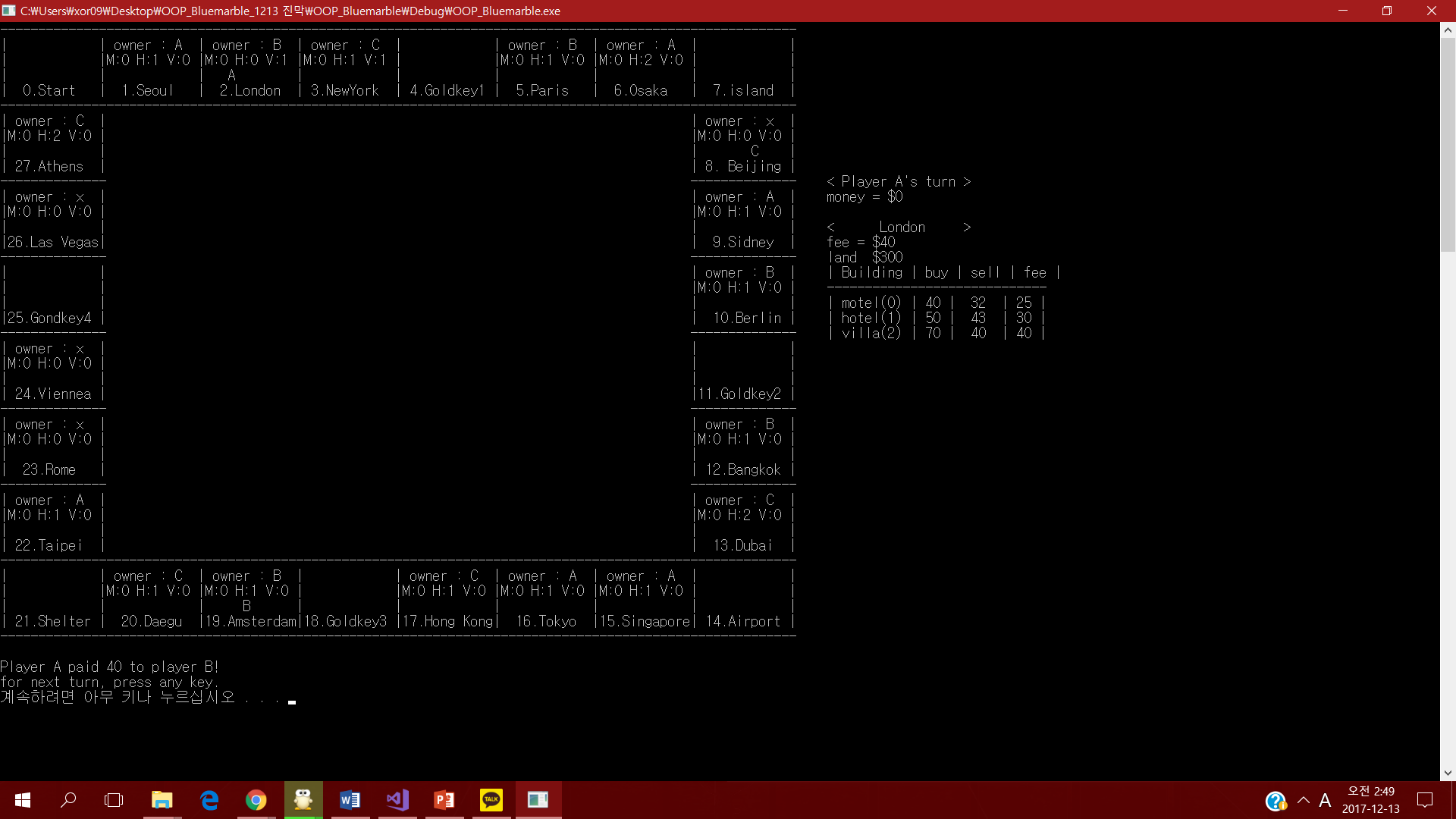
<Figure 03. Algorithm and header for Bluemarble program>

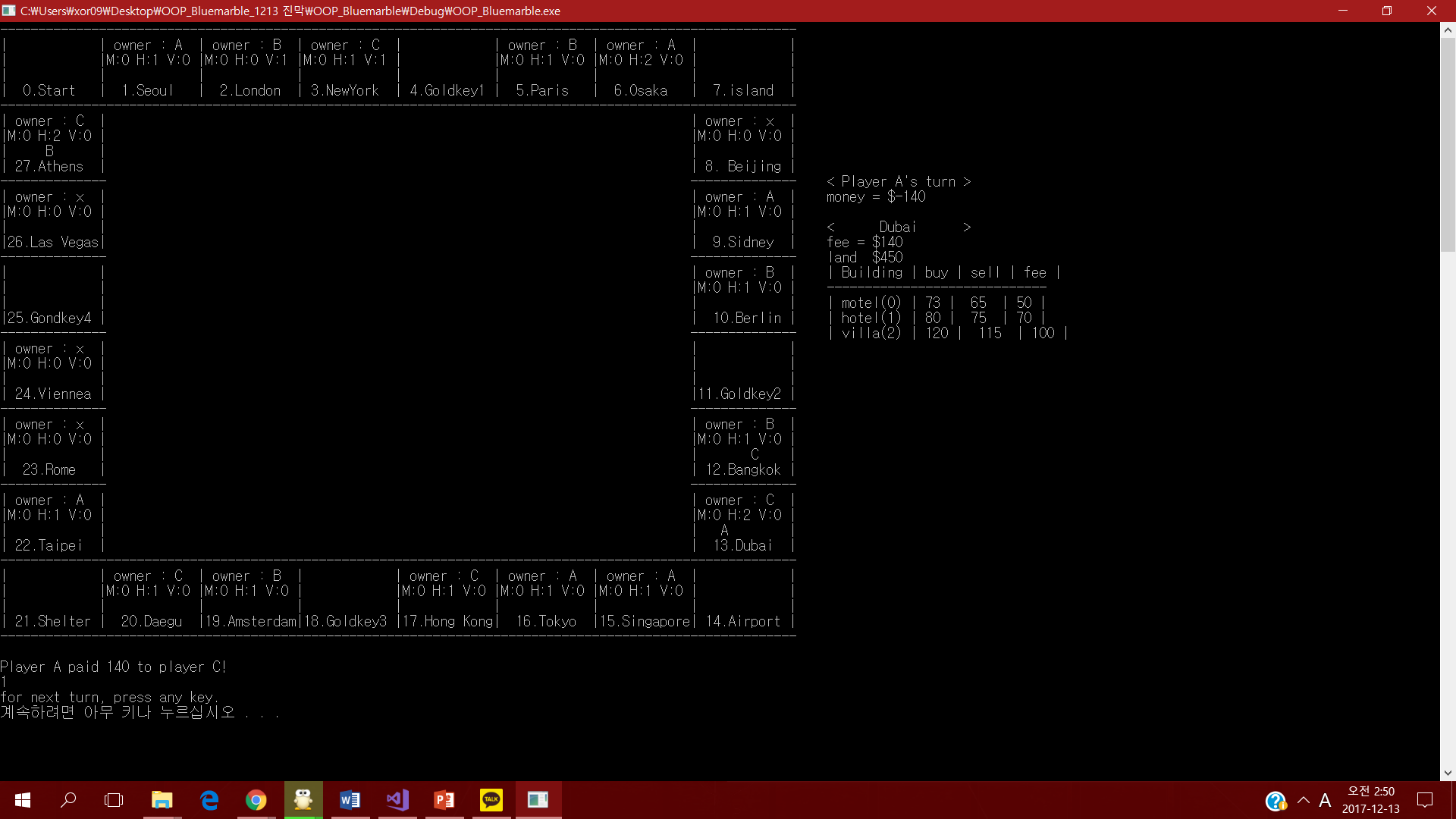
Figure 03에서 볼 수 있듯, 본 프로그램은 한 사람을 제외한 모든 사람이 파산하는 경우를 만족할 때까지 while문 안에서 입력을 받고, 해당 입력을 바탕으로 연산을 진행하는 구조이다. 또한 main문에 사용하는 거의 모든 함수는 bluemarble.h 라는 헤더에 선언되어 있다.

테스트 입력에 대한 설명

각 플레이어가 1턴씩 진행한 뒤의 출력 결과값은 다음과 같다.

또한, 하나의 플레이어가 게임을 떠나거나 파산하였을 경우 출력값은 다음과 같다.





마지막으로, 최종 플레이어가 승리하였을 때의 출력 결과값은 다음과 같다.

**본 프로젝트의 강점과 특징**

1. Class의 사용으로 인한 extensibility

본 프로젝트는 보드게임 중의 하나인 부루마블을 구현한 것으로써, 게임 플레이어와 칸, 그리고 총괄 함수를 각각의 클래스로 선언하였다. 이는 다시 말하면, 부루마블과 비슷한 또다른 보드게임들에 대해서도 동일한 구성으로 프로그래밍을 진행할 수 있음을 뜻한다. 각 클래스 내부의 함수나 변수는 바뀔 수 있지만, 동일한 클래스 구성을 사용하면서 비슷한 종류의 많은 보드게임들을 c++로 구현할 수 있을 것이다.

2. GUI 구성

본 프로젝트는 단순히 실행 결과와 선택지만을 플레이어에게 제시하지 않는다. 이에 추가로, 아날로그 형태의 판을 cmd상에 출력하여 플레이어들이 진행 상황을 한 눈에 알 수 있도록 간단한 GUI를 구현하였다. 이는 본 프로그램을 사용하는 유저들이 쉽게 부루마불을 즐길 수 있게 도와줄 것이다.

3. Level Design

본 프로젝트는 흥미로운 게임 진행을 위해서 몇 번의 offline test를 거쳐, 게임에 관여하는 다양한 변수의 초기값을 설정하였다. 초기화에 포함된 변수는 플레이어의 초기 소지금, 빌딩의 건설과 매각 가격, 특수칸에서 얻거나 잃게 되는 자금 등이 있었다.

[Appendix 1.] Class 내부의 변수, 함수 설명

1. Bluemarble

City : City 객체들을 City 클래스 배열에 저장한다.

Player : Player 객체들을 player 클래스 배열에 저장한다.

pn : 게임에 참가하는 플레이어들의 수를 저장한다.

cn : 게임을 구성하고 있는 도시의 수를 저장한다.

winner : 한 명의 플레이어를 제외하고 모두 파산하면, 남은 한 명의 플레이어는 승리한다.

Move : 플레이어가 굴려서 나온 주사위의 수만큼 위치를 이동시키는 함수.

Pay : 플레이어가 밟은 땅에 통행료를 지불하는 함수

Buy : 플레이어가 밟은 땅이 소유자가 없을 경우, 땅을 구입하는 함수

Construct : 플레이어 소유의 땅에 건물을 건설하는 함수

Bankrupt : 플레이어가 모든 자금을 소비했을 경우, 파산을 선언하고 플레이어를 게임에서 제외시키는 함수.

Special\_case: 무인도랑 은신처에 도착했을 경우, 플레이어에게서 특정 턴수동안 선택권을 뺏는 함수.

이하 시각화 함수는 GUI구현 목적으로 짠 함수이므로, 설명을 생략한다.

2. Special

City\_ID : 해당 칸의 고유 ID

Get\_ID : 칸의 고유 ID를 받아오는 함수

Check\_special\_p : 해당 칸이 특수칸인지 판별하기 위한 변수

3. City(special을 상속받음)

Name : 해당 칸의 이름을 뜻한다(ID와는 별개. Char 형태로 저장. Ex. Paris)

Owner\_ID : 해당 칸을 소유하고 있는 플레이어의 ID

Fee : 해당 칸을 지나갈 때 받는 통행료

Land : 해당 칸의 땅값.

Building : 건물의 정보를 기록할 3X4 array

City\_init : 각 칸을 초기화할 때 사용하는 함수

Get\_name : 칸의 이름을 받아오는 함수

Print\_owner : 칸의 주인 ID를 출력하는 함수

3. Player

player\_ID : 플레이어의 ID를 저장하는 변수.

free : 플레이어가 특수칸(쉼터, 아일랜드)에 있을 때, 그 칸에서 얼마나 쉬고 있는지 세기 위한 변수

own\_num : 플레이어가 소유하고 있는 도시의 수를 저장하는 변수

money : 플레이어가 가지고 있는 돈이 얼마인지 저장하는 변수

int \* own : 플레이어가 가지고 있는 도시들의 ID를 저장하는 정수형 배열

position : 플레이어가 위치하고 있는 도시의 ID를 저장하는 변수

bankrupt : bankrupt 여부를 저장하는 변수

get\_ID() : private으로 선언된 player\_ID를 반환하기 위한 함수