

심화프로그래밍

프로젝트 2

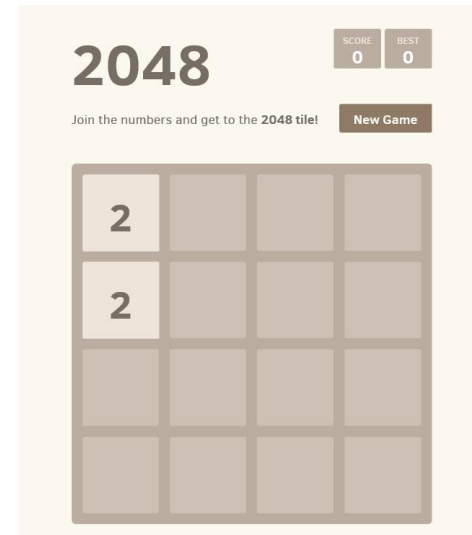
2018학년도 1학기

기말 프로젝트는...

- 중간 프로젝트에서 난이도가 너무 쉽다는 사람과 어렵다는 사람이 있어 살짝 조정했습니다.
- 추가로, 과제 채점은 **결과(코드 구현) 5점, 보고서 점수 5점**을 합산할 계획입니다. (10점 만점)
- 즉, 코드를 제대로 구현을 못해도 보고서를 잘 작성하면 됩니다.
- 포기하지 말고 어떻게 하면 좋을지 생각하고 최대한 구현해 보세요.

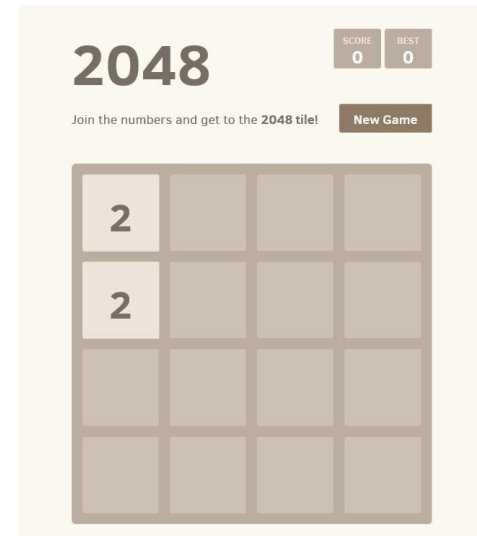
2048

- 게임 판을 상하좌우로 이동시키면 판 위의 블록도 전부 그 방향으로 이동한다.
- 이동하면서 같은 숫자가 있을 경우 합쳐지며, 빈 자리 중 한 칸에 랜덤하게 2 또는 4가 나온다.
- 설명 보다는 직접 해보는 게 이해가 빠르고 쉽습니다.
- <https://gabrielecirulli.github.io/2048/>
- 또는, 나무위키
- [https://namu.wiki/w/2048\(%EA%B2%8C%EC%9E%84\)](https://namu.wiki/w/2048(%EA%B2%8C%EC%9E%84))

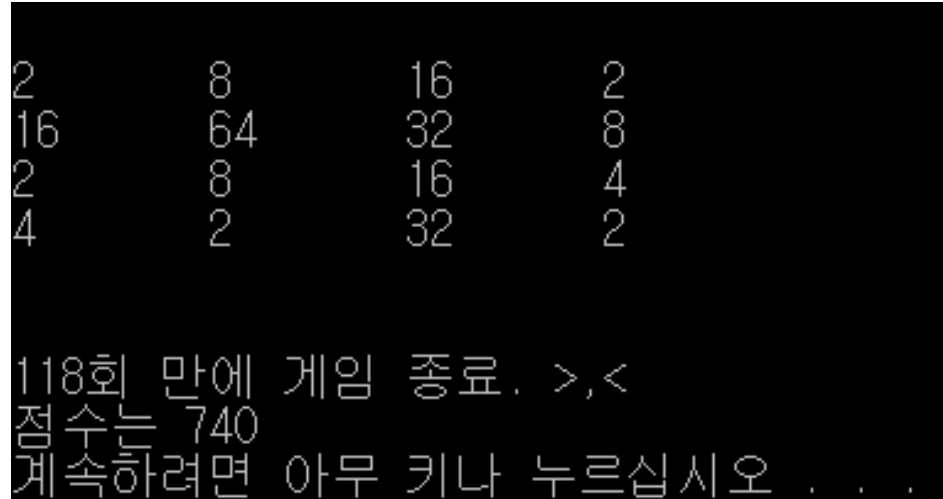


2048

- 점수 계산은 블록이 합쳐져 새로운 수를 만들 때마다 더해집니다.
- 이번 프로젝트는 2048 블록을 만들어도 **게임이 계속 진행됩니다**. 4096, 8192 등 더 큰 수를 만들어 봅시다!



우리가 할 2048



A screenshot of a 2048 game interface. The game board is a 4x4 grid with the following values:

2	8	16	2
16	64	32	8
2	8	16	4
4	2	32	2

Below the board, the following text is displayed:

118회 만에 게임 종료. >,<
점수는 740
계속하려면 아무 키나 누르십시오 . . .

- 게임 판의 상태가 주어지고 4 방향 중 하나를 골라 이동시켜서 가장 큰 점수를 얻는 프로그램(간단한 인공지능)을 구현.
- **중간 프로젝트와 유사합니다.**

우리가 할 2048

- 목표: 최대한 **많은 점수**를 획득. (**큰 수의 블록을 생성**)
- 플레이 방법: 턴 방식
- 한 턴에 이루어지는 행동:
 1. 플레이어는 이동할 방향을 정한다. (상, 하, 좌, 우 중에서 하나)
 2. 게임 판을 이동시킨다.
 3. 이동이 가능한 경우, 모든 블록이 한 방향으로 이동하며 같은 수이면 합쳐진다. 또한, 새로운 랜덤 블록이 생긴다.
 4. 이동하려는 방향으로 이동이 불가능한 경우. 움직이지 않는다.

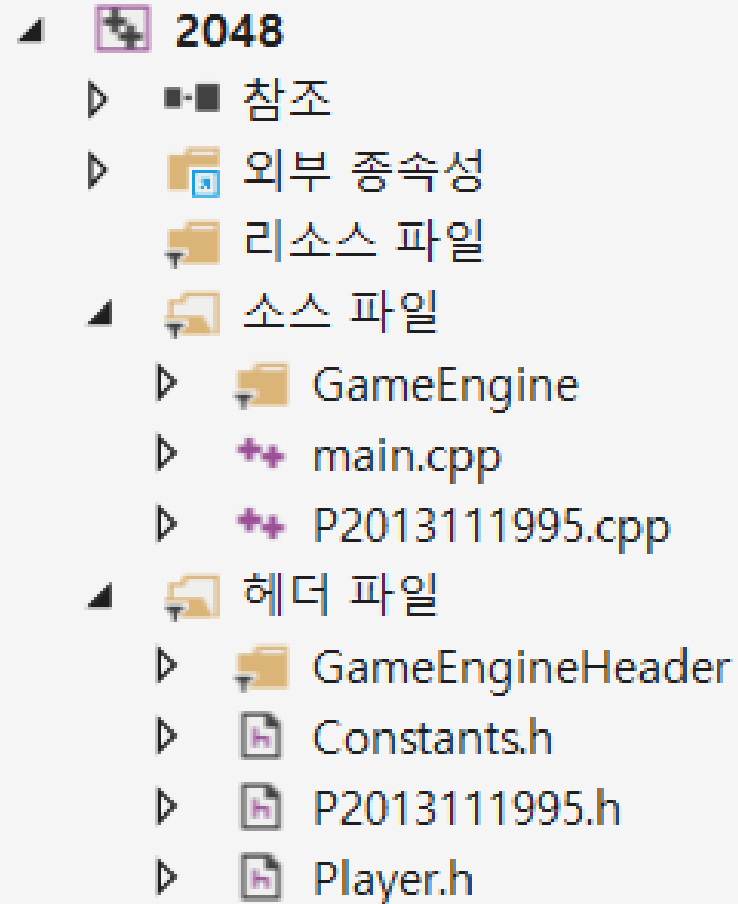
프로젝트 내용

- 2048 게임의 전체적인 틀은 구현되어 있습니다.
- 앞의 룰에 따라 **블록을 크게 만들며 점수를 많이 얻는 코드를 작성하는 것이 목표**입니다!
- **간단한 AI를 구현**하여 챔피언에 도전해봅시다! (큰 점수를 얻으면 좋은 코드.)
- **본인의 생각**을 구현하세요.
- 실제 제출한 과제로 간단한 대회가 치러집니다.

코드를 알아보기에 앞서

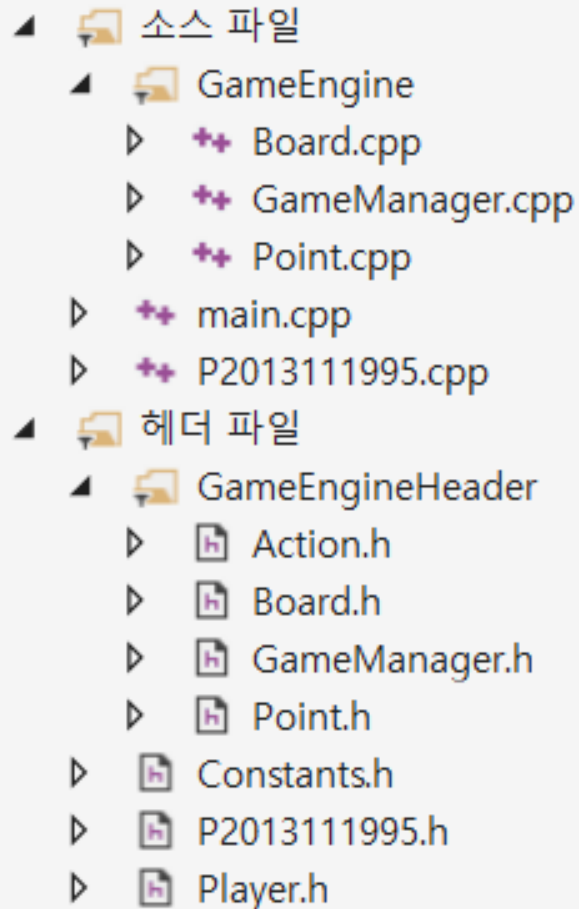
- 전체 코드를 이해할 필요는 없습니다.
- 전체를 몰라도 **어떤 기능을 구현해야 하는지 명확히** 알면 됩니다.
- 내가 직접 2048을 플레이 한다면 어떻게 할지 잘 고려해보세요.
- 어떻게 하면 **조금 더 효율적일지** 생각해보세요.

프로그램 구조



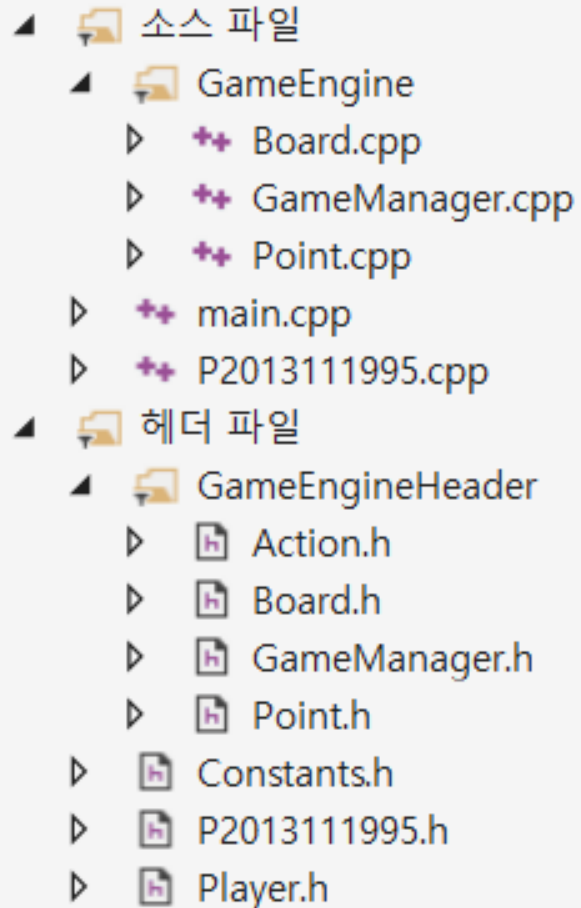
- 크게 두 부류로 나뉜다.
- GameEngine과 Player
- 과제로 구현할 것은 Player를 상속받은 **본인의 Player**입니다.

GameEngine



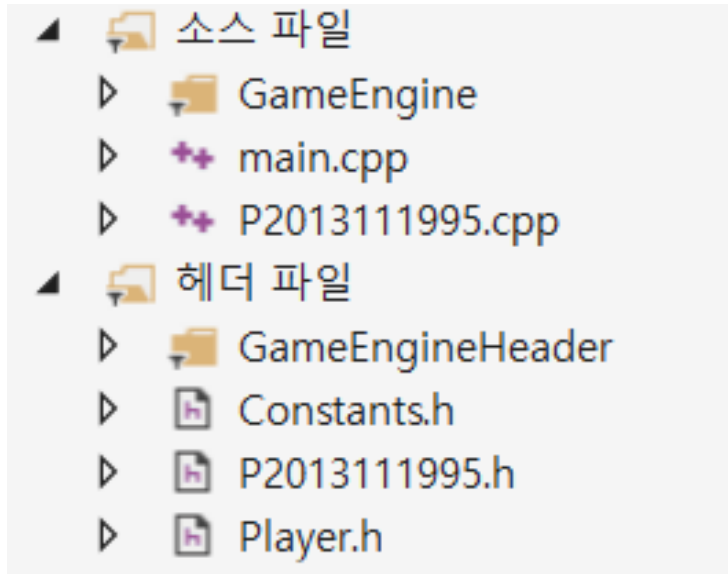
- GameManager: 게임 전체적인 것을 담당.
- main: 메인 함수, GameManager 호출
- Board: 게임 중 판(블록들을 관리)에 대한 것을 담당
- Point: 좌표를 나타내는 클래스.
- Action: 상 하 좌 우로 이동 할 때의 enum

GameEngine



- 코드를 자세히 알 필요는 없습니다.
- 어떤 식으로 돌아가는지만 알면 됩니다.

Player



- Player는 실제 게임을 플레이하는 사용자에게 대한 내용입니다.
- 과제는 Player.h를 상속받아 **자신만의 플레이어 클래스를 생성하는 것**입니다.

자세한 소스코드

- 자세한 내용은 프로젝트 내의 소스코드에 주석을 달아 놓았으므로 생략합니다.
- 전체적인 구조는 중간 프로젝트와 유사합니다. (GameManager가 게임 진행을 관리하고 Board가 실제 게임 데이터를 관리. Player는 게임을 진행.)
- **Player.h만 어떤 식으로 동작하는지 알면 됩니다.**

Constants.h

```
#pragma once
// 상수들. 자유롭게 조정 가능하지만 제출 및 채점 시에는 원래 값으로 진행합니다
// 원래 값: BOARD_SIZE = 4, MAX_TURN = 10000

// 디버그 모드. 한 턴마다 게임 판을 출력한다.
const bool DEBUG = true;
// 게임 판의 크기
const int BOARD_SIZE = 4;
// 최대 플레이 가능한 턴 수.
const int MAX_TURN = 10000;
```

- 디버그 편의를 위해 화면에 출력하는 DEBUG 플래그를 하나 추가하였습니다.
- True로 하면 화면에 매 턴마다 게임 판이 출력됩니다.
- 무한 루프 방지를 위해 턴 수도 제한하였습니다.

Player.h

```
class Player
{
public:
    Player() {}

    // 게임 처음 시작 시 호출됩니다.
    // 현재 게임 판의 상태가 2차원 배열로 주어집니다.
    virtual void gameStart(int board[BOARD_SIZE][BOARD_SIZE]) = 0;

    // 다음에 "움직일 방향"을 반환하세요.
    // 해당 방향으로 이동 시도 후 아래 ableToMove, notAbleToMove 함수 중 하나가 호출됩니다.
    //
    // Action enum안에는 { MOVE_UP, MOVE_LEFT, MOVE_RIGHT, MOVE_DOWN } 가 들어 있습니다.
    // ex) return Action::MOVE_LEFT; // 왼쪽으로 이동
    virtual Action nextMove() = 0;

    // 블록들이 움직였다면 이 함수가 호출됩니다.
    // prevAction: 움직임을 시도한 방향
    // board: 현재 게임 판의 상태. 랜덤으로 한 블록이 생성된 상태입니다.
    virtual void ableToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE]) = 0;

    // 블록들이 움직이지 못했다면 이 함수가 호출됩니다.
    // prevAction: 움직임을 시도한 방향
    // board: 현재 게임 판의 상태. 새로운 블록은 생성되지 않습니다.
    virtual void notAbleToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE]) = 0;
};
```

- Player 클래스를 상속받아 내부를 구현해야 합니다.

P2013111995.h

```
class P2013111995 : public Player
{
public:
    void gameStart(int board[BOARD_SIZE][BOARD_SIZE]);
    Action nextMove();
    void ableToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE]);
    void notAbleToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE]);

private:
    // 디버깅을 위한 문자열로 변환.
    char* Action_Debug[4] = { "MOVE_UP, 위로 이동", "MOVE_LEFT, 왼쪽으로 이동" , "MOVE_RIGHT, 오른쪽으로 이동" , "MOVE_DOWN, 아래로 이동" };
};
```

- 이런 식으로 Player를 상속받아 자신만의 플레이어를 구현하면 됩니다.

P2013111995.cpp

```
#include "P2013111995.h"

void P2013111995::gameStart(int board[BOARD_SIZE][BOARD_SIZE])
{
    // 시작
    //std::cout << "시작" << std::endl;
}

Action P2013111995::nextMove()
{
    // 랜덤
    return (Action)(rand() % 4);
}

void P2013111995::ableToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE])
{
    // 성공
    //std::cout << Action_Debug[prevAction] << "성공" << std::endl;
}

void P2013111995::notAbleToMove(Action prevAction, int board[BOARD_SIZE][BOARD_SIZE])
{
    // 실패
    //std::cout << Action_Debug[prevAction] << "실패" << std::endl;
}
```

- 구현 부분.

주의사항!

- 본 게임의 게임 판(board)좌표는 **아래 방향이 행, 오른쪽 방향이 열**입니다.
- **2차원 배열**을 생각하면 쉽습니다. `board[x][y]`

	4	2	8	0
	0	0	0	0
	2	0	0	4
	2	16	0	0

X BOARD_SIZE

Y BOARD_SIZE

주의사항!!

- **최종 과제 제출 시 입출력은 금지합니다.** (cin, cout, scanf, printf, 등)
- (소스코드 작성 중에 디버깅을 위한 입출력은 가능함.)
- **또한, P학번 코드를 제외한 다른 코드는 수정 금지입니다.**
- 클래스 이름, 헤더파일 이름, 소스파일 이름은 "**P자신의학번**"으로 합니다.
- (ex, 학번이 2013111995일 경우 "P2013111995")
- 헤더 파일과 cpp 파일을 나눠서 둘 다 제출해 주세요.

과제 제출

- 자신이 작성한 헤더파일, cpp 파일(주석 포함)과 간단한 보고서만 제출하세요.
- 프로젝트 코드 전체는 필요 없습니다.
- 주석은 함수 위에 어떤 생각으로 어떻게 구현하였는지 적으세요.
- (ex, /*이러이러해서 저렇게 함*/)

과제 제출

- 보고서는 소스코드와 주석, 실행화면 캡처 (6개), 간단한 소감을 적으세요.
- 실행한 결과 화면은 프로그램을 6번 돌리면 됩니다.

실행 화면 예시

```
C:\WINDOWS\system32\cmd.exe
2      8      2      4
3      64     4      32
4      2      16     4
2      4      32     2

99회 만에 게임 종료. >,<
점수는 648
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\WINDOWS\system32\cmd.exe
4      16     4      2
32     4      64     4
3      128    32     16
2      4      16     4

179회 만에 게임 종료. >,<
점수는 1460
계속하려면 아무 키나 누르십시오 . . .
```

- 랜덤으로 4방향을 이동하는 코드 실행 결과.
- 128 블록을 가끔 만들기는 한다. 생각보다 효율이 괜찮다.
- 잘 생각해서 큰 수의 블록을 만들어 보세요!
(힌트: 나무위키?)

질문 사항이 있다면

- 조교 이메일
- zeikar@naver.com
- 연구실
- 신공학관 5112호
- 감사합니다.