Gebze Technical University
Department of Computer Engineering

CSE 241 – Object Oriented Programming
Fall 2021
HW #5

In this homework, we have a base class named `class BoardGame2D` and 3 other classes derived from `class BoardGame2D`

1- `class PegSolitaire : public BoardGame2D`
2- `class EightPuzzle : public BoardGame2D`
3- `class Klotski : public BoardGame2D`

`class BoardGame2D` is a class that can direct *derived classes* objects and interact with the user.

This class has interface as follows:

```
public:
    BoardGame2D(void);
    virtual void playUser(string mov) = 0;
    virtual void playUser() final;
    virtual void playAuto(void) = 0;
    virtual void playAutoAll(void) final;
    virtual void print(void) const = 0;
    friend ostream& operator<<(ostream &out, const BoardGame2D &B);
    virtual bool endGame(void) const = 0;
    virtual int boardScore(void) const = 0;
    virtual void initiliaze(void) = 0;
    static void playVector(vector<BoardGame2D*> games);
    virtual ~BoardGame2D();
protected:
    vector<vector<char>> board;
    string move;
```

There are some *pure virtual functions* which will be implement in *derived classes*. Since they are not needed implement in the base class they will be discussed in the below *derived classes.*

<u>They are;</u>

- `void playUser(string mov)`
  // Takes string and makes a single move.
- `void playAuto(void)`
  // Makes a random move by computer.
- `void print(void)`
  // Prints the board to the screen in proper.
- `bool endGame(void)`
  // Checks if the game over.
- `int boardScore(void)`
  // Return the score.
- `void initiliaze(void)`
  // Initiliazes the board.

`class BoardGame2D` Implementations

1- `BoardGame2D::BoardGame2D(void){}`
Default constructor. It does not do anything but derived class constructors going to do. Therefor it is needed and empty.

2- `void BoardGame2D::playUser(void)`
This function is where there is user interaction.
User gives decision if he/she make a single move with direction or make a random move by computer or finish the game automatically by random moves.

```
-------- Game Has Started --------


Instructors:

1- For Peg Solitare enter cell and direction. <2B UP>
2- For Eight Puzzle and Klotski games enter cell and direction. <2 UP>
3- Type 'AUTO' to play automatically one step.
4- Type 'FINISH' to play automatically until done.
5- Type 'EXIT' to terminate the game.


Press enter to continue.
```

According to the user selection, it is directed to the necessary functions.

```
while (!(endGame() || ins.compare("EXIT") == 0))
{
    cout << *this;
    ins.clear();
    getline(cin, ins);
    if (ins.compare("FINISH") == 0)
        playAutoAll();
    else if (ins.compare("AUTO") == 0)
        playAuto();
    else if (ins.compare("EXIT") != 0)
        playUser(ins);
}
```

And this continues until game is over. Finally prints the score to the screen.

```
Game Over!
Last score is: 9 (0 is the best score)
```

3- `void BoardGame2D::playAutoAll(void)`
This function plays game automatically until it is over. Displays boards with some pause gradually decreasing.
Some games may be over after many moves. For this reason, the user is asked whether to continue viewing or not.

```
 _____
| . | 7 | 4 |
 _____
| 8 | 6 | 1 |
 _____
| 5 | 2 | 3 |
 _____


Move 7 RIGHT has been performed.


Press 1 to skip viewing.
Press 2 to continue viewing without pause. (It may take a long time to over)
1
```

To give some pause we have *sleep_until* function in `#include <chrono>` library.
`sleep_until(system_clock::now() + milliseconds(<some number>));`

Finally it display number of random move then returns.

```
Played all moves automatically. (Total random move performed: 25949)
```

4- `ostream& operator<<(ostream &out, const BoardGame2D &B)`
Here the screen is cleared, the cursor top is moved to the left, the *print()* function is invoked.

```
cout << "\u001b[2J";
cout << "\u001b[1000A";
B.print();
return out;
```

5- `void BoardGame2D::playVector(vector<BoardGame2D*> games)`
This function takes a *vector* of `BoardGame2D pointer.` Plays the board one by one until it is over.

6- `BoardGame2D::~BoardGame2D(void){}`
Destructor of `BoardGame2D` class.

Lets take a look at the *derived classes* interfaces and implementations.

`class PegSolitaire : public BoardGame2D` is a class derived from `class BoardGame2D` which has implementations for 5th Peg Solitaire board.

This class has interface as follows:

```
public:
    PegSolitaire(void);
    virtual void playUser(string mov);
    using BoardGame2D::playUser;
    // Checks function playUser in base class either.
    virtual void playAuto(void);
    virtual void print(void) const;
    virtual bool endGame(void) const;
    virtual int boardScore(void) const;
    virtual void initiliaze(void);
    bool isEntryProper(string mov) const;
    bool isMoveValid(int i1, int i2, string dir) const;
    ~PegSolitaire();
```

`class PegSolitaire` has 6 virtual function inherited from base `class BoardGame2D` and also 2 member function.

`class PegSolitaire : public BoardGame2D` Implementations

1- `PegSolitaire::PegSolitaire(void)()`
Default constructor. It invokes base class constructor and `initiliaze();` inside of it.

2- `void PegSolitaire::playUser(string mov)`
`playUser()` function has a `string mov` parameter.

Firstly invokes `isEntryProper()` to check if the entry is proper.
Each game can have different move.
In Peg Solitaire game user should enter for example, *"7A LEFT"*.

```
if (!isEntryProper(mov))
{
    move = "invalid entry";
    return;
}
```

If entry is not proper, then *inherited* `string move` is equated to the `"invalid entry"`

If the entry is proper then it will invoke `isMoveValid()` function to check if move is valid.

```
if (!isMoveValid(i1, i2, dir))
{
    move = "invalid move";
    return;
}
```

If move is invalid, then *inherited* `string move` is equated to the `"invalid move"`

Finally if entry and move is valid, then it performs the move by a simple *switch case statement*.

```
    A B C D E F G H I

1         o o o
2         o o o
3         o o o
4     o o o o o o o o o
5     o o o o . o o o o
6     o o o o o o o o o
7         o o o
8         o o o
9         o o o


5G LEFT
```

For example if the direction is "*LEFT*"

```
case 'L':
    board[i1][i2] = '.';
    board[i1][i2 - 1] = '.';
    board[i1][i2 - 2] = 'o';
    break;
```

It changes the current and one left cells to the `'.'` which represents empty cell and 2 left cell `'o'` which represents peg.

3- `void PegSolitaire::playAuto(void)`

This function makes a random peg and direction selection by `rand()` until it make a valid selection.

```
do
{
    i1 = rand() % 9;
    i2 = rand() % 9;
    dirint = rand() % 4 + 1;
    if (dirint == 1) dir = "UP";
    else if (dirint == 2) dir = "DOWN";
    else if (dirint == 3) dir = "LEFT";
    else if (dirint == 4) dir = "RIGHT";
} while(!isMoveValid(i1, i2, dir));
```

If it makes a valid selection to the move, then it performs the move by changing peg from peg to the empty cell.

```
switch (dir[0])
{
case 'U':
    board[i1][i2] = '.';
    board[i1 - 1][i2] = '.';
    board[i1 - 2][i2] = 'o';
    break;
case 'D':
    board[i1][i2] = '.';
    board[i1 + 1][i2] = '.';
    board[i1 + 2][i2] = 'o';
    break;
case 'L':
    board[i1][i2] = '.';
    board[i1][i2 - 1] = '.';
    board[i1][i2 - 2] = 'o';
    break;
case 'R':
    board[i1][i2] = '.';
    board[i1][i2 + 1] = '.';
    board[i1][i2 + 2] = 'o';
    break;
}
```

4- `void PegSolitaire::print(void) const;`

This function simply prints the board according to the index vector has.

For empty cells it prints ‘.’ for pegs it prints ‘o’.

```
    A B C D E F G H I

1         o o o
2         o o o
3         o o o
4     o o o o o o o o o
5     o o o o . o o o o
6     o o o o o o o o o
7         o o o
8         o o o
9         o o o
```

5- `bool PegSolitaire::endGame(void) const`

Iterates the board in a loop, then invokes `isMoveValid()` function for every possibilities. If there is a possible move, then the game is not over.

```
for(int i = 0; i < board.size(); i++)
{
    for(int j = 0; j < board[0].size(); j++)
    {
        if(isMoveValid(i, j, "UP") || isMoveValid(i, j, "DOWN") ||
           isMoveValid(i, j, "LEFT") || isMoveValid(i, j, "RIGHT"))
           return false;
    }
}
```

6- `int PegSolitaire::boardScore(void) const`

In a loop, it counts number of pegs remains and returns the result.

```
int count = 0;

for(int i = 0; i < board.size(); i++)
{
    for(int j = 0; j < board[0].size(); j++)
    {
        if(board[i][j] == 'o')
            count++;
    }
}
return count;
```

7- `void PegSolitaire::initiliaze(void)`

Here I create a 2D char vector then assign it to the inherited 2D char vector as a 2D board.

```
vector<vector<char>> newBoard =
{
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '},
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '},
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', '.', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '},
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '},
    {' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' '}
};
board = newBoard;
```

8- `bool PegSolitaire::isMoveValid(int i1, int i2, string dir) const`

Checks if the direction for the move is proper.

If it is `returns true` other wise `returns false.`

For example if the direction is '*DOWN*'

```
case 'D':
      if (i1 <= 6 && board[i1][i2] == 'o' && board[i1 + 1][i2] == 'o' &&
board[i1 + 2][i2] == '.')
           return true;
     break;
```

9- `bool PegSolitaire::isEntryProper(string mov) const`
Check if the entry is proper, for example "*8A RIGHT*" then return true, other wise return false.

10- `PegSolitaire::~PegSolitaire(void){}`
Destructor for the `class PegSolitaire`

Here `class PegSolitaire` implementations done.

`class EightPuzzle : public BoardGame2D` is a class derived from `class BoardGame2D` which has implementations for Eight Puzzle game.

This class has interface as follows:

```
public:
    EightPuzzle(void);
    virtual void playUser(string mov);
    using BoardGame2D::playUser;
    // Checks playUser function in base class either.
    virtual void playAuto(void);
    virtual void print(void) const;
    virtual bool endGame(void) const;
    virtual int boardScore(void) const;
    virtual void initiliaze(void);
    void shuffle(vector<char> &shuf) const;
    int totalInversion(const vector<char> &vD1) const;
    bool isEntryProper(string mov) const;
    bool isMoveValid(int i1, int i2, string dir) const;
    ~EightPuzzle();
```

`class EightPuzzle` has 6 virtual function inherited from base `class BoardGame2D` and also 4 member function.

`class EightPuzzle : public BoardGame2D` Implementations

1. `EightPuzzle::EightPuzzle(void)`
   Default constructor of `class EightPuzzle`
   It invokes `initiliaze()` function to start an `EightPuzzle` object with a random board.

2. `void EightPuzzle::playUser(string mov)`
   `playUser()` function has a `string mov` parameter.

   Firstly invokes `isEntryProper()` to check if the entry is proper.
   Each game can have different move.
   In Klotski game user should enter for example, *"7 LEFT"*.

```
if (!isEntryProper(mov))
{
    move = "invalid entry";
    return;
}
```

If entry is not proper, then *inherited* `string move` is equated to the *"invalid entry"*

If the entry is proper then it will invoke `isMoveValid()` function to check if move is valid.

```
if (!isMoveValid(i1, i2, dir))
{
    move = "invalid move";
    return;
}
```

If move is invalid, then *inherited* `string move` is equated to the *"invalid move"*

Finally if entry and move is valid, then it performs the move by a simple *switch case statement*.

```
 ___________
| 2 | 8 | 4 |
|___|___|___|
| 3 | . | 6 |
|___|___|___|
| 1 | 5 | 7 |
|___|___|___|


8 DOWN
```

For example if the direction is "*DOWN*"

```
case 'D':
    board[i1 + 1][i2] = board[i1][i2];
    board[i1][i2] = '.';
    break;
```

It replace the peg with the '.' which represent the empty cell.

3. `void EightPuzzle::playAuto(void)`
This function makes a random peg and direction selection by `rand()` until it make a valid selection.

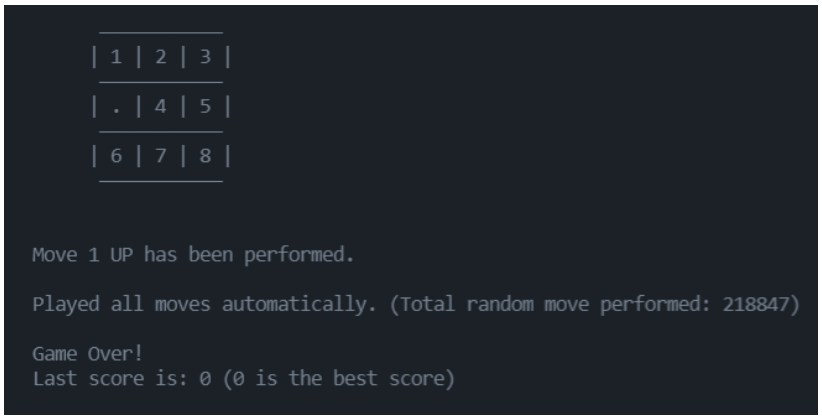
```
do
{
    i1 = rand() % 3;
    i2 = rand() % 3;
    dirint = rand() % 4 + 1;

    if (dirint == 1)
        dir = "UP";
    else if (dirint == 2)
        dir = "DOWN";
    else if (dirint == 3)
        dir = "LEFT";
    else if (dirint == 4)
        dir = "RIGHT";
} while(!isMoveValid(i1, i2, dir));
```

Here i1 and i2 is represent the index of `vector<vector<char>> board`.
After it makes a valid selection in the loop then it performs the move.

4. `void EightPuzzle::print(void) const`
   This function prints the board according to the which index `vector<vector<char>> board` has.

5. `bool EightPuzzle::endGame(void) const`
   `endGame()` function create a temporary 1D vector.
   Then insert all elements of the `board` into it then invoke the `totalInversion()` function with it.
   This function simply calculates the inversions, if the inversion is 0 this mean game is over.



6. `int EightPuzzle::boardScore(void) const`
   This function is simply returns the value that returns by `totalInversion()` function.

7. `void EightPuzzle::initiliaze(void)`
   Create a random board according to the value that returns by `totalInversion()` function.
   If that value is odd this mean the board is unsolvable, otherwise it is solvable.
   Firstly we define a 1D vector follow:

```
vector<char> shuf = {'1', '2', '3', '4', '5', '6', '7', '8', '.'};
```

   then shuffle it with `shuffle()` until a solvable case is done

```
do
 {
     shuffle(shuf);
 } while (!(totalInversion(shuf) % 2 == 0 && totalInversion(shuf) != 0));
```

   Then convert this shuffled 1D vector into the 2D vector.

```
for(int i = 0; i < shuf.size(); i++)
{
    temp.push_back(shuf[i]);
    if(i % 3 == 2)
    {
        board.push_back(temp);
        temp.clear();
    }
}
```

8. `void EightPuzzle::shuffle(vector<char> &shuf) const`
   Choos 2 index of vector randomly then swap the values in that index. Do that 1000 times.

```
int i1, i2;
char temp;
for (int i = 0; i < 1000; i++)
{
    i1 = rand() % 9;
    i2 = rand() % 9;
    temp = shuf[i1];
    shuf[i1] = shuf[i2];
    shuf[i2] = temp;
}
```

9. `int EightPuzzle::totalInversion(const vector<char> &vD1) const`
   Check the right side of the all elements then calculate total inversions.
   Which mean count number of smaller number from the selected number on the right side of the array.

   For example `temp` is a `vector` of `char`:
   `vector<char> vector<char> temp = {'5', '3', '1', '2', '6', '8', '7', '4'};`

   5 has 4 number smaller then itself on the right side of vector from itself.
   3 has 2 number smaller then itself on the right side of vector from itself.
   1 has 0 number smaller then itself on the right side of vector from itself.
   2 has 0 number smaller then itself on the right side of vector from itself.
   6 has 1 number smaller then itself on the right side of vector from itself.
   8 has 2 number smaller then itself on the right side of vector from itself.
   7 has 1 number smaller then itself on the right side of vector from itself.
   4 has 0 number smaller then itself on the right side of vector from itself.

   Then sum of (4 + 2 + 0 + 0 + 1 + 2 + 1) is 10.
   Therefor total inversion is 10.

10. `bool EightPuzzle::isMoveValid(int i1, int i2, string dir) const`
    Checks if the direction for the move is proper.
    If it is return `true` other wise returns `false.`
    For example if the direction is '*DOWN*'

```
case 'D':
    if(i1 != 2 && board[i1 + 1][i2] == '.')
        return true;
    break;
```

11. `bool EightPuzzle::isEntryProper(string mov) const`
    Check if the entry is proper, for example "*8 RIGHT*" then return true, other wise return false.

12. `EightPuzzle::~EightPuzzle(void) {}`
    Destructor for the `class EightPuzzle`

Here `class EightPuzzle` implementations done.

`class Klotski: public BoardGame2D` is a class derived from `class BoardGame2D` which has implementations for Eight Puzzle game.

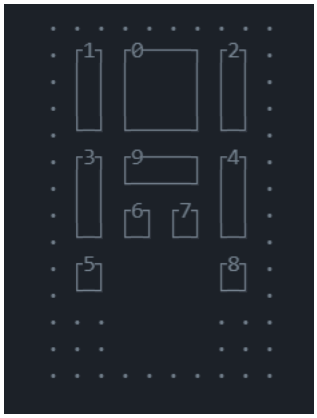
This class has interface as follows:

```
    public:
        Klotski(void);
        virtual void playUser(string mov);
        using BoardGame2D::playUser;
        // Checks function playUser in base class either.
        virtual void playAuto(void);
        virtual void print(void) const;
        virtual bool endGame(void) const;
        virtual int boardScore(void) const;
        virtual void initiliaze(void);
        bool isEntryProper(string mov) const;
        bool isMoveValid(char peg, string dir) const;
        ~Klotski();
```

`class Klotski` has 6 virtual function inherited from base `class BoardGame2D` and also 2 member function.

`class Klotski : public BoardGame2D` Implementations

1. `enum PieceType {v1 = '1', v2 = '2', v3 = '3', v4 = '4', s1 = '5', s2 = '6', s3 = '7', s4 = '8', h1 = '9', Re = '0', Sp = ' ', Wa = '.'};`

   In Klotski game there are a board and 4 vertical (4 x 2) 1 horizontal (2 x 4) rectangle, 4 small (2 x 2) and 1 large (4 x 4) square.
   To represent all there is an enum `PieceType`.

2. `Klotski::EightPuzzle(void)`
   Default constructor of `class Klotski`
   It invokes `initiliaze()` function to start an `Klotski` object with the constant board.

3. `void Klotski::playUser(string mov)`
   `playUser()` function has a `string mov` parameter.

   Firstly invokes `isEntryProper()` to check if the entry is proper.
   Each game can have different move.

In Klotski game user should enter for example, *“7 LEFT”.*

```
if (!isEntryProper(mov))
{
    move = "invalid entry";
    return;
}
```

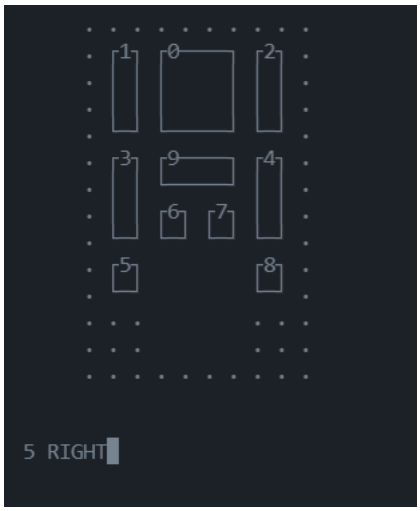
If entry is not proper, then *inherited* `string move` is equated to the *"invalid entry"*

If the entry is proper then it will invoke `isMoveValid()` function to check if move is valid.

```
if (!isMoveValid(i1, i2, dir))
{
    move = "invalid move";
    return;
}
```

If move is invalid, then *inherited* `string move` is equated to the *"invalid move"*

Finally if entry and move is valid, then it performs the move by *switch case statement*.



Firstly finds left top and right buttom corner of selected peg.

```
int luc1, luc2, rdc1, rdc2;
bool isfound = false;

for (int i = 0; i < board.size(); i++)
{
    for (int j = 0; j < board[0].size(); j++)
    {
        if (isfound == false && board[i][j] == peg)
        {
            luc1 = i; luc2 = j; isfound = true;
        }
        else if (isfound == true && board[i][j] == peg)
        {
            rdc1 = i; rdc2 = j;
        }
    }
}
```

Then, carry the selected peg. For example selection is “*RIGHT*”

```
case 'R':
    for (int i = 0; i < board.size(); i++)
    {
        for (int j = 0; j < board[0].size(); j++)
        {
            if (i >= luc1&& i <= rdc1)
                if (j >= luc2 + 2 && j <= rdc2 + 2)
                    board[i][j] = peg;
        }
    }
    break;
```

4. `void Klotski::playAuto(void)`
   This function makes a random peg and direction selection by `rand()` until it make a valid selection.

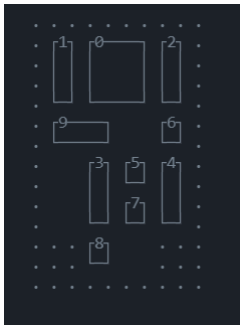
```
do
{
    peg = rand() % 10 + '0';
    dirint = rand() % 4 + 1;

    if (dirint == 1)
        dir = "UP";
    else if (dirint == 2)
        dir = "DOWN";
    else if (dirint == 3)
        dir = "LEFT";
    else if (dirint == 4)
        dir = "RIGHT";
} while(!isMoveValid(peg, dir));
```

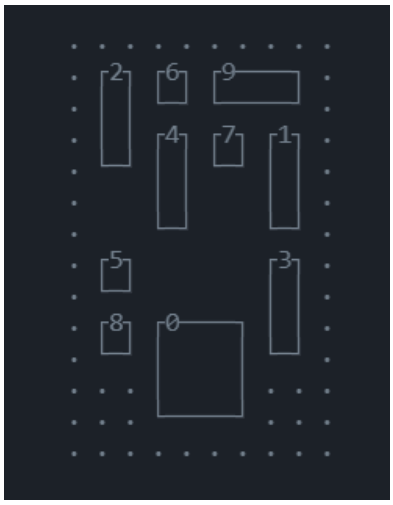
Then automatically plays the move.
For example random direction is “*LEFT*”

```
case 'L':
    for (int i = 0; i < board.size(); i++)
    {
        for (int j = 0; j < board[0].size(); j++)
        {
            if (i >= luc1&& i <= rdc1)
                if (j >= luc2 - 2 && j <= rdc2 - 2)
                    board[i][j] = peg;
        }
    }
    break;
```

5. `void Klotski::print(void) const`
   This function prints the board according to the which index `vector<vector<char>> board` has.
   Here it use corner characters and long dash! Be careful to have a appropriate terminal.



6. `bool EightPuzzle::endGame(void) const`
   If the top left of big square is on the place which makes game over then return *true*. Otherwise return *false*.



7. `int Klotski::boardScore(void) const`
   It returns a value according to the place of the 4 x 4 peg. If game is over then it returns 0 otherise it returns vertical location of the peg.

```
int i1, i2;
for (int i = 0; i < board.size(); i++)
    for (int j = 0; j < board[0].size(); j++)
        if(Re == board[i][j])
            {i1 = i; i2 = j;}
return ((-1) * (i1 - 12));
```

8. `void Klotski::initiliaze(void)`
   Creates a 2D temporary char vector and fill this vector with enum *PieceType*. Then, assign it protected `vector<vector<char>> board.`

```
vector<vector<char>> newBoard =
{
    {Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa},
    {Wa, v1, v1, Re, Re, Re, Re, v2, v2, Wa},
    {Wa, v1, v1, Re, Re, Re, Re, v2, v2, Wa},
    {Wa, v1, v1, Re, Re, Re, Re, v2, v2, Wa},
    {Wa, v1, v1, Re, Re, Re, Re, v2, v2, Wa},
    {Wa, v3, v3, h1, h1, h1, h1, v4, v4, Wa},
    {Wa, v3, v3, h1, h1, h1, h1, v4, v4, Wa},
    {Wa, v3, v3, s2, s2, s3, s3, v4, v4, Wa},
    {Wa, v3, v3, s2, s2, s3, s3, v4, v4, Wa},
    {Wa, s1, s1, Sp, Sp, Sp, Sp, s4, s4, Wa},
    {Wa, s1, s1, Sp, Sp, Sp, Sp, s4, s4, Wa},
    {Wa, Wa, Wa, Sp, Sp, Sp, Sp, Wa, Wa, Wa},
    {Wa, Wa, Wa, Sp, Sp, Sp, Sp, Wa, Wa, Wa},
    {Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa, Wa}
};
board = newBoard;
```

9. `bool Klotski::isMoveValid(char peg, string dir) const`
   Checks if the direction for the move is proper.
   If it is return `true` other wise returns `false.`

10. `bool Klotski::isEntryProper(string mov) const`
    Check if the entry is proper, for example "*8 RIGHT*" then return `true`, other wise return `false`.

11. `EightPuzzle::~EightPuzzle(void) {}`
    Destructor for the `class Klotski`