C# 콘솔게임 제작

- Memory of Volfied -

작성일: 2023.07.01

작성자: 배경택

목차

1. 개요

1.1 시작하며

제작된 게임의 코드는 https://github.com/GTAEKI/Volfied.git 에서 확인 할 수 있습니다.

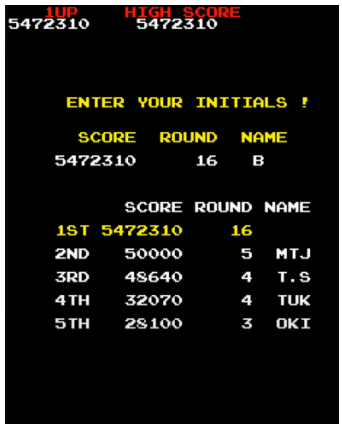
1.1.1 제작 이유

4주간 배운 C# 기본 문법을 게임 제작을 통해 활용하며 친해지기 위하여 제작하였습니다.

1.1.2 제작기간

게임 알아보기: 2023. 06. 23 (금)
게임 제작 성공 가능성 확인(기능 구현이 가능한지 확인): 2023. 06. 24(토)
사전기획서 제작: 2023. 06. 25(일)
게임 제작: 2023. 06.26 ~ 2023. 06. 30

1.1.3 참조할 게임 Volfied 게임 소개



1989년 발매된 땅따먹기 게임입니다.
보스가 초록색으로 된 네모 바닥타일 안쪽에서 지속적으로 움직이며 주인공 캐릭터가 땅에 줄을 그어 땅을 점령합니다.
땅 점령은 땅이 캐릭터가 그린 선에 의해 둘로 나누어질 경우 나누어진 땅 크기에는 상관없이 보스가 없는쪽의 땅이 점령됩니다.
땅은 80%이상 점령되었을 시 게임을 클리어하게 됩니다.
게임 승리 또는 종료 시 점수(스코어)와 이름을 적을 수 있는 화면이 나오며, 아래쪽에 점수(스코어)에 따라 순위가 점수, 이름과 함께 반영됩니다.

2. 개발 환경

2.1 개발 도구 및 언어

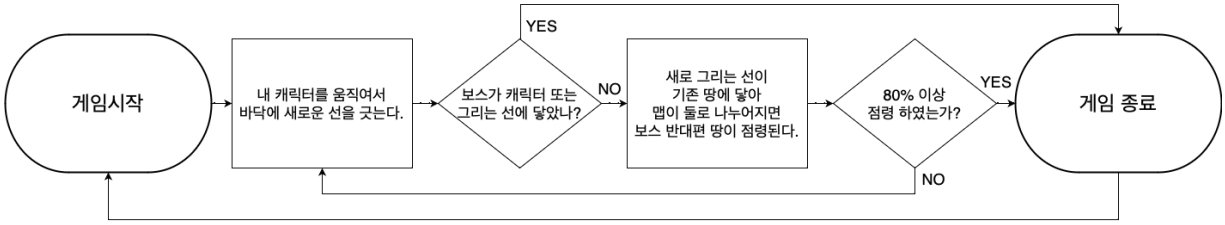
- Visual Studio for mac 2022
- .Net 7.0
- C#
- Newtonsoft.Json 13.0.3

2.2 일정관리 도구

날짜 별 진행 일정 관리: 구글 캘린더(Google Calendar)
기능 구현 관리: 트렐로(Trello)
개발 일지 작성: 노션(Notion)

3. 게임 로직

3.1 플로우 차트



4. 기능 구현 내용

4.1 보스 몬스터

4.1.1 실시간 이동

타이머를 통하여 일정시간마다 움직일 수 있도록 구현하였습니다.

```
// Timer 선언
static System.Threading.Timer timer;

// Main함수에서 타이머 할당
// 처음 10ms가 지나고 100ms마다 MoveMonster함수 실행
timer = new System.Threading.Timer(MoveMonster, null, 10, 100);

// MoveMonster함수
// Timer에서 실행하는 함수의 매개변수는 object타입으로 받습니다.
// 타이머 할당시 null값으로 된 위치에 변수를 주어 object place에 값을 줄 수 있습니다. 이번 코드에서는 따로 활용하지 않고있습니다.
static void MoveMonster(object place)
{
//함수내용
}
```

4.1.2 벽에 부딪힐 경우 반대쪽으로 이동

```
// Enum을 사용하여 보스 움직임을 상수로 표현
// 콘솔창에서 대각선은 90도, 45도 정도 표현이 가능합니다.
// 따라서 45도로 생각할 경우 보스의 움직임은 4가지로 볼수 있습니다.
enum Direction
{
        UPPER_LEFT,
        UPPER_RIGHT,
        BOTTOM_LEFT,
        BOTTOM_RIGHT
}

// enum값을 활용할 bossDirection변수를 지정합니다.
static Direction bossDirection = default;

// for문 2개를 사용하여 배열의 y,x값에 보스 "Ω"가 들어있는지 확인합니다.
for (int y = 0; y < MAP_SIZE_Y + 1; y++)
{
        for (int x = 0; x < MAP_SIZE_X + 1; x++)
        {
                if (mapBasic[y,x]== "Ω")
                {
                  //실행내용
                }
        }
}

// 실행내용, switch문을 사용하여 보스 움직임에 맞춰 내용을 적어줍니다.
switch (bossDirection)
{
        case Direction.UPPER_LEFT: // 실행내용2 break;
        case Direction. UPPER_RIGHT: // 실행내용2 break;
        case Direction. BOTTOM_LEFT: // 실행내용2 break;
        case Direction. BOTTOM_RIGHT: // 실행내용2 break;
}

// 실행내용2, 보스몬스터가 시계방향으로 움직이고 있는중인지 반시계방향으로 움직이는 중인지
확인합니다.
// 부딪히는 위치에 따라 시계방향, 반시계방향을 변경합니다.
```

```
if (IsClockwise == true) // 시계방향일 경우
{
        if (mapBasic[y,x-1] == "▣") // 방향전환 조건
        {
                IsClockwise = false; // 방향 전환
                bossDirection = Direction.UPPER_RIGHT; // 방향 전환
                return;
        }
}
else if (IsClockwise == false) // 반시계방향일 경우
{
        if (mapBasic[y-1, x] == "▣") // 방향전환 조건
        {
                IsClockwise = true; // 방향 전환
                bossDirection = Direction.BOTTOM_LEFT; // 방향 전환
                return;
        }
}
```

4.2 내 캐릭터

4.2.1 입력 W,A,S,D를 통한 이동

```
// key 생성
ConsoleKeyInfo key = default;

//필요한 위치에서 키 입력 받기
key = Console.ReadKey(true);

// if문과 switch문을 이용하여 움직임 구현 (W,A,S,D 공통)
if (key.KeyChar == 'W' || key.KeyChar == 'w') //W키를 눌렀을때
{
        // 실행내용, 4.2.2 참조
}
else if (key.KeyChar == 'A' || key.KeyChar == 'a'){ }
else if (key.KeyChar == 'S' || key.KeyChar == 's'){ }
else if (key.KeyChar == 'D' || key.KeyChar == 'd'){ }
```

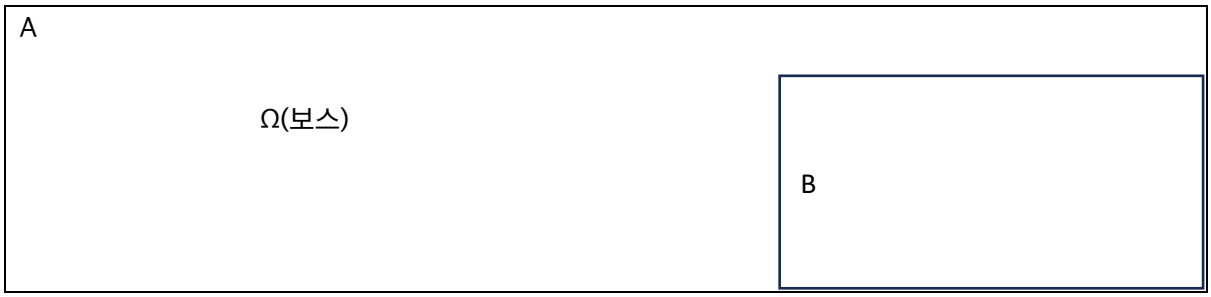
4.2.2 바닥 타일 종류에 따라 캐릭터 이동시 다른 바닥 변화

```
// switch문을 이용하여 이동할 위치의 바닥 모양에 따라서 바닥변화를 주었습니다.
switch (mapBasic[myLocationY - 1, myLocationX]) // 위쪽 한칸 비교(key 입력이 w일때)
{
        case " ": //공백이라면
        // 실행내용
        break;
        case "▣": // 벽이라면
        // 실행내용
        break;
        case "Ω": //보스 몬스터라면
        // 실행내용
        break;
        default: // 예외사항
        // 실행내용
        break;
}
```

4.3 바닥 타일

4.3.1 2개로 나누어진 바닥 타일 중 보스 반대편 바닥타일 점령

Flood Fill 알고리즘을 사용하여 구현하였습니다.(Flood Fill은 그림판에서 색 채우기 기능과 같습니다.)
아래는 기본맵이 새로운 선으로 인해 두개의 박스(A,B)로 나누어진 상황입니다.



보스의 좌표(y,x)값을 기준으로 Flood Fill(채우기)를 실행하게 되면 보스가 있는 위치가 같은 모양으로 채워지게 됩니다. (A는 같은 모양으로 채워지고, B는 공백으로 남아있게 됩니다.)
이때 반전을 주어 A를 공백으로 B를 벽 모양으로 변환한다면 보스 반대편 바닥타일이 점령되도록 구현하였습니다.

4.3.2 Flood Fill 알고리즘

반복문과 비슷합니다. 배열의 범위를 벗어나거나 찾는 모양(originalMark)가 아니라면 반복해서 실행합니다. 반대로 배열의 범위 내이거나 찾는 모양(originalMark)일때 원하는 모양(targetMark)로 변경하는것을 반복하여 채워나가는 알고리즘입니다.

```
// FloodFill 함수
private void FloodFill(string[,] mapBasic, int bossY, int bossX, string targetMark)
{
        string originalMark = " "; // “ ”빈칸을 originalMark로 지정합니다.
                                   // 지정된 빈칸을 아래 식에서 모두 찾아내 변경합니다.
// 시작 위치가 타겟 마크와 동일하거나 시작 위치가 맵 배열 범위를 벗어나면 중지합니다.
        if (originalMark == targetMark || bossX < 0 || bossX >= MAP_SIZE_X
            || bossY < 0 || bossY >= MAP_SIZE_Y)
            return;
// 위 if문을 불만족한다면 Fill함수를 실행합니다.
         Fill( mapBasic, bossX, bossY, targetMark, originalMark);
}

// Fill 함수, 재귀함수로 사용됩니다.
// 재귀함수란 함수 자신의 body에 함수 자신을 호출하여 실행하는것을 말합니다.
private void Fill(string[,] mapBasic, int x, int y, string targetMark, string originalMark)
{
        // 현재 위치가 배열범위를 벗어나거나
        // 현재 위치 모양이 originalMark가 아니면 중지합니다.
        if (x < 0 || x >= MAP_SIZE_X || y < 0 || y >= MAP_SIZE_Y || mapBasic[y, x] != originalMark)
        return;

        // 현재 위치의 모양을 targetMark로 변경합니다.
        mapBasic[y, x] = targetMark;

        // 상하좌우로 재귀적으로 호출합니다.
        Fill( mapBasic, x + 1, y, targetMark, originalMark); // 오른쪽
        Fill( mapBasic, x - 1, y, targetMark, originalMark); // 왼쪽
        Fill( mapBasic, x, y + 1, targetMark, originalMark); // 위쪽
        Fill( mapBasic, x, y - 1, targetMark, originalMark); // 아래쪽
}
```

4.3.3 바닥 타일에 그림 입히기

게임에서 땅을 점령할 경우 뒤에 있는 그림이 보이는 부분을 구현하였습니다.
점령한 부분에 Console.ForegroundColor와 Console.BackgroundColor를 이용하여 색을 입힙니다.
반복하는 작업만 들어가면 되는 부분이며 그림을 위해 색을 어디서부터 입혀야 할지 감이 오지 않으신다면 Microsoft Excel을 이용하면 좋습니다.
Excel에서 프린트할 배열을 표를 통해 y,x를 구현을 한 뒤 색을 입혀놓고 y,x값을 보며 작업을 하면 간편합니다.

4.4 게임 기록 저장 기능

4.4.1 점수에 맞춰 닉네임을 저장하기

크기가 자동으로 변하는 List와 key값을 통해 Value값을 바로 찾아올 수 있는 Dictionary의 장점을 이용하여 구현하였습니다.
한 게임이 끝날때마다 점수(Score)를 List에 추가하여 저장합니다.
이름(Name)을 입력받아 점수(Score)와 함께 Dictionary에 추가합니다.
이때 이름(Name)은 Value값에 점수(Score)는 key값에 넣습니다.
List에 들어간 점수(Score)는 한 게임이 끝날때마다 정렬(Sort)를 이용하여 순위를 지정해 줍니다.

```
//List와 Dictionary 생성
static List<int> scoreRecord = new List<int>();
static Dictionary<int, string> infoRecord = new Dictionary<int,string>();

scoreRecord.Add(score); // List에 score추가
name = Console.ReadLine(); // 이름 입력받기
infoRecord[score] = name; //key값 score에 value값 name 추가
scoreRecord.Sort(); //List 정렬
```

4.4.2 Json 형식으로 텍스트(.txt)파일에 쓰기 및 읽기

Newtonsoft.Json 13.0.3을 이용하여 구현하였습니다.
다른 사용자 컴퓨터에서는 텍스트파일 경로를 변경해줘야 합니다.
콘솔창을 종료하였다가 다시 실행하여도 이전 기록이 남아있습니다.

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

// json파일을 읽어오는 부분입니다.
static string json = System.IO.File.ReadAllText(@"//경로지정/path.txt");
static List<data> _data = JsonConvert.DeserializeObject<List<data>>(json);

//json파일에 입력하는 부분입니다.
_data = new List<data>();
_data.Add(new data()
{
score_ = scoreRecord,
infoRecord_ = infoRecord
});

string json = JsonConvert.SerializeObject(_data.ToArray());
System.IO.File.WriteAllText((@"//경로지정/path.txt",json);
```

```
namespace MemoryOfVolfied
{
    // Json 데이터를 읽고 쓰기위한 클래스
    public class data
    {
        public Dictionary<int, string> infoRecord_ { get; set; }
        public List<int> score_ { get; set; }
    }
}
```

4.5 주의 사항

4.5.1 타이머 사용시 콘솔에 프린트된 이미지 깜빡이는 현상

현재 메인 게임화면의 프린트는 Timer에 계속해서 실행하는 함수 내부에 있습니다.
코드 추가를 하게되면서 메인 게임화면 프린트를 Timer 외부에서 실행할 경우 콘솔창에서 화면이 밀리는 현상이 발생할 수 있습니다.
같은 이유로 SetCursorPoint를 사용을 하게된다면 화면이 밀리는 현상이 발생할 수 있습니다.

메인 게임화면에서만 보스 움직임에 Timer가 사용되고 있으므로 점수판이나 시작화면 프린트는 어디에서 하여도 상관없습니다.