```cpp
#include "libFuncs.h"

using namespace std;
using namespace ZorkSpace;

int main(int argc, char** argv)
{
    //Seed the random number generator
    srand(time(nullptr));
    //Test command line args
    if(argc != 4)
    {
        cerr << "Incorrect number of command line arguments." << endl
             << "Please run " << argv[0] << " <numOfRows> <numOfCols> <numOfBatteries>" << endl;
        exit(ERROR_NUM_ARGS);
    }

    //Convert command line args.
    int intTotalRows = ToInt(argv[1]);
    int intTotalCols = ToInt(argv[2]);
    int intTotalBatteries = ToInt(argv[3]);



    //Set initial variables
    bool blnContinue = true;
    bool blnTorchOn = false;
    bool blnSurvived = false;
    bool blnCaught = false;
    bool blnPitFall = false;
    bool blnEnd = false;
    int intTurnsLeft = intTotalBatteries * 2;
    int intBatteryPower = 3;
    ARR_TWO arrWorld = InitWorld(intTotalRows,intTotalCols,intTotalBatteries);
    char chInput = '\0';

    //Set main loop
    do
    {
        PrintScreen(arrWorld,intTotalRows,intTotalCols,intTurnsLeft,blnTorchOn,intBatteryPower);
        cin >> chInput;
        switch(tolower(chInput))
        {
        case 'w':
        case 'd':
        case 's':
        case 'a':
        case 'x':
            MovePlayer(arrWorld,intTotalRows,intTotalCols,intTurnsLeft,blnTorchOn,intBatteryPower,blnPitFall,chInput);
            break;
        case 'q':
            blnContinue = false;
        }
        blnEnd = TestEnd(intTurnsLeft, blnPitFall, blnSurvived, blnCaught, blnTorchOn);

    }while(blnContinue && !blnEnd);

    PrintScreen(arrWorld,intTotalRows,intTotalCols,intTurnsLeft,blnTorchOn,intBatteryPower);
    //Display end message
    cout << "*****************************************************************************" << endl;
    if(!blnSurvived)
        cout << "*                         Oh no you died!!!                                 *" << endl;
    else
        cout << "*                      Congratulations you survived!!                       *" << endl;
    if(blnCaught)
        cout << "*                    You were eaten by the Zoorkian Grue                    *" << endl;
    cout << "*****************************************************************************" << endl;

    Dealloc(arrWorld,intTotalRows);
    return 0;
}
```

```cpp
#ifndef LIBFUNCS_H_INCLUDED
#define LIBFUNCS_H_INCLUDED

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

using namespace std;

namespace ZorkSpace
{
    //World feature representation
    const char FEATURES[] = {'.','P','B','#'};
    const int VALUE_SPACE = 0;
    const int VALUE_PLAYER = 1;
    const int VALUE_BATTERY = 2;
    const int VALUE_TRAP = 3;

    //Error codes
    const int ERROR_NUM_ARGS = -1;
    const int ERROR_CONV = -2;
    const int ERROR_RANGE = -3;

    //2D array
    typedef int* ARR_ONE;
    typedef ARR_ONE* ARR_TWO;

    //Const values defined by the world
    const int CHANCE_TRAP = 15;
    const int CHANCE_GRUE = 20;

    //Function that converts strValue to int.
    int ToInt(string strValue);

    //Initialises a two dimensional array with all the settings defined by the problem
    ARR_TWO InitWorld(int intTRows, int intTCols, int intTBatts);

    //Outputs the screen
    void PrintScreen(ARR_TWO arrWorld, int intTRows, int intTCols, int intTurnsLeft, bool
blnTorchOn, int intBatterPower);

    //Moves the player
    void MovePlayer(ARR_TWO arrWorld, int intTRows, int intTRCols, int &intTurnsLeft,
                    bool &blnTorchOn, int &intBatteryPower, bool &blnPitFall, char chInput);

    //Tests if the game ends
    bool TestEnd(int intTurnsLeft, bool &blnPitFall, bool &blnSurvived, bool &blnCaught, bool
blnTorchOn);

    //Deallocates the 2D array
    void Dealloc(ARR_TWO& arrWorld, int intRows);

}


#endif // LIBFUNCS_H_INCLUDED

```

```cpp
#include "libFuncs.h"

namespace ZorkSpace
{
    //Generates a random number in the range intLower -> intUpper
    int GetRand(int intLower, int intUpper)
    {
        assert(intUpper>intLower);                //Tests whether intUpper is greater than intLower
        int intRange = intUpper - intLower + 1;
        return rand() % intRange + intLower;
    }

    //Safely converts strValue to an int.  Exits if it fails.
    int ToInt(string strValue)
    {
        stringstream ss {strValue};
        int intNum;
        ss >> intNum;
        if(ss.fail())
        {
            cerr << "Could not convert " << strValue << " to an Integer" << endl;
            exit(ERROR_CONV);
        }
        return intNum;
    }

    //Returns a random location that is empty
    void GetEmpty(ARR_TWO arrWorld, int intTRows, int intTCols, int &intRow, int &intCol)
    {
        intRow = GetRand(0,intTRows-1);
        intCol = GetRand(0,intTCols-1);
        while(arrWorld[intRow][intCol] != VALUE_SPACE)
        {
            intRow = GetRand(0,intTRows-1);
            intCol = GetRand(0,intTCols-1);
        }
    }

    //Deallocates the memory for the 2D array
    void Dealloc(ARR_TWO& arrWorld, int intRows)
    {
        assert(arrWorld!=nullptr);            //Make sure the 2D array actually exists.
        for(int r=0;r<intRows;r++)
            delete[] arrWorld[r];

        delete[] arrWorld;
        arrWorld = nullptr;
    }

    //Initialises the game world
    ARR_TWO InitWorld(int intTRows, int intTCols, int intTBatts)
    {
        //Declares the 2D array variable
        ARR_TWO arrWorld;
        //Allocates memory for the 2D array
        arrWorld = new ARR_ONE[intTRows];
        for(int r=0;r<intTRows;r++)
        {
            arrWorld[r] = new int[intTCols];

            //Initialises the values in the 2D array according to some of the rules.
            for(int c=0;c<intTCols;c++)
            {
                arrWorld[r][c] = VALUE_SPACE;
                //There is a 15% chance that a pit trap will appear
                if(GetRand(1,100)<=CHANCE_TRAP)
                {
                    arrWorld[r][c] = VALUE_TRAP;
                }
            }
        }

        //Place the player in an empty area
        int intRow = 0;
        int intCol = 0;
        GetEmpty(arrWorld,intTRows,intTCols,intRow,intCol);
        arrWorld[intRow][intCol] = VALUE_PLAYER;

        //Place the batteries in a random empty location
        for(int n=0;n<intTBatts-1;n++)
        {
            GetEmpty(arrWorld,intTRows,intTCols,intRow,intCol);
            arrWorld[intRow][intCol] = VALUE_BATTERY;
        }

        return arrWorld;
    }

    //Returns the intRow and intCol of the player
    void FindPlayer(ARR_TWO arrWorld, int intTRows, int intTCols, int &intRow, int &intCol)
    {
        intRow = -1;
        intCol = -1;
        for(int r=0;r<intTRows;r++)
        {
            for(int c=0;c<intTCols;c++)
            {
                if(arrWorld[r][c]==VALUE_PLAYER)
                {
                    intRow = r;
                    intCol = c;
                    return;
                }
            }
        }
    }

    //Outputs the game world
    void PrintScreen(ARR_TWO arrWorld, int intTRows, int intTCols, int intTurnsLeft, bool blnTorchOn, int intBatterPower)
    {
        system("cls");              //Clear the screen
        //We need to know the location of the player in case the torch is off
        int intPRow = -1;
        int intPCol = -1;
        FindPlayer(arrWorld,intTRows,intTCols,intPRow,intPCol);
        //Outputs each feature in the 2D array
        for(int r=0;r<intTRows;r++)
        {
            for(int c=0;c<intTCols;c++)
            {
                char chOutput = '\0';           //Declares a character that we will use to determine what needs to be output
                //If the torch is on, then we output whatever value is in the game world
                if(blnTorchOn)
                    chOutput = FEATURES[arrWorld[r][c]];
                //If the torch is not on and we are in the one square radius of the player, then we output the feature as well
                else if(abs(r-intPRow)<=1 && (abs(c-intPCol)<=1))
                    chOutput = FEATURES[arrWorld[r][c]];
                //If the torch is off and we are outside the one square radius of the player, then just output darkness.
                else
                    chOutput = ' ';
                cout << chOutput << " ";
            }
            cout << endl;
        }
        //Outputting some basic game-related information.
        cout << "Turns until dawn:" << intTurnsLeft << endl;
        cout << "Torch:" << ((blnTorchOn?"On":"Off")) << endl;
        cout << "Battery Power:" << intBatterPower << endl;
        cout << "Move Up: w" << endl
             << "Move Down: s" << endl
             << "Move Left: a" << endl
             << "Move Right: d" << endl
             << "Torch on\\off: x" << endl
             << "Quit: q" << endl;
    }

    //Returns true if intRow and intCol is inside the boundaries of the 2D array
    bool IsInWorld(int intTRows, int intTCols, int intRow, int intCol)
    {
        return (intRow >= 0 && intRow < intTRows &&
                intCol >= 0 && intCol < intTCols);
    }

    //Moves the player.  Updates intTurnsLeft, blnTorchOn, intBatteryPower and blnPitFall
    void MovePlayer(ARR_TWO arrWorld, int intTRows, int intTRCols, int &intTurnsLeft,
                    bool &blnTorchOn, int &intBatteryPower, bool &blnPitFall, char chInput)
    {
        assert(arrWorld!=nullptr);  //Make sure the 2D array has been allocated memory.
        //Getting the current row and col of the player
        int intCRow = -1;
        int intCCol = -1;
        FindPlayer(arrWorld, intTRows,intTRCols,intCRow,intCCol);
        //Setting the temporary destination row and col
        int intDRow = intCRow;
        int intDCol = intCCol;
        //We use this to determine whether we moved or not.  (Switching the torch on or off does not constitue a move)
        bool blnMoved = true;
        //Calculateing the destination location
        switch(tolower(chInput))
        {
        case 'w':
            intDRow--;
            break;
        case 's':
            intDRow++;
            break;
        case 'a':
            intDCol--;
            break;
        case 'd':
            intDCol++;
            break;
        //Switching the torch on or off
        case 'x':
            if(intBatteryPower>0)           //Can only be switched on if the torch has battery power
                blnTorchOn = !blnTorchOn;
            blnMoved = false;
            break;
        }

        //See if we are in the world and we have moved
        if(IsInWorld(intTRows,intTRCols,intDRow,intDCol) && blnMoved)
        {
            //If the torch is on, then we have to decrement the torch power.
            if(blnTorchOn)
            {
                if(intBatteryPower > 0)
                    intBatteryPower--;
                //If there is no more battery power in the torch, then switch the torch off
                else
                    blnTorchOn = false;
            }

            //If we move over a battery then increase the battery power
            if(arrWorld[intDRow][intDCol] == VALUE_BATTERY)
            {
                intBatteryPower+=3;
            }
            //If we stepped on a pit trap, then we will die.
            if(arrWorld[intDRow][intDCol] == VALUE_TRAP)
            {
                blnPitFall = true;
            }
            //Change the location of the player
            arrWorld[intDRow][intDCol] = VALUE_PLAYER;
            arrWorld[intCRow][intCCol] = VALUE_SPACE;
            //Update the number of turns that are left in the game.
            intTurnsLeft--;
        }

    }

    //Returns true if the game should end.  We evaluate if we fell into a trip and we also test if the Grue caught us
    bool TestEnd(int intTurnsLeft, bool &blnPitFall, bool &blnSurvived, bool &blnCaught, bool blnTorchOn)
    {
        //If there are no turns left, then we survived and the game ends.
        if(intTurnsLeft<=0)
        {
            blnSurvived = true;
            return true;
        }

        //If we fell into a pit trap, then we died.
        if(blnPitFall)
        {
            blnSurvived = false;
            return true;
        }

        //If the torch is not on, then we test to see if the Grue catches us.
        if(!blnTorchOn)
        {
            if(GetRand(1,100)<=CHANCE_GRUE)
            {
```

```
        blnCaught = true;
        blnSurvived = false;
        return true;
      }
    }

    //If none of the above are true, then we do not end the game.
    return false;
  }

}
```