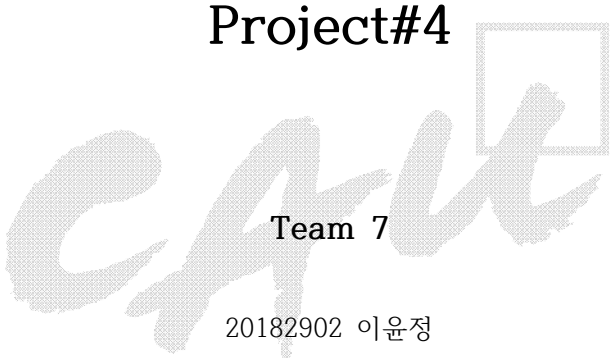
Object-Oriented Programming

Project#4



Team 7

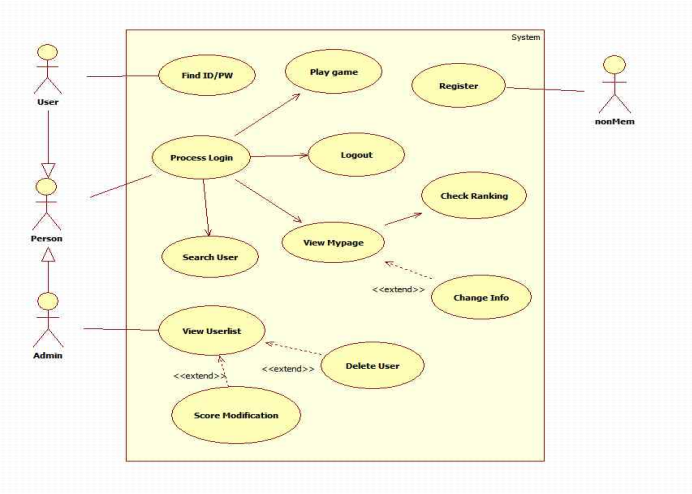
20182902 이윤정
20185536 박종혁
20172990 강혜연

Table of Contents

1. Topic

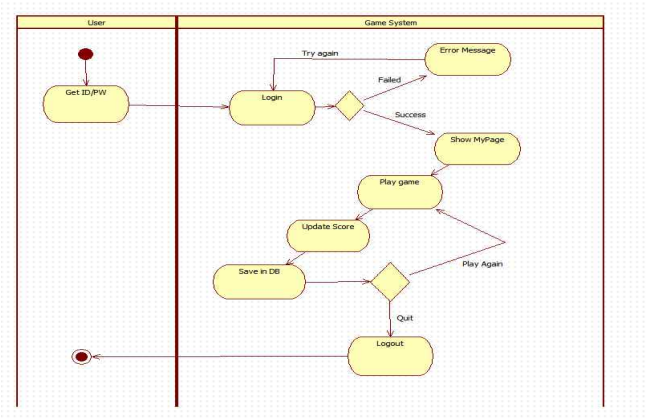
We aimed to make mini games out of many topics, and developed them based on the game 'brain wars' in the google app store. While implementing each mini games, we addressed object-oriented elements; reusing codes by inheritance and expand programs to new mini games from upper class. Also we tried to increase the complexity of the program by adding login and membership management systems.

2. Initial Design Structure



<Use-Case Diagram>

The diagram is a high-level design of the program that was created prior to the real implementation, which briefly describes the functions of the actor, user and admin, logging in to play the game, to check personal information, modify their information, check rankings, search users, etc. within the program, or to register a new user.



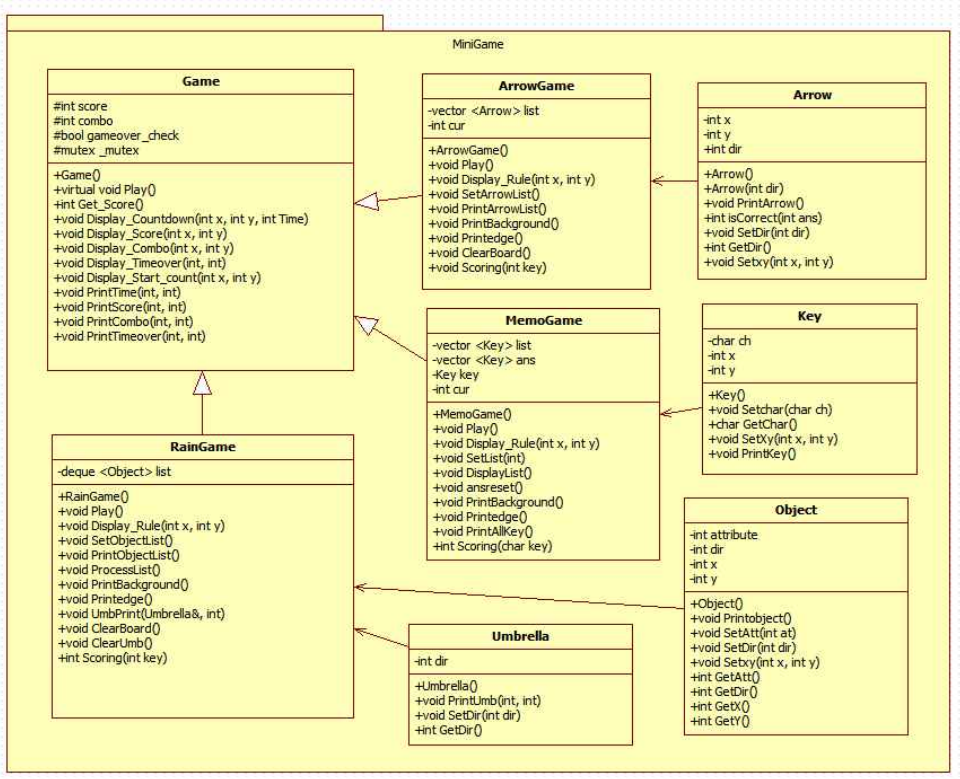
<Activity Diagram>

This diagram indicates how the program works when the user logs in and starts the game. If the user enters the correct ID and password, login succeeds, otherwise retry is recommended with an error message. After the user plays the game, score will be updated and saved in the database. Then the user chooses whether to play the game again or quit, and logs out at the end.

3. Actual Implemented Program

The number of mini games is set at 3, and we named each game as 'Arrow Game', 'Memo Game', and 'Rain Game'. There are two ways to play a game: 'Practice Game' mode and 'Play Game' mode. The 'practice game' is a mode in which the user can select and practice each of the three games, and in the 'Play Game' mode, the three games appears in a random order, and the final scores will be saved. The game only runs for a set period of time and must be finished when the time limit is over.

We implemented the list of the general users, except the admin user, and added function to add new users or check the rankings of user themselves. The user can also check their information from 'My Page'.



<Class Diagram>

4. Code Details

1) General Outline

The class 'Game' will be the parent class of the class ArrowGame, MemoGame, and RainGame, and the class will set the overall function. Each game starts with a virtual function Play().

There is a time limit for the game, and when the time is over, the game is over. Therefore we have to check and print the time continuously while getting inputs from user, so we used thread. Generated thread calculates and outputs time within the loop, and if the time exceeds the specified time, it checks if the game is over and exit the repeating statement. This applies in common to all 3 games.

2) Game

```
29      enum { UP = 72, DOWN = 80, LEFT = 75, RIGHT = 77 }; //키보드 입력 처리
30
31    class Game
32    {
33    protected:
34        int score; //게임 점수
35        int combo; //게임 콤보
36        bool gameover_check; //Time over 체크를 위한 변수
37        mutex _mutex; //스레드 동기화에 사용
38
39    public:
40        Game() : score(0), gameover_check(0), combo(0) { }
41        virtual void Play() {}
42
43        int Get_Score() { return score; }
44
45        /*----------게임UI 출력함수----------*/
46        void Display_Countdown(int x, int y, int Time);
47        void Display_Score(int x, int y);
48        void Display_Combo(int x, int y);
49        void Display_Timeover(int, int);
50        void Display_Start_count(int x, int y);
51        void PrintTime(int, int);
52        void PrintScore(int, int);
53        void PrintCombo(int, int);
54        void PrintTimeover(int, int);
55    };
```

- Member Variables:
  score: Score of the game
  combo : Combo of the game
  gameover_check : verifies if time limit has been exceeded, The default value is 0, and changes to 1 when the time exceeds.
- Member Functions:
  Various basic functions that are used throughout whole game

3) ArrowGame

```
class ArrowGame : public Game
{
    #define ARROW_SIZE 9 //Arrow 개수
    const int TIME_LIMIT = 20; //시간제한 설정

    class Arrow
    {
    private:
        int x;
        int y;
        int dir;

    public:
        Arrow() : dir(0) {}
        Arrow(int dir) { dir = dir; }
        void PrintArrow(); //화살표 출력
        int isCorrect(int ans);
        void SetDir(int dir) { this->dir = dir; }
        int GetDir() { return dir; }
        void Setxy(int x, int y) { this->x = x, this->y = y; }
    };
```

```
private:
    vector <Arrow> list; //화살표를 순서대로 담을 벡터
    int cur; //사용자의 입력커서 위치

public:
    ArrowGame() : cur(-1) {}
    void Play();
    void Display_Rule(int x, int y);

    void SetArrowList(); //화살표 랜덤생성 & 벡터에 PUSH
    void PrintArrowList(); //list에 있는 화살표 출력

    void PrintBackground();
    void Printedge();
    void ClearBoard();

    void Scoring(int key); //점수계산
};
```

- Member Variables:

  x, y : coordinates

  dir : direction

  vector <Arrow> list : Vector that contains the output arrows

  cur : Position of current cursor among vectors

- Member Functions:

  SetArrowList() : Generate random arrows in size of ARROW_SIZE, and put in the list

  ClearBoard() : Function to delete specific part of the console

  void Scoring() : Compare the input key with Arrow in the list[cur]

- Operation

  Generate 9 arrows randomly and put in the list. Check that the direction of the arrow in the cur of the list and input value is same, and if true, increase the curs. If cur==9, newly generate arrow and repeat the work so far.

4) MemoGame

```
class MemoGame : public Game
{
    const int TIME_LIMIT = 30;

    class Key
    {

    private:
        char ch;
        int x, y;

    public:
        Key() : ch(0), x(0), y(0) {}
        void Setchar(char ch) { this->ch = ch; }
        char GetChar() { return ch; }
        void SetXy(int x, int y) { this->x = x, this->y = y; }
        void PrintKey();
    };

private:
    vector <Key> list; //하이라이트 될 KEY가 저장된 벡터
    vector <Key> ans; //사용자가 입력한 KEY가 저장된 벡터
    Key key[KEY_SIZE]; //전체 출력될 KEY
    int cur; //현재 사용자의 입력커서 위치

public:
    MemoGame() :cur(0) {}
    void Play();
    void Display_Rule(int x, int y);
    void SetList(int); //하이라이트 될 KEY 랜덤 설정
    void DisplayList(); //KEY 하이라이트
    void ansreset(); //사용자 입력 초기화

    void PrintBackground();
    void Printedge();
    void PrintAllKey();

    int Scoring(char key); //점수계산
};
```

- Member Variables:

  ch : character for keyboard

  x, y : coordinates

  vector <Key> list : Vector for the highlighted key

  vector <Key> ans : Vector for the input key from the user

- Member Functions:

  PrintAllKey(); Print all keys, only called at initial state

  SetList(); Randomly push the key that will be highlighted

  DisplayList(); Highlight the key sequentially

  ansreset(); Change the colors of the input key to the original color

- Operation:

  Create keys to be used and print them out in keyboard shape. Then, the setList function randomly generates the index of the key to be highlighted and puts the key in the list. Next, highlight sequentially and receive input from the user. If the user make a mistake during the game, the set the list newly and print. Logic is similar to above-mentioned ArrowGame.

5) RainGame

```
6   class RainGame : public Game
7   {
8       enum { rain = 1, light = 2 };
9       enum { center = 1 };
10      const int TIME_LIMIT = 20;
11
12      class Object { ... };
34
35      class Umbrella
36      {
37      private:
38          int dir;
39
40      public:
41          Umbrella() : dir(0) {}
42          void PrintUmb(int, int);
43          void SetDir(int dir) { this->dir = dir; }
44          int GetDir() { return dir; }
45      };
46
47  private:
48      deque <Object> list;
49
50  public:
51      RainGame() {}
52      void Play();
53      void Display_Rule(int x, int y);
54      void SetObjectList(); //OBJECT 랜덤 생성
55      void PrintObjectList(); //list에 있는 OBJECT 출력
56      void ProcessList(); //front의 OBJECT POP & 새로운 OBJECT PUSH
57
58      void PrintBackground();
59      void Printedge();
60      void UmbPrint(Umbrella&, int);
61      void ClearBoard();
62      void ClearUmb();
63
64      int Scoring(int key); //점수계산
```

- Member Variable:
  attribute : Identify whether it is a water drop or a lightning bolt
  deque <Object> list : deque containing the object to be printed.

- Member Functions:
  SetObjectList(); Create and list objects randomly, called at initial state
  ProcessList(); If correct, pop on deque and push new object

- Operation:
  A total of 4 objects are randomly generated (water drop probability of 66.6%, lightning probability of 33.3%) and put them in the list and print. Get input from keyboard and compare it with the object in front of the current list. If it is true, repeat the processList function to remove the front value and insert a new object.

4) GameRun

```
class GameRun {
private:
    std::vector<UserInfo>::iterator nowUser; // 현재 게임을 하고 있는
public:
    GameRun();
    GameRun(std::vector<UserInfo>::iterator copy);

    std::vector<UserInfo>::iterator findUser(std::string ID);
    void Start(int menu);
    bool Login();
    void mainMenu();
    void RealGame();
    void PracticeGame();
    void showRank();
    void myPage();
};
```

- Member Functions:

login(): called at the start of the progam, if success, return true. If false, retry login if input is 'n', or exit the program if the input is 'y'.

mainMenu(): execute program until the user input is '5. exit'

RealGame(): Randomly execute 3 games and save score

practiceGame(): Choose the game in practice mode

showRank(): If you choose a game to show by rank, print the userlist in a descending order. Repeat until the user selects the mainmenu

mypage(): show user id, score, ranking

5) Main

```
1   //#include <iostream>
2   #include "GameRun.h"
3   #include <windows.h>
4
5   int main()
6   {
7       int menu=0;
8
9       system("title Random Game");
10      while (menu != 1 && menu != 2) {
11          system("cls");
12          std::cout << "------- game start ----------\n"
13              << "1. Login.\n"
14              << "2. Create ID.\n"
15              << ">> ";
16          std::cin >> menu;
17          if (!cin) {
18              std::cin.clear();
19              std::cin.ignore(INT_MAX, '\n');
20          }
21      }
22
23      GameRun gr = GameRun();
24      gr.Start(menu);
25      return 0;
26  }
```

If login success and the game starts, 'info.txt' is saved to userlist. If you generate new ID, it will be saved in 'info.txt'

6) UserInfo

```
class UserInfo
{
private:
    std::string userID;
    std::string userPW;
    int scoreArrow;
    int scoreMemo;
    int scoreRain;

public:
    UserInfo(std::string newID, std::string newPW);
    UserInfo(char* originalData);

    std::string getID() { return userID; }
    std::string getPW() { return userPW; }
    bool checkIDPW();
    int getScore(int sel);
    void updateScore(int newScore,int gnum);
    void scorePrint(int newScore, bool save);

    bool operator==(const std::string a)
    {
        if (this->userID.compare(a) == 0) return true;
        else return false;
    }
    static bool comp_arrow(const UserInfo& a, const UserInfo& b);
    static bool comp_memo(const UserInfo& a, const UserInfo& b);
    static bool comp_rain(const UserInfo& a, const UserInfo& b);
};
```

Member Function:

updateScore(): If the result is higher than the existing score, update the value.

scorePrint(): Print the total score with the information of whether it is renewed or not at the end of the game.

5. Execution Result

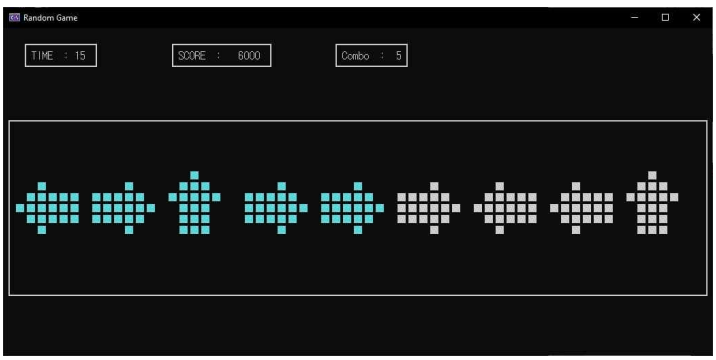
- Development Environment: Window Visual Studio
- Execute by project4_v1.exe, or may directly compile from VS studio
- (Path: proj4_ > Debug > project4_v1.exe)

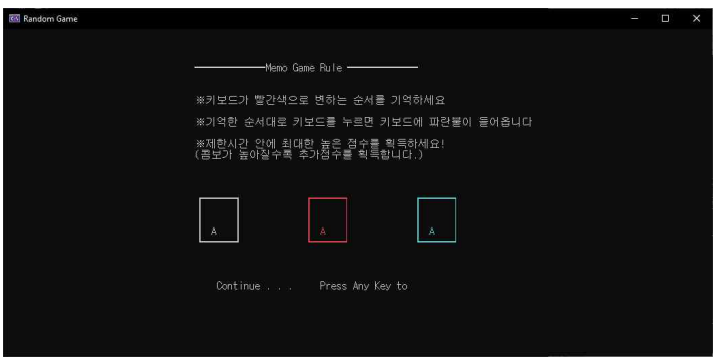
1) Start Screen

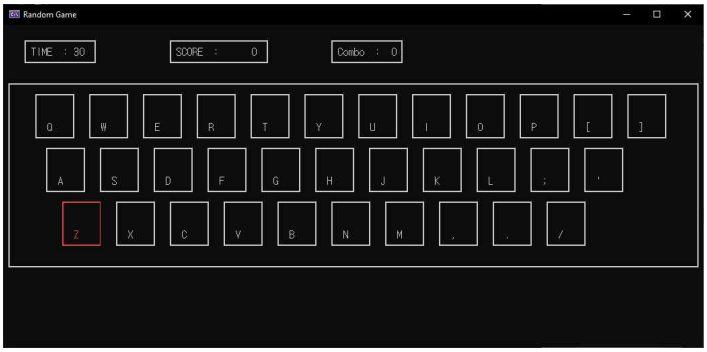
2) Create ID

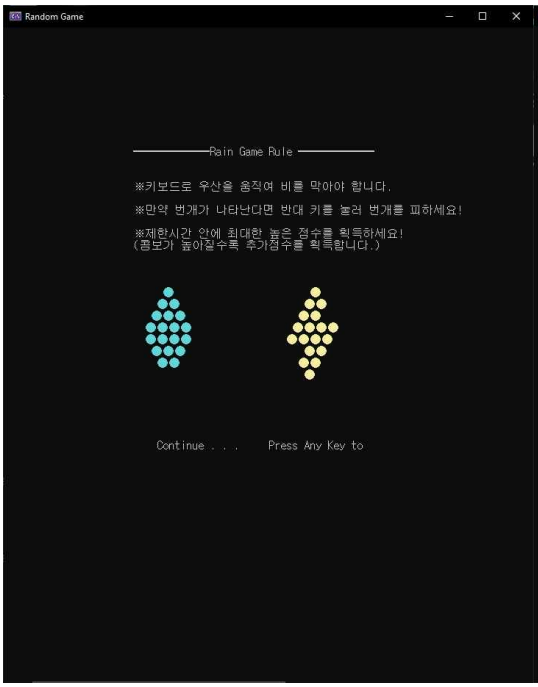
3) Main Screen of the game

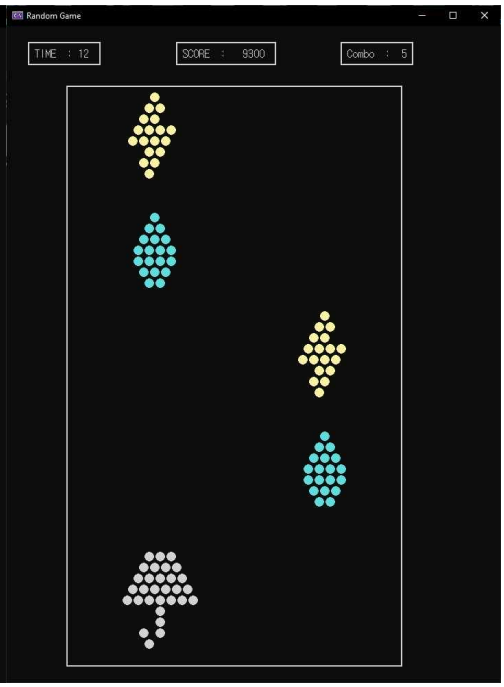
4) Practice Game



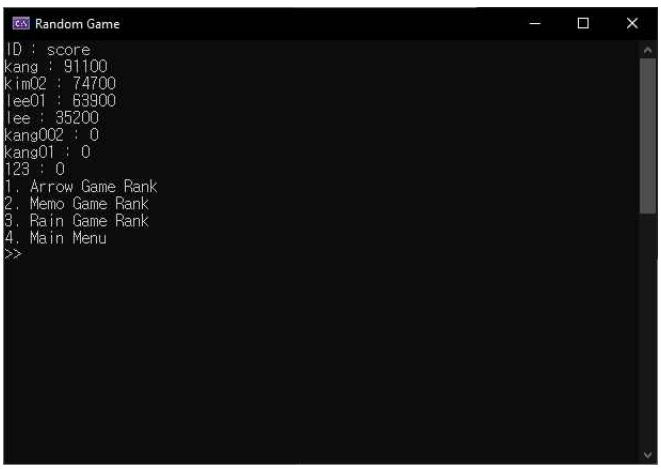








4) Show Ranking



6. Conclusion

We've come to complete a project that's somewhat smaller than originally planned, nevertheless has high completeness and attractive.

By designing the project above, we have come to understand the overall structure of the object-oriented design. And as we proceed in detail, we had a deeper understanding of object-oriented programming through application of class inheritance, reusing codes and expanding programs to new mini games from upper class; relationships between inner classes, and class-level collaboration.