HOMEWORK 5 REPORT

AHMET USLUOGLU

1801042602

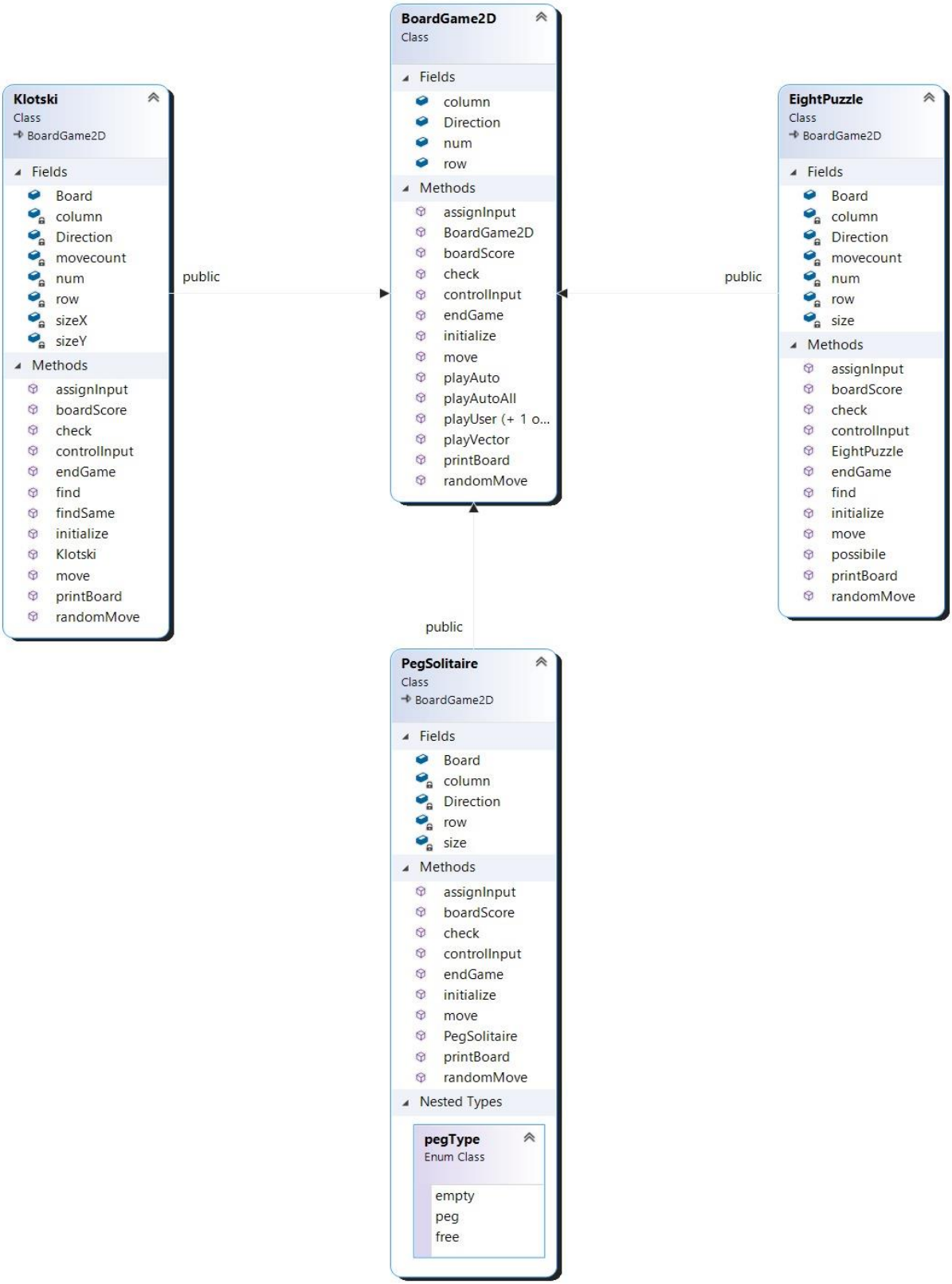
Creating a Program That Plays 3 Different 2D Console Games Automatically and Manually.

Peg Solitaire

Eight Puzzle

Klotski

CLASS DIAGRAM



BoardGame2D Class

```
class BoardGame2D
{
    public:

        int row , column ;
        char Direction, num;

        BoardGame2D(/* args */);

        void virtual playUser() final;
        void virtual playUser(string input);

        void virtual playAutoAll() final;
        void playAuto();

        friend ostream & operator<<(ostream& out, BoardGame2D & board);

        virtual bool endGame() = 0;
        virtual void printBoard(ostream& out) = 0;
        virtual void initialize() = 0;
        virtual int boardScore() const = 0;

        virtual void randomMove() = 0;
        virtual int check() = 0;
        virtual int move() = 0;

        static void playVector(vector<BoardGame2D*>);

        virtual int controlInput(string input) = 0;
        virtual void assignInput(string input) = 0;


};
```

->BoardGame2D Class contains necessary functions for playing 2D games.

->BoardGame2D class is parent to 3 child classes.

->Peg Solitaire, Eight Puzzle and Klotski Classes inherit BoardGame2D class.

->PlayUser(String)function plays the game for one move based on the user input.

->PlayUser function plays the game until the end based on user input.

->PlayAutoAll function plays the game automatically until the end.

->PlayAuto function plays the game for one move automatically.

-> Pure virtual functions in this class are implemented in the child classes.

->PlayVector function plays the games in the BoardGame2D pointer vector until the end.

```cpp
BoardGame2D :: BoardGame2D(/* args */){};

//Plays the game until it's finish
void BoardGame2D :: playAutoAll()
{
    while (endGame()) {playAuto();}
    printBoard(cout);
}
//Plays The game for one move
void BoardGame2D ::playAuto()
{
    while (1)
    {
        randomMove();
        if(check())
        {
            move();
            //printBoard(cout);
            break;
        }
    }
}
//plays the Game according to user input until it's finishes
void BoardGame2D ::playUser()
{
    string input = "";
    while (endGame())
    {
        getline(cin,input);
        playUser(input);
    }
    cout << "Game is ended. Your Score is " << boardScore() << endl;
}
//Plays the for one move based on user input
void BoardGame2D ::playUser(string input)
{
    if (controlInput(input))
    {
        assignInput(input);
        if(check() == 1)
        {
            move();
            printBoard(cout);
        }
        else cout << "Invalid Move! Try Again. \n";
    }
    else cout << "Invalid Input! Try Again. \n";
}
// Plays all games inside the vector
void BoardGame2D :: playVector(vector<BoardGame2D*> games)
{
    for(auto game : games) game->playAutoAll();
}
// Stream insertion operator overload
ostream & operator<<(ostream& out, BoardGame2D &board) ···

```

->EndGame checks the game is ended or not ,

EndGame Functions

```
// Checks all pegs if there is any valid move or not, returns true if there is valid move
bool EightPuzzle :: endGame()
{
    bool flag = false;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if((i + j != 4) && Board[i][j] != '0' + ((i*3) + (j+1))) flag = true;
        }
    }
    return flag;
}
```

```
// Checks all pegs if there is any valid move or not, returns true if there is valid move
bool Klotski :: endGame()
{
    if(Board[3][1] == '2' && Board[3][2] == '2' && Board[4][1] == '2' && Board[4][2] == '2') return false;
    else return true;
}
```

```
// Checks all pegs if there is any valid move or not, returns true if there is valid move
bool PegSolitaire :: endGame()
{
    int end = 0;
    vector<char> directionArray {'U','D','R','L'};
    char direction;

    for(int i = 0; i < size; ++i)
    {
        for(int j = 0; j < size; ++j)
        {
            for(int k = 0;k < 4; ++k)
            {
                row = i, column = j, Direction = directionArray[k];
                /*If ther is a possible move Check returns 1 game is not ended, else endGame equals 0*/
                end = check();
                /*If there is any valid move Game is not ended*/
                if(end == 1)
                {
                    end = 1;
                    i = size;
                    j = size;
                    k = 4;
                }
            }
        }
    }
    /*If no valid move is possible endGame returns 0 by default*/
    return end;
};
```

PrintBoard prints the board,

PrintBoard Functions

```
// prints the board
void Klotski :: printBoard(ostream& out)
{
    out << endl;
    for(int i = 0; i < sizeX; i++)
    {
        out << "   ";

        for(int j = 0; j < sizeY; j++)
        {
            out <<(char)Board[i][j]<<' ';
        }
        out << endl;
    }
    out << endl;
}
```

```
// prints the board
void EightPuzzle :: printBoard(ostream& out)
{
    out << endl;
    for(int i = 0; i < size; i++)
    {
        out << "   ";

        for(int j = 0; j < size; j++)
        {
            out <<(char)Board[i][j]<<' ';
        }
        out << endl;
    }
    out << endl;
}
```

```
// prints the board
void PegSolitaire :: printBoard(ostream& out)
{
    int rownum = 0;
    char columnletter = 'a';
    out << "    ";
    for(int k = 0; k < size; k++)
        {
           out << char(columnletter) << ' ';
            columnletter++;
        }
        out << endl;
    for(int i = 0; i < size; i++)
    {
        out << rownum + 1<< "   ";

        for(int j = 0; j < size; j++)
        {
            out <<(char)Board[i][j]<<' ';
        }
        out << endl;
        rownum++;
    }
    out << endl;
}
```

Initialize function creates the main board for selected games,

Initialize Functions

```
// initializes the board
void PegSolitaire :: initialize()
{
    Board =
    {
        {pegType :: free,pegType :: free,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: free,pegType :: free},
        {pegType :: free,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: free},
        {pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg},
        {pegType :: peg,pegType :: peg,pegType :: peg,pegType :: empty,pegType :: peg,pegType :: peg,pegType :: peg},
        {pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg},
        {pegType :: free,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: free},
        {pegType :: free,pegType :: free,pegType :: peg,pegType :: peg,pegType :: peg,pegType :: free,pegType :: free},
    };
}
```

```
// initializes the board
void EightPuzzle :: initialize()
{
    int x , y, xnew, ynew;
    char temp;
    Board = {{'1','2','3'}, {'4','5','6'}, {'7','8',' '}};

    for (int i = 0; i < size*size; ++i)
    {
        x = rand() % size;
        y = rand() % size;
        temp = Board[x][y];

        xnew = rand() % size;
        ynew = rand() % size;

        Board[x][y] = Board[xnew][ynew];

        Board[xnew][ynew] = temp;
    }
    possibile();
}
```

```
// initializes the board
void Klotski :: initialize()
{
    Board =
    {
        {'1','2','2','3'},
        {'1','2','2','3'},
        {'4','5','5','6'},
        {'4','7','8','6'},
        {'9',' ',' ','0'}
    };
}
```

ControlInput checks the inputs validity,

Control Input Functions

```
// Contols string input ,if it's valid or not
int PegSolitaire :: controlInput(string movement)
{
    int flag = 1;
    if(movement.size() != 4) flag = 0;
    if(toupper(movement[0]) < 'A' || toupper(movement[0]) > 'Z') flag = 0;
    if(movement[1] < '0' || movement[1] > '7') flag = 0;
    if(movement[2] != '-') flag = 0;
    if(!(toupper(movement[3]) == 'U' || toupper(movement[3]) == 'D'|| toupper(movement[3]) == 'R'|| toupper(movement[3]) == 'L')) flag = 0;

    return flag;
}
```

```
// Contols string input ,if it's valid or not
int EightPuzzle :: controlInput(string movement)
{
    int flag = 1;
    if(movement.size() != 3) flag = 0;
    if(toupper(movement[0]) < '1' || toupper(movement[0]) > '8') flag = 0;
    if(movement[1] != '-') flag = 0;
    if(!(toupper(movement[2]) == 'U' || toupper(movement[2]) == 'D'|| toupper(movement[2]) == 'R'|| toupper(movement[2]) == 'L')) flag = 0;
    return flag;
}
```

```
// Contols string input ,if it's valid or not
int Klotski :: controlInput(string movement)
{
    int flag = 1;
    if(movement.size() != 3) flag = 0;
    if(toupper(movement[0]) < '0' || toupper(movement[0]) > '9') flag = 0;
    if(movement[1] != '-') flag = 0;
    if(!(toupper(movement[2]) == 'U' || toupper(movement[2]) == 'D'|| toupper(movement[2]) == 'R'|| toupper(movement[2]) == 'L')) flag = 0;
    return flag;
}
```

AssignInput assigns the valid values to variables,

Assign Input Functions

```
// Divides the input and assigns the values to variables
void Klotski :: assignInput(string input)
{
    num = toupper(input[0]);
    Direction = toupper(input[2]);
    find();
}
```

```
// Divides the input and assigns the values to variables
void PegSolitaire :: assignInput(string input)
{
    row = input[1] - '0' - 1;
    column = toupper(input[0]) - 'A';
    Direction = toupper(input[3]);
}
```

```
// Divides the input and assigns the values to variables
void EightPuzzle :: assignInput(string input)
{
    num = toupper(input[0]);
    Direction = toupper(input[2]);
}
```

FLOW OF THE PROGRAM

When the program starts and objects are created constructors initializes the games with main boards that have been implemented and constant set of sizes.

If the move will be made automatically randomMove function creates a valid random movement values.

EndGame Function Checks if the game is ended, if game is not ended loop continues.

Check functions control if the move is possible.

If the move is possible move function executes relocation process.

This loop repeats until the games is ended.

If the move will be made manually playUser functions get input from the user.

Input is checked by controlInput function to see if it’s a valid move

If the move is valid it wil be assigned to class variable by assignInput function.

EndGame Function Checks if the game is ended, if game is not ended loop continues.

Check functions control if the move is possible.

If the move is possible move function executes relocation process.