# Index

# Arena:

```
//Arena.cpp
#include "Arena.h"
#include "Robot.h"
#include "globals.h"
//#include "utilities.cpp"
#include <iostream>
#include <string>
using namespace std;

///////////////////////////////////////////////////////////////////////////
//  Arena implementations
///////////////////////////////////////////////////////////////////////////

Arena::Arena(int nRows, int nCols)
{
    if (nRows <= 0 || nCols <= 0 || nRows > MAXROWS || nCols > MAXCOLS)
    {
        cout << "***** Arena created with invalid size " << nRows << " by "
            << nCols << "!" << endl;
        exit(1);
    }
    m_rows = nRows;
    m_cols = nCols;
    m_player = nullptr;
    m_nRobots = 0;
    m_robots[0] = nullptr;
}

Arena::~Arena()
{
    delete m_player;
    for (int i = 0; i < m_nRobots; i++) {
        //if (m_robots[i] != nullptr){
        delete m_robots[i];
        //}
    }
```

```
}

int Arena::rows() const
{
    return m_rows;
}

int Arena::cols() const
{
    return m_cols;
}

Player* Arena::player() const
{
    return m_player;
}

int Arena::robotCount() const
{
    return m_nRobots;
}

int Arena::nRobotsAt(int r, int c) const
{
    int count = 0;
    int row_r;
    int col_r;
    for (int i = 0; i < m_nRobots; i++) {
        if (m_robots[i] != nullptr) {
        row_r = m_robots[i]->row();
        col_r = m_robots[i]->col();
        if ((r == row_r) && (c == col_r)) {
            count++;
        }
        }
    }
    return count;
}
```

```
void Arena::display(string msg) const
{
    // Position (row,col) in the arena coordinate system is represented in
    // the array element grid[row-1][col-1]
    char grid[MAXROWS][MAXCOLS];
    int r, c;

    // Fill the grid with dots
    for (r = 0; r < rows(); r++)
        for (c = 0; c < cols(); c++)
            grid[r][c] = '.';

    // Indicate each robot's position

    for (r = 0; r < rows(); r++) {
        for (c = 0; c < cols(); c++) {
            int n = nRobotsAt(r+1, c+1);
            if (n == 0) {
                grid[r][c] = '.';
            }
            else if (n == 1) {
                grid[r][c] = 'R';
            }
            else if (n > 8) {
                grid[r][c] = '9';
            }
            else {
                grid[r][c] = char(n + '0');
                if (grid[r][c] == '1') {
                    grid[r][c] = 'R';
                }
            }
        }
    }

    // Indicate player's position
    if (m_player != nullptr)
    {
        // Set the char to '@', unless there's also a robot there,
```

```
        // in which case set it to '*'.
        char& gridChar = grid[m_player->row() - 1][m_player->col() - 1];
        if (gridChar == '.')
            gridChar = '@';
        else
            gridChar = '*';
    }

    // Draw the grid
    clearScreen();
    for (r = 0; r < rows(); r++)
    {
        for (c = 0; c < cols(); c++)
            cout << grid[r][c];
        cout << endl;
    }
    cout << endl;

    // Write message, robot, and player info
    cout << endl;
    if (msg != "")
        cout << msg << endl;
    cout << "There are " << robotCount() << " robots remaining." << endl;
    if (m_player == nullptr)
        cout << "There is no player." << endl;
    else
    {
        if (m_player->age() > 0)
            cout << "The player has lasted " << m_player->age() << " steps." <<
endl;
        if (m_player->isDead())
            cout << "The player is dead." << endl;
    }
}

bool Arena::addRobot(int r, int c)

{
    if (m_nRobots == MAXROBOTS) {
```

```
        return false;
    }
    else {
        //for (int i = 0; i <= m_nRobots; i++) {
            //if (m_robots[i] == nullptr) {
                m_robots[m_nRobots] = new Robot(this, r, c);
                m_nRobots++;
                return true;
            //}
        //}

    }
}

bool Arena::addPlayer(int r, int c)
{
    // Don't add a player if one already exists
    if (m_player != nullptr)
        return false;

    // Dynamically allocate a new Player and add it to the arena
    m_player = new Player(this, r, c);
    return true;
}

void Arena::damageRobotAt(int r, int c)
{
    bool live;
    for (int i = 0; i < m_nRobots; i++) {
        if (m_robots[i] != nullptr) {
            if ((m_robots[i]->row() == r) && (m_robots[i]->col() == c)) {
                live = m_robots[i]->takeDamageAndLive();
                if (!live){
                    delete m_robots[i];
                    for (int k=i; k<m_nRobots; k++) {
                        m_robots[k]=m_robots[k+1];
                    }
                    m_nRobots--;
```

```
                }
                break;
            }
        }
    }
}

bool Arena::moveRobots()
{
    for (int k = 0; k < m_nRobots; k++)
    {
        if (m_robots[k] != nullptr) {
            m_robots[k]->move();
            if ((m_robots[k]->row() == m_player->row()) && (m_robots[k]->col() ==
m_player->col())) {
                m_player->setDead();
            }
        }
    }

    // return true if the player is still alive, false otherwise
    return !m_player->isDead();
}
```

# Game:

```
//Game.cpp
#include "Game.h"
#include "globals.h"
#include "Arena.h"
//#include "Player.h"
//#include "utilities.cpp"
#include <iostream>
#include <string>
using namespace std;
////////////////////////////////////////////////////////////////////////
//  Game implementations
////////////////////////////////////////////////////////////////////////

Game::Game(int rows, int cols, int nRobots)
{
    if (nRobots > MAXROBOTS)
    {
        cout << "***** Trying to create Game with " << nRobots
            << " robots; only " << MAXROBOTS << " are allowed!" << endl;
        exit(1);
    }

    // Create arena
    m_arena = new Arena(rows, cols);

    // Add player
    int rPlayer = 1 + rand() % rows;
    int cPlayer = 1 + rand() % cols;
    m_arena->addPlayer(rPlayer, cPlayer);

    // Populate with robots
    while (nRobots > 0)
    {
        int r = 1 + rand() % rows;
        int c = 1 + rand() % cols;
        // Don't put a robot where the player is
```

```
        if (r == rPlayer && c == cPlayer)
            continue;
        m_arena->addRobot(r, c);
        nRobots--;
    }
}

Game::~Game()
{
    delete m_arena;
}

void Game::play()
{
    Player* p = m_arena->player();
    if (p == nullptr)
    {
        m_arena->display("");
        return;
    }
    string msg = "";
    do
    {
        m_arena->display(msg);
        msg = "";
        cout << endl;
        cout << "Move (u/d/l/r/su/sd/sl/sr/c//q): ";
        string action;
        getline(cin, action);
        if (action.size() == 0)
            p->stand();
        else
        {
            switch (action[0])
            {
            default:   // if bad move, nobody moves
                cout << '\a' << endl;  // beep
                continue;
            case 'q':
```

```
                return;
            case 'c':  // computer moves player
                msg = p->takeComputerChosenTurn();
                break;
            case 'u':
            case 'd':
            case 'l':
            case 'r':
                p->move(decodeDirection(action[0]));
                break;
            case 's':
                if (action.size() < 2)  // if no direction, nobody moves
                {
                    cout << '\a' << endl;  // beep
                    continue;
                }
                switch (action[1])
                {
                default:   // if bad direction, nobody moves
                    cout << '\a' << endl;  // beep
                    continue;
                case 'u':
                case 'd':
                case 'l':
                case 'r':
                    if (p->shoot(decodeDirection(action[1])))
                        msg = "Hit!";
                    else
                        msg = "Missed!";
                    break;
                }
                break;
            }
        }
        m_arena->moveRobots();
    } while (!m_arena->player()->isDead() && m_arena->robotCount() > 0);
    m_arena->display(msg);
}
```

# Player:

```
//Player.cpp
#include "Player.h"
#include "Arena.h"
#include "globals.h"
#include <iostream>
#include <string>
using namespace std;

///////////////////////////////////////////////////////////////////////////
//  Player implementations
///////////////////////////////////////////////////////////////////////////

Player::Player(Arena* ap, int r, int c)
{
    if (ap == nullptr)
    {
        cout << "***** The player must be in some Arena!" << endl;
        exit(1);
    }
    if (r < 1 || r > ap->rows() || c < 1 || c > ap->cols())
    {
        cout << "**** Player created with invalid coordinates (" << r
            << "," << c << ")!" << endl;
        exit(1);
    }
    m_arena = ap;
    m_row = r;
    m_col = c;
    m_age = 0;
    m_dead = false;
}

int Player::row() const
{
    return m_row;
}
```

```
int Player::col() const
{
    return m_col;
}

int Player::age() const
{
    return m_age;
}

string Player::takeComputerChosenTurn()
{

    int r = row();
    int c = col();

    int nUp = 6;
    int nDown = 6;
    int nRight = 6;
    int nLeft = 6;

    for (int i = 1; i < 6; i++) {
        if (nUp == 6) {
            nUp = i;
        }
        if (nDown == 6) {
            nDown = i;
        }
        if (nRight == 6) {
            nRight = i;
        }
        if (nLeft == 6) {
            nLeft = i;
        }

    }

    if (nUp == 6 && nDown == 6 && nRight == 6 && nLeft == 6) {
```

```
        stand();
        return "Stood.";
    }
    if (nUp < 6 && nDown < 6 && nRight < 6 && nLeft == 6) {
        move(LEFT);
        if (col() == 1) {
            return "stood";
        }
        return "Moved.";
    }
    else if (nUp < 6 && nDown < 6 && nRight == 6 && nLeft < 6) {
        move(RIGHT);
        if (col() == MAXCOLS) {
            return "stood";
        }
        return "Moved.";
    }
    else if (nUp < 6 && nDown == 6 && nRight < 6 && nLeft < 6) {
        move(DOWN);
        if (row() == MAXROWS) {
            return "stood";
        }
        return "Moved.";
    }
    else if (nUp == 6 && nDown < 6 && nRight < 6 && nLeft < 6) {
        move(UP);
        if (row() == 1) {
            return "stood";
        }
        return "Moved.";
    }
    if (nUp < 6 && nDown < 6 && nRight < 6 && nLeft < 6) {
        int arr[] = {nUp,nDown,nLeft,nRight};
        int dirs[] = {UP,DOWN,LEFT,RIGHT};
        int min = 6;
        int min_i = 0;
        bool hit;
        for (int i=0;i<4;i++) {
            if (arr[i]<min){
```

```
                min = arr[i];
                min_i = i;
            }
        }
        hit = shoot(dirs[min_i]);
        if (hit){
            return "Shot and hit!";
        }
        else{
            return "Shot and missed!";
        }
    }


}


void Player::stand()
{
    m_age++;
}

void Player::move(int dir)
{
    m_age++;
    switch (dir)
    {
    case UP:
        if (row() > 1) {
            m_row--;
        }
        break;
    case DOWN:
        if (row() < MAXROWS) {
            m_row++;
        }
        break;
    case LEFT:
        if (col() > 1) {
```

```
            m_col--;
        }
        break;
    case RIGHT:
        if (col() < MAXCOLS) {
            m_col++;
        }
        break;
    }
}

bool Player::shoot(int dir)
{
    m_age++;

    if (rand() % 3 == 0)  // miss with 1/3 probability
        return false;

    int r = row();
    int c = col();

    for (int i = 0; i < 5; i++) {
        if (UP) {
            r--;
        }
        else if (DOWN) {
            r++;

        }
        else if (RIGHT) {
            c++;

        }
        else if (LEFT) {
            c--;
        }
        int n = m_arena->nRobotsAt(r,c);
        if (n>0){
            m_arena->damageRobotAt(r,c);
```

```
            return true;
        }
    }
}

bool Player::isDead() const
{
    return m_dead;
}

void Player::setDead()
{
    m_dead = true;
}
```

# Robot:

```cpp
//Robot.cpp
#include "Robot.h"
#include "Arena.h"
#include "Player.h"
#include "globals.h"

#include <iostream>
#include <string>
using namespace std;

///////////////////////////////////////////////////////////////////////////
//  Robot implementation
///////////////////////////////////////////////////////////////////////////

Robot::Robot(Arena* ap, int r, int c)
{
    if (ap == nullptr)
    {
        cout << "***** A robot must be in some Arena!" << endl;
        exit(1);
    }
    if (r < 1 || r > ap->rows() || c < 1 || c > ap->cols())
    {
        cout << "***** Robot created with invalid coordinates (" << r << ","
            << c << ")!" << endl;
        exit(1);
    }
    m_arena = ap;
    m_row = r;
    m_col = c;
    health = 2;
}

int Robot::row() const
{
    return m_row;
}
```

```
int Robot::col() const
{
    return m_col;
}

void Robot::move()
{
    // Attempt to move in a random direction; if we can't move, don't move
    switch (rand() % 4)
    {
    case UP:
        if (row() > 1) {
            m_row--;
        }
        break;
    case DOWN:
        if (row() < MAXROWS) {
            m_row++;
        }
        break;
    case LEFT:
        if (col() > 1) {
            m_col--;
        }
        break;
    case RIGHT:
        if (col() < MAXCOLS) {
            m_col++;
        }
        break;
    }
}

bool Robot::takeDamageAndLive()
{
    health--;
    if (health == 0) {
        return false;
```

```
    }
    else {
        return true;
    }
}
```

# Utilities:

```
//utilities.cpp
#include "globals.h"

////////////////////////////////////////////////////////////////////////
//  Auxiliary function declarations
////////////////////////////////////////////////////////////////////////

int decodeDirection(char dir);
void clearScreen();


////////////////////////////////////////////////////////////////////////
//  Auxiliary function implementations
////////////////////////////////////////////////////////////////////////

int decodeDirection(char dir)
{
    switch (dir)
    {
    case 'u':  return UP;
    case 'd':  return DOWN;
    case 'l':  return LEFT;
    case 'r':  return RIGHT;
    }
    return -1;  // bad argument passed in!
}
```

# Main:

```cpp
// main.cpp
#include "Game.h"
#include "globals.h"

#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
using namespace std;

void doBasicTests();

////////////////////////////////////////////////////////////////////////
//  main()
////////////////////////////////////////////////////////////////////////

int main()
{
    // Initialize the random number generator.
    srand(static_cast<unsigned int>(time(0)));

    Game g(15, 18, 80);

    // Play the game
    g.play();
    doBasicTests();
}

////////////////////////////////////////////////////////////////////////
//  clearScreen implementations
////////////////////////////////////////////////////////////////////////

#ifdef _MSC_VER  //  Microsoft Visual C++

#include <windows.h>
```

```
void clearScreen()
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(hConsole, &csbi);
    DWORD dwConSize = csbi.dwSize.X * csbi.dwSize.Y;
    COORD upperLeft = { 0, 0 };
    DWORD dwCharsWritten;
    FillConsoleOutputCharacter(hConsole, TCHAR(' '), dwConSize, upperLeft,
        &dwCharsWritten);
    SetConsoleCursorPosition(hConsole, upperLeft);
}

#else  // not Microsoft Visual C++, so assume UNIX interface

#include <cstring>
#include <iostream>

void clearScreen()
{
    static const char* term = getenv("TERM");
    if (term == nullptr || strcmp(term, "dumb") == 0)
        std::cout << std::endl;
    else
    {
        static const char* ESC_SEQ = "\x1B[";  // ANSI Terminal esc seq:  ESC [
        std::cout << ESC_SEQ << "2J" << ESC_SEQ << "H" << std::flush;
    }
}

#endif

#include <cassert>
#include "Robot.h"
#include "Player.h"
#include "Arena.h"
#include "Game.h"

#define CHECKTYPE(f, t) { (void)(t)(f); }
```

```
void thisFunctionWillNeverBeCalled()
{

    Robot(static_cast<Arena*>(0), 1, 1);
    CHECKTYPE(&Robot::row,               int  (Robot::*)() const);
    CHECKTYPE(&Robot::col,               int  (Robot::*)() const);
    CHECKTYPE(&Robot::move,              void (Robot::*)());
    CHECKTYPE(&Robot::takeDamageAndLive, bool (Robot::*)());

    Player(static_cast<Arena*>(0), 1, 1);
    CHECKTYPE(&Player::row,                    int    (Player::*)() const);
    CHECKTYPE(&Player::col,                    int    (Player::*)() const);
    CHECKTYPE(&Player::age,                    int    (Player::*)() const);
    CHECKTYPE(&Player::isDead,                 bool   (Player::*)() const);
    CHECKTYPE(&Player::takeComputerChosenTurn, string (Player::*)());
    CHECKTYPE(&Player::stand,                  void   (Player::*)());
    CHECKTYPE(&Player::move,                   void   (Player::*)(int));
    CHECKTYPE(&Player::shoot,                  bool   (Player::*)(int));
    CHECKTYPE(&Player::setDead,                void   (Player::*)());

    Arena(1, 1);
    CHECKTYPE(&Arena::rows,          int     (Arena::*)() const);
    CHECKTYPE(&Arena::cols,          int     (Arena::*)() const);
    CHECKTYPE(&Arena::player,        Player* (Arena::*)() const);
    CHECKTYPE(&Arena::robotCount,    int     (Arena::*)() const);
    CHECKTYPE(&Arena::nRobotsAt,     int     (Arena::*)(int,int) const);
    CHECKTYPE(&Arena::display,       void    (Arena::*)(string) const);
    CHECKTYPE(&Arena::addRobot,      bool    (Arena::*)(int,int));
    CHECKTYPE(&Arena::addPlayer,     bool    (Arena::*)(int,int));
    CHECKTYPE(&Arena::damageRobotAt, void    (Arena::*)(int,int));
    CHECKTYPE(&Arena::moveRobots,    bool    (Arena::*)());

    Game(1,1,1);
    CHECKTYPE(&Game::play, void (Game::*)());
}

void doBasicTests()
{
```

```
{
    Arena a(10, 20);
    assert(a.addPlayer(2, 6));
    Player* pp = a.player();
    assert(pp->row() == 2  &&  pp->col() == 6  && ! pp->isDead());
    pp->move(UP);
    assert(pp->row() == 1  &&  pp->col() == 6  && ! pp->isDead());
    pp->move(UP);
    assert(pp->row() == 1  &&  pp->col() == 6  && ! pp->isDead());
    pp->setDead();
    assert(pp->row() == 1  &&  pp->col() == 6  && pp->isDead());
}
{
    Arena a(2, 2);
    assert(a.addPlayer(1, 1));
    assert(a.addRobot(2, 2));
    assert(a.robotCount() == 1);
    Player* pp = a.player();
    assert(a.moveRobots());
assert( ! pp->isDead());
    for (int k = 0; k < 1000  &&  ! pp->isDead()  &&  a.moveRobots(); k++)
        ;
assert(pp->isDead());
}
{
    Arena a(1, 4);
    assert(a.addPlayer(1, 1));
    assert(a.addRobot(1, 4));
    assert(a.addRobot(1, 4));
    assert(a.addRobot(1, 3));
    assert(a.robotCount() == 3  &&  a.nRobotsAt(1, 4) == 2);
    Player* pp = a.player();
    for (int k = 0; k < 1000  &&  a.robotCount() != 0; k++)
        pp->shoot(RIGHT);
    assert(a.robotCount() == 0);
    assert(a.addRobot(1, 4));
    for (int k = 0; k < 1000  &&  a.robotCount() != 0; k++)
        pp->takeComputerChosenTurn();
    assert(a.robotCount() == 0);
```

```
    }
    cout << "Passed all basic tests" << endl;
}
```

# Globals.h

```
//globals.h
#pragma once

int decodeDirection(char dir);
void clearScreen();

////////////////////////////////////////////////////////////////////////
// Manifest constants
////////////////////////////////////////////////////////////////////////

const int MAXROWS = 20;             // max number of rows in the arena
const int MAXCOLS = 40;             // max number of columns in the arena
const int MAXROBOTS = 130;          // max number of robots allowed
const int MAXSHOTLEN = 5;           // max number of steps you can shoot

const int UP = 0;
const int DOWN = 1;
const int LEFT = 2;
const int RIGHT = 3;
```