```cpp
#include "libCrime.h"
#include <iostream>

using namespace std;
using namespace CrimeSpace;

int main(int argc, char** argv)
{
    //Seeding the random number generator
    srand(time(nullptr));

    //Double check number of command line args.
    if(argc!=5)
    {
        cerr << "Incorrect number of command line arguments" << endl;
        exit(ERR_ARGC);
    }

    //Allocate command line args to variables.
    int intRows = GetInt(argv[1]);
    int intCols = GetInt(argv[2]);
    int intTurns = GetInt(argv[3]);
    int intCountClues = GetInt(argv[4]);

    //Basic range checking
    RangeCheck(intRows,DIM_MIN,DIM_MAX);
    RangeCheck(intCols,DIM_MIN,DIM_MAX);
    RangeCheck(intTurns,MIN_TURNS,MAX_TURNS);
    RangeCheck(intCountClues,MIN_CLUE,(intRows*intCols)/3);     //Make sure the maximum clues isn't more than the third amount of space.

    //Initialise the game variable.
    tGame stcGame = InitGame(intRows,intCols,intCountClues,intTurns);
    char chInput = '\0';
    bool blnContinue = true;

    //Main loop
    do
    {
        //Output the world
        PrintWorld(stcGame);
        //Get input
        cin >> chInput;
        chInput = tolower(chInput);
        //Handle input
        switch(chInput)
        {
        case 'w':
            MovePlayer(stcGame,MOVE_UP);
            break;
        case 's':
            MovePlayer(stcGame,MOVE_DOWN);
            break;
        case 'a':
            MovePlayer(stcGame,MOVE_LEFT);
            break;
        case 'd':
            MovePlayer(stcGame,MOVE_RIGHT);
            break;
        case 'p':
            Investigate(stcGame);
            break;
        case 'q':
            stcGame.enStatus = QUIT;
            break;
        default:
            cerr << "Incorrect option.  Please retry" << endl;
            Pause();
        }
        //Update some basic game information every turn
        Update(stcGame);
        //See if the game needs to stop
        if(stcGame.enStatus!=RUNNING)
            blnContinue = false;
    }while(blnContinue);

    cout << "************************************************************" << endl;
    if(stcGame.enStatus == QUIT)
        cout << "*                    YOU QUIT THE GAME                     *" << endl;
    if(stcGame.enStatus == LOST)
        cout << "*     YOU DID NOT MANAGE TO COLLECT ALL THE CLUES           *" << endl;
    if(stcGame.enStatus == WON)
        cout << "*     CONGRATULATIONS. YOU COLLECTED ALL THE CLUES          *" << endl;
    cout << "************************************************************" << endl;
```

```
84
85          Dealloc(stcGame.arrGame, intRows);
86
87          return SUCCESS;
88      }
89
```

```cpp
#ifndef LIBCRIME_H_INCLUDED
#define LIBCRIME_H_INCLUDED

#define NDEBUG //This is added to disable all assert instructuion

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <sstream>
#include <cctype>
#include <cassert>

using namespace std;

namespace CrimeSpace
{
    //Enumeration that describes error messages
    enum tErrors
    {
        SUCCESS,
        ERR_ARGC,
        ERR_CONV,
        ERR_RANGE,
        ERR_SPACE
    };
    //Enumeration that declares the different type of features.
    enum tFeatures
    {
        EMPTY,
        PLAYER,
        CLUE_REAL,
        CLUE_HIDDEN,    //Hidden real clue, will be transformed into CLUE_REAL
        CLUE_POTENTIAL
    };

    //Describes the movement direction.  Abstracts away the keys from the movement function.
    enum tMovement
    {
        MOVE_LEFT,
        MOVE_RIGHT,
        MOVE_UP,
        MOVE_DOWN
    };

    //Enumeration declaring the different states of the game.
    enum tStatus
    {
        RUNNING,
        QUIT,
        WON,
        LOST
    };

    //One D array of features, with corresponding features in tFeatures.
    const char FEATURES[5] = {'.','P','!','X','X'};

    //Basic range variable information
    const int DIM_MIN = 5;
    const int DIM_MAX = 30;
    const int MIN_TURNS = 5;
    const int MAX_TURNS = 100;
    const int MIN_CLUE = 1;

    //Aliases for one and two D arrays.
    typedef int* t1DArray;
    typedef int** t2DArray;

    //Struct that defines the information in the game
    struct tGame
    {
        t2DArray arrGame;       //The Two-D array that will store the features.
        int intRows;            //Total Rows
        int intCols;            //Total Cols
        int intPRow;            //Player Row
        int intPCol;            //Player Col
        int intTotalClues;      //Total Clues in the game world
        int intCluesFound;      //Number of clues found
        int intTurns;           //Number of turns left
        tStatus enStatus;       //Status of the game
    };

    tGame InitGame(int intRows, int intCols, int intClues, int intTurns);
    int GetInt(string strNum);
    void PrintWorld(tGame stcGame);
```

```
85        void MovePlayer(tGame& stcGame, int intMovement);
86        void Pause();
87        void Investigate(tGame& stcGame);
88        void Update(tGame& stcGame);
89        void Dealloc(t2DArray& arrGame, int intRows);
90        void RangeCheck(int intVal, int intMin, int intMax);
91    }
92
93    #endif // LIBCRIME_H_INCLUDED
94
```

```cpp
#include "libCrime.h"

namespace CrimeSpace
{
    //Returns a random int number between intLow and intHigh
    int GetRand(int intLow, int intHigh)
    {
        assert(intHigh>intLow);
        int intRange = intHigh - intLow + 1;
        return rand()%intRange + intLow;
    }

    //Converts strNum to int.  Exits if fail
    int GetInt(string strNum)
    {
        stringstream ss {strNum};
        int intNum;
        ss >> intNum;
        if(ss.fail())
        {
            cerr << "Could not convert string to int" << endl;
            exit(ERR_CONV);
        }
        return intNum;
    }

    //Checks if intVal is between intMin and intMax
    void RangeCheck(int intVal, int intMin, int intMax)
    {
        if(intVal<intMin || intVal > intMax)
        {
            cerr << intVal << " should be between " << intMin << " and " << intMax << endl;
            exit(ERR_RANGE);
        }

    }

    //Returns a new 2D array, initialised with empty spaces
    t2DArray AllocMem(int intRows, int intCols)
    {
        t2DArray arrGame;
        arrGame = new t1DArray[intRows];
        for(int r=0;r<intRows;r++)
        {
            arrGame[r] = new int[intCols];
            for(int c=0;c<intCols;c++)
            {
                arrGame[r][c] = EMPTY;
            }
        }

        return arrGame;
    }

    //Place intCount number of intFeatures in the game.  Updates the intRow and intCol with the last placed location
    void PlaceFeature(tGame& stcGame, int intCount, int intFeature, int& intRow, int& intCol)
    {
        for(int n=0;n<intCount;n++)
        {
            intRow = GetRand(0,stcGame.intRows-1);
            intCol = GetRand(0,stcGame.intCols-1);
            while(stcGame.arrGame[intRow][intCol]!=EMPTY)
            {
                intRow = GetRand(0,stcGame.intRows-1);
                intCol = GetRand(0,stcGame.intCols-1);
            }
            stcGame.arrGame[intRow][intCol] = intFeature;
        }
    }

    //Creates a new tGame struct with initial game values.
    tGame InitGame(int intRows, int intCols, int intClues, int intTurns)
    {
        //Struct for the game
        tGame stcGame;
        //Allocates memory for the arrGame member
        stcGame.arrGame = AllocMem(intRows,intCols);
        //Sets the varias initial values.
        stcGame.intRows = intRows;
        stcGame.intCols = intCols;
        stcGame.intTotalClues = intClues;
        stcGame.intCluesFound = 0;
        stcGame.intTurns = intTurns;
```

```
84          stcGame.enStatus = RUNNING;
85
86          //Ensures that there is enough space for the total number of real and potential clues
87          int intTotalSpace = intRows * intCols;
88          int intTotalFeatures = intClues + (intClues * 2);
89          if(intTotalFeatures>=intTotalSpace)
90          {
91              cerr << "There is not enough space in the game for all the features" << endl;
92              exit(ERR_SPACE);
93          }
94
95          //Places the different features in the 2D array
96          int intRow = 0;
97          int intCol = 0;
98
99          //Place the clues
100         PlaceFeature(stcGame,intClues,CLUE_HIDDEN,intRow, intCol);
101
102         //Place the potential clues
103         PlaceFeature(stcGame,intClues*2,CLUE_POTENTIAL,intRow, intCol);
104
105         //Place the player
106         PlaceFeature(stcGame,1,PLAYER,intRow,intCol);
107         stcGame.intPRow = intRow;
108         stcGame.intPCol = intCol;
109
110         return stcGame;
111     }
112
113     //Outputs the world
114     void PrintWorld(tGame stcGame)
115     {
116         assert(stcGame.arrGame!=nullptr);
117         system("cls");
118         for(int r=0;r<stcGame.intRows;r++)
119         {
120             for(int c=0;c<stcGame.intCols;c++)
121             {
122                 cout << FEATURES[stcGame.arrGame[r][c]] << " ";
123             }
124             cout << endl;
125         }
126         cout << "w) Move Up" << endl
127              << "s) Move Down" << endl
128              << "a) Move Left" << endl
129              << "d) Move Right" << endl
130              << "p) Investigate" << endl
131              << "q) Quit" << endl
132              << "Clues collected: " << stcGame.intCluesFound << endl
133              << "Turns left:" << stcGame.intTurns << endl;
134     }
135
136     //Will pause the game
137     void Pause()
138     {
139         cin.ignore(100,'\n');
140         cout << "Press Enter to continue" << endl;
141         cin.get();
142     }
143
144     //Returns true if intRow and intCol is within the boundaries of the intRows and intCols
145     bool IsInWorld(int intRows, int intCols, int intRow, int intCol)
146     {
147         return (intRow>=0 && intRow < intRows &&
148                 intCol>=0 && intCol < intCols);
149
150     }
151
152     //Moves the player
153     void MovePlayer(tGame& stcGame, int intMovement)
154     {
155         assert(stcGame.arrGame!=nullptr);
156         //Get the current row and col of the player and set to potential destination
157         int intDRow = stcGame.intPRow;
158         int intDCol = stcGame.intPCol;
159         //Modify potential destination, given the movement
160         switch(intMovement)
161         {
162         case MOVE_UP:
163             intDRow--;
164             break;
165         case MOVE_DOWN:
166             intDRow++;
167             break;
```

```
168             case MOVE_LEFT:
169                 intDCol--;
170                 break;
171             case MOVE_RIGHT:
172                 intDCol++;
173                 break;
174             }
175 
176             //Confirms that the destination location is in the world
177             if(IsInWorld(stcGame.intRows,stcGame.intCols,intDRow,intDCol))
178             {
179                 //Move only if the destination is empty
180                 if(stcGame.arrGame[intDRow][intDCol]==EMPTY)
181                 {
182                     stcGame.arrGame[stcGame.intPRow][stcGame.intPCol]=EMPTY;
183                     stcGame.arrGame[intDRow][intDCol] = PLAYER;
184                     stcGame.intPRow = intDRow;
185                     stcGame.intPCol = intDCol;
186                 }
187             }
188         }
189 
190         //Search for clues around the player
191         void Investigate(tGame& stcGame)
192         {
193             assert(stcGame.arrGame!=nullptr);
194             //Loop through each area around the player
195             for(int r=stcGame.intPRow-1;r<=stcGame.intPRow+1;r++)
196             {
197                 for(int c=stcGame.intPCol-1;c<=stcGame.intPCol+1;c++)
198                 {
199                     //Make sure the area is in the array before trying to access
200                     if(IsInWorld(stcGame.intRows,stcGame.intCols,r,c))
201                     {
202                         //Change real hidden clues into real clues.  Update the found clues.
203                         if(stcGame.arrGame[r][c]==CLUE_HIDDEN)
204                         {
205                             stcGame.arrGame[r][c]=CLUE_REAL;
206                             stcGame.intCluesFound++;
207                         }
208                         //Remove potential clues from the array
209                         if(stcGame.arrGame[r][c]==CLUE_POTENTIAL)
210                             stcGame.arrGame[r][c]=EMPTY;
211                     }
212                 }
213             }
214         }
215 
216         //Decrement the number of turns.
217         //Test if all the clues were found.
218         void Update(tGame& stcGame)
219         {
220             stcGame.intTurns--;
221             if(stcGame.intTurns<0)
222                 stcGame.enStatus = LOST;
223             if(stcGame.intCluesFound==stcGame.intTotalClues)
224                 stcGame.enStatus = WON;
225         }
226 
227         //Deallocate the memory associated with a 2D array.
228         void Dealloc(t2DArray& arrGame, int intRows)
229         {
230             assert(arrGame!=nullptr);
231             for(int r=0;r<intRows;r++)
232                 delete[] arrGame[r];
233             delete[] arrGame;
234             arrGame = nullptr;
235         }
236     }
237 
```