```cpp
#ifndef LIBSPY_H_INCLUDED
#define LIBSPY_H_INCLUDED

#include <string>
using namespace std;

namespace SpySpace{
    typedef int* t_OneDArray;
    typedef int** t_TwoDArray;

    enum enFeatures
    {
        EMPTY,
        WALL,
        ENEMY_UP,
        ENEMY_DOWN
    };

    enum enErrors
    {
        SUCESS,
        ERR_ARGC = -1,
        ERR_CONV = -2,
        ERR_RANGE = -3
    };

    enum enState
    {
        RUNNING,
        WON,
        LOST,
        QUIT
    };

    struct stcGame{
        // Constants
        const char FEATURES[4] = {'.','|','^','!'};
        const char F_PLAYER = 'P';
        // Member variables
        t_TwoDArray arrGame;
        int intRows;
        int intCols;
        int intPRow;
        int intPCol;
        enState state;
        bool blnSpotted;
        bool blnSameCol;
        int intMove;

        // Function pointers
        void (*PrintWorld)(stcGame* objThis);
        void (*MovePlayer)(stcGame* objThis, char chInput);
        void (*MoveEnemies)(stcGame* objThis);
    };

    // "Constructor" function.
    stcGame* CreateGame(int intRows, int intCols);
    // "Destructor" function.
    void DestroyGame(stcGame*& objGame);
    // "Utility" functions
    void DestroyArray(t_TwoDArray, int intRows);
    t_TwoDArray AllocMem(int intRows, int intCols);
    int GetInt(string strNum);
    void Pause();
}

#endif // LIBSPY_H_INCLUDED
```

```cpp
#include "LibSpy.h"

#include <iostream>
#include <sstream>

using namespace std;

namespace SpySpace{
    /*
     * Concrete functions with implementations
     * for the function pointers to point to.
     */
    void PrintWorldImplementation(stcGame* objThis)
    {
        system("cls");
        for(int r=0;r<objThis->intRows;r++)
        {
            for(int c=0;c<objThis->intCols;c++)
            {
                if(r==objThis->intPRow && c==objThis->intPCol)
                    cout << objThis->F_PLAYER;
                else
                    cout <<objThis->FEATURES[objThis->arrGame[r][c]];
                cout << " ";
            }
            cout << endl;
        }
        cout << "w: Move Up" << endl
             << "s: Move Down" << endl
             << "a: Move Left" << endl
             << "d: Move Right" << endl
             << "q: Quit" << endl;
    }

    void CopyArray(t_TwoDArray arrFrom, t_TwoDArray arrTo, int intRows, int intCols, int intExcept)
    {
        for(int r=0;r<intRows;r++)
        {
            for(int c=0;c<intCols;c++)
            {
                arrTo[r][c] = EMPTY;
                if(arrFrom[r][c]<intExcept)
                    arrTo[r][c] = arrFrom[r][c];
            }
        }
    }

    bool IsInWorld(int intRows, int intCols, int intRow, int intCol)
    {
        return (intRow>=0&&intRow<intRows &&
                intCol>=0&&intCol<intCols);
    }


    void MoveEnemy(stcGame* objGame, t_TwoDArray arrTemp, int intRow, int intCol, int intFeature)
    {
        int intDRow = intRow;
        int intDCol = intCol;
        if(intFeature==ENEMY_UP)
            intDRow--;
        else
            intDRow++;
        if(IsInWorld(objGame->intRows,objGame->intCols,intDRow,intDCol))
        {
            arrTemp[intDRow][intDCol]=intFeature;
            arrTemp[intRow][intCol]=EMPTY;
        }
        else
        {
            if(intFeature==ENEMY_UP)
                intFeature=ENEMY_DOWN;
            else
                intFeature=ENEMY_UP;
            arrTemp[intRow][intCol]=intFeature;
        }

        //See if the enemy is in the same column as the player
```

```
78          if(intCol==objGame->intPCol)
79          {
80              objGame->blnSameCol = true;
81          }
82      }
83
84      void DestroyArray(t_TwoDArray arrGame, int intRows)
85      {
86          for(int r=0;r<intRows;r++)
87              delete[] arrGame[r];
88          delete[] arrGame;
89          arrGame = nullptr;
90      }
91
92      void MoveEnemiesImplementation(stcGame* objThis)
93      {
94          //Assume the enemy and the player is not in the same col
95          objThis->blnSameCol = false;
96          objThis->blnSpotted = false;
97
98          //Move each of the enemies
99          t_TwoDArray arrTemp = AllocMem(objThis->intRows,objThis->intCols);
100         CopyArray(objThis->arrGame,arrTemp,objThis->intRows,objThis->intCols,ENEMY_UP);
101         for(int r=0;r<objThis->intRows;r++)
102         {
103             for(int c=1;c<objThis->intCols-1;c+=2)
104             {
105                 if(objThis->arrGame[r][c]>=ENEMY_UP)
106                     MoveEnemy(objThis, arrTemp, r, c, objThis->arrGame[r][c]);
107             }
108         }
109         CopyArray(arrTemp,objThis->arrGame,objThis->intRows,objThis->intCols,999);
110         DestroyArray(arrTemp,objThis->intRows);
111
112         //Modify the player movement if the enemy and the player is in the same column.
113         if(objThis->blnSameCol)
114         {
115             //Find the row the enemy is in
116             int intERow = 0;
117             for(int r=0;r<objThis->intRows;r++)
118             {
119                 if(objThis->arrGame[r][objThis->intPCol]==enFeatures::ENEMY_DOWN ||
120                     objThis->arrGame[r][objThis->intPCol] == enFeatures::ENEMY_UP)
121                     intERow = r;
122             }
123
124             int intFeature = objThis->arrGame[intERow][objThis->intPCol];
125             if(intFeature==ENEMY_UP)
126             {
127                 if(objThis->intPRow<intERow)
128                 {
129                     objThis->blnSpotted=true;
130                     return;
131                 }
132                 objThis->blnSpotted = false;
133                 return;
134             }
135
136             if(intFeature==ENEMY_DOWN)
137             {
138                 if(objThis->intPRow>intERow)
139                 {
140                     objThis->blnSpotted=true;
141                     return;
142                 }
143                 objThis->blnSpotted = false;
144                 return;
145             }
146         }
147     }
148
149     void MovePlayerImplementation(stcGame* objThis, char chInput)
150     {
151         //Reset the move counter if spotted and finished moving
152         if(objThis->blnSpotted && objThis->intMove==0)
153             objThis->intMove = 2;
154
155         //If not spotted, ensure the movement is normal
156         if(!objThis->blnSpotted)
```

```
157                 objThis->intMove = 1;
158
159             if(--objThis->intMove==0)
160             {
161                 int intDRow = objThis->intPRow;
162                 int intDCol = objThis->intPCol;
163                 switch(chInput)
164                 {
165                 case 'w':
166                     intDRow--;
167                     break;
168                 case 's':
169                     intDRow++;
170                     break;
171                 case 'a':
172                     intDCol--;
173                     break;
174                 case 'd':
175                     intDCol++;
176                     break;
177                 }
178                 if(IsInWorld(objThis->intRows,objThis->intCols,intDRow,intDCol))
179                 {
180                     //See if the destination contains an enemy
181                     if(objThis->arrGame[intDRow][intDCol]>=ENEMY_UP)
182                     {
183                         objThis->state = LOST;
184                         return;
185                     }
186
187                     //See if the existing location contains an enemy
188                     if(objThis->arrGame[objThis->intPRow][objThis->intPCol]>=ENEMY_UP)
189                     {
190                         objThis->state = LOST;
191                         return;
192                     }
193
194                     if(objThis->arrGame[intDRow][intDCol]!=WALL)
195                     {
196                         objThis->intPRow = intDRow;
197                         objThis->intPCol = intDCol;
198                     }
199
200                     //See if we reached the safe space
201                     if(objThis->intPCol==0)
202                         objThis->state = enState::WON;
203                 }
204             }
205         }
206
207         t_TwoDArray AllocMem(int intRows, int intCols)
208         {
209             t_TwoDArray arrGame = new t_OneDArray[intRows];
210             for(int r=0;r<intRows;r++)
211             {
212                 arrGame[r] = new int[intCols];
213                 for(int c=0;c<intCols;c++)
214                 {
215                     arrGame[r][c] = EMPTY;
216                 }
217             }
218             return arrGame;
219         }
220
221         int GetRand(int intLower, int intUpper)
222         {
223             int intRange = intUpper-intLower+1;
224             return rand()%intRange+intLower;
225         }
226
227         void PlaceEnemies(stcGame* objGame)
228         {
229             for(int c=1;c<objGame->intCols-1;c++)
230             {
231                 if(c%2==1)
232                 {
233                     int intRow = GetRand(0,objGame->intRows-1);
234                     objGame->arrGame[intRow][c] = GetRand(2,3);
235                 }
```

```
236             }
237         }
238 
239         stcGame* CreateGame(int intRows, int intCols)
240         {
241             t_TwoDArray arrGame = AllocMem(intRows,intCols);
242             for(int r=0;r<intRows;r++)
243             {
244                 for(int c=1;c<intCols-1;c++)
245                 {
246                     if(c%2==0)
247                         arrGame[r][c] = WALL;
248                 }
249             }
250             stcGame* objGame = new stcGame;
251             objGame->arrGame = arrGame;
252 
253             //Place the player
254             objGame->intPCol = intCols-1;
255             objGame->intPRow = GetRand(0,intRows-1);
256             objGame->intRows = intRows;
257             objGame->intCols = intCols;
258             objGame->state = RUNNING;
259             objGame->blnSpotted = false;
260             objGame->blnSameCol = false;
261             objGame->intMove = 1;
262 
263             //Place the enemies
264             PlaceEnemies(objGame);
265             //PlaceFeature(game,intEnemies,ENEMY,1,intCols-2);
266 
267             //Open the doors
268             for(int c=1;c<intCols-1;c++)
269             {
270                 if(c%2==0)
271                 {
272                     int intRow = GetRand(0,intRows-1);
273                     objGame->arrGame[intRow][c] = EMPTY;
274                 }
275             }
276 
277             // Initialise the function pointers.
278             objGame->MoveEnemies = &MoveEnemiesImplementation;
279             objGame->MovePlayer = &MovePlayerImplementation;
280             objGame->PrintWorld = &PrintWorldImplementation;
281 
282             return objGame;
283         }
284 
285         void DestroyGame(stcGame*& objGame)
286         {
287             DestroyArray(objGame->arrGame, objGame->intRows);
288             delete objGame;
289             objGame = nullptr;
290         }
291 
292         int GetInt(string strNum)
293         {
294             stringstream ss {strNum};
295             int intNum;
296             ss >> intNum;
297             if(ss.fail())
298             {
299                 cerr << "Cannot convert string to int" << endl;
300                 exit(ERR_CONV);
301             }
302             return intNum;
303         }
304 
305         void Pause()
306         {
307             cin.ignore(100,'\n');
308             cout << "Press Enter to continue" << endl;
309             cin.get();
310         }
311     }
312 
```

```cpp
#include "libSpy.h"

#include <ctime>
#include <iostream>

using namespace SpySpace;

int main(int argc, char** argv)
{
    srand(time(nullptr));
    bool blnContinue = true;
    char chInput = '\0';

    if(argc!=3)
    {
        cerr << "Incorrect num of command line args" << endl;
        exit(ERR_ARGC);
    }

    int intRows = GetInt(argv[1]);
    int intCols = GetInt(argv[2]);

    stcGame* objGame = CreateGame(intRows,intCols);
    cout << "createGame" << endl;
    do
    {
        objGame->PrintWorld(objGame);
        cout << "createGame" << endl;
        cin >> chInput;
        chInput = tolower(chInput);
        cout << "createGame" << endl;
        switch(chInput)
        {
        case 'w':
        case 's':
        case 'a':
        case 'd':
            objGame->MovePlayer(objGame,chInput);
            break;
        case 'q':
            objGame->state = QUIT;
            break;
        default:
            cerr << "Please select a valid input" << endl;
            Pause();
        }
        objGame->MoveEnemies(objGame);
        if(objGame->state!=RUNNING)
            blnContinue = false;
    }while(blnContinue);

    return SUCESS;
}

```