

Ren Li Assignment

by Ren Li

FILE	12098_REN_LI_REN_LI_ASSIGNMENT_177048_443208905.DOCX (421.31K)		
TIME SUBMITTED	22-MAR-2019 05:38PM (UTC+0000)	WORD COUNT	2814
SUBMISSION ID	103036976	CHARACTER COUNT	16029

Advanced AI & Machine Learning Assignment

Ren Li 1805212

1. Basic Theory of Influence Map

Influence map is an Artificial Intelligence technique used in computer games to help decision makings. It uses numerical values to represent spatial features and environment conditions of the game [1][2]. In practice, there will be some influence sources in the game map, whose influence will be propagated to other grids of map. There are two approaches to spread influence, one is using convolution filter and the other is using formula, which will be discussed separately in following subsections [3].

1.1 Convolution Filter Updating

Convolutional filtering technique has been applied successfully to many fields like digital signal processing, electrical engineering and image processing. For influence map, convolution filter is used to spread out the influence from sources. A typical convolution filter type is 3x3 Gaussian kernel filter.

For every grid in map, its surrounding influence value will be multiplied by the corresponding value of the filter. Then the products are added together to be new influence value of the central grid. Therefore, with filter sweeping over the whole map, the influence will spread out from the source. As an example, following is the convolutional computation process for the (4,2) cell of influence map with Gaussian kernel filter:

$$\begin{aligned} \text{NewInfluence} &= 18 \times \frac{1}{16} + 18 \times \frac{2}{16} + 18 \times \frac{1}{16} + 14 \times \frac{2}{16} + 16 \times \frac{4}{16} + 16 \times \frac{2}{16} + 14 \times \frac{1}{16} + 14 \times \frac{2}{16} + 14 \times \frac{1}{16} \\ &= \frac{18 + 36 + 18 + 28 + 64 + 32 + 14 + 28 + 14}{16} \\ &= \frac{252}{16} \\ &= 15.75 \end{aligned}$$

To analyse the space and time complexity of this method, we first write the outline algorithm of it.

```
initialize another influence map2 of the same shape with original influence map
for each row in original influence map:
    for each pixel in row:
        set sum to zero
        for each filter row in filter:
            for each element in filter row:
                multiply the filter element by the corresponding value of original influence map
                add the product result to sum
            set the pixel's value of influence map2 to sum
        update the original influence map with influence map2
```

From the above algorithm we can see that there are four for-loops, so the time complexity should be the product of execution times of each loop. Consequently, time complexity is $O(\text{map_height} * \text{map_width} * \text{filter_height} * \text{filter_width})$. The space complexity is $O(2 * \text{map_height} * \text{map_width})$ because there is another influence map used to store temporary results.

1.2 formula updating

It is also feasible to update the influence using formula. A typical updating formula is below:

$$Id = \frac{Io}{(e^{(d-r)}+1)}$$

Following is the computation of cell (4,4)'s influence value using this formula, and two influence sources are located at (1,1) and (1,6).

$$\begin{aligned} Id &= \frac{Io_1}{e^{d_1-r} + 1} + \frac{Io_2}{e^{d_2-r} + 1} \\ &= \frac{3.5}{e^{d_1-7} + 1} + \frac{4.3}{e^{d_2-7} + 1} \end{aligned}$$

We use two approaches to compute the distance, which are Euclidean Distance and Manhattan Distance. For Euclidean Distance,

$$\begin{aligned} d_1 &= \sqrt{(1-4)^2 + (1-4)^2} \\ &= \sqrt{3^2 + 3^2} \\ &= 3\sqrt{2} \\ d_2 &= \sqrt{(1-4)^2 + (6-4)^2} \\ &= \sqrt{3^2 + 2^2} \\ &= \sqrt{13} \\ Id &= \frac{3.5}{e^{3\sqrt{2}-7} + 1} + \frac{4.3}{e^{\sqrt{13}-7} + 1} \\ &= 3.29 + 4.16 \\ &= 7.45 \end{aligned}$$

For Manhattan Distance:

$$d_1 = |1 - 4| + |1 - 4|$$

$$= 3 + 3$$

$$= 6$$

$$d_2 = |1 - 4| + |6 - 4|$$

$$= 3 + 2$$

$$= 5$$

$$Id = \frac{3.5}{e^{6-7} + 1} + \frac{4.3}{e^{5-7} + 1}$$

$$= 2.56 + 3.79$$

$$= 6.35$$

To analyse the time complexity and space complexity, we firstly show the outline algorithm for updating influence map:

```
initialize another influence map2 of the same shape with original influence map
for each row in original influence map:
    for each pixel in row:
        set sum to zero
        for each influence source:
            compute the influence value using spreading formula
            add the result to sum
        set the value of corresponding pixel in influence map2 to sum
update the original influence map with influence map2
```

The time complexity of algorithm is $O(\text{map_height} * \text{map_width} * \text{number of influence source})$ and space complexity is $O(2 * \text{map_height} * \text{map_width})$. Because in practice there are usually large amounts of influence sources, the calculation consumption will become huge and unaffordable. Therefore, it is unlikely to directly use this approach to implement influence, and in order to improve the time efficiency, we can use convolution filter updating discussed above, or update with only the highest influence values and ignore others.

2. Influence Map Implementation

In this section, we will introduce the implementation details of the influence map.

2.1 Data structure

The influence map is represented by a 2-D array, where the first axis of array represents rows of map and the second axis represents columns. Except for the array to store influence value,

there are two another arrays, one is to store temporary calculation results and the other is to store the values of influence sources read from file. Influence sources are stored in *InfluenceSource.txt* file in the form of matrix, which is located in the same directory with .cpp file. In *InfluenceSource.txt* file, influence sources are marked by positive or negative values and other cells are 0.

2.2 Updating Algorithm

The Gaussian convolution filter is also represented by a 3×3 2-D array. While updating influence map, the influence value of cell will be computed with corresponding value of convolution filter. The filter will sweep over the whole map (except the border part), so influence will spread from the influence sources.

3. Influence Map Test

To test the effectiveness of influence map, we implement a robot guiding scenario where the robot needs to navigate its way to destination and avoid enemy positions. Destination is represented by red point and will send out positive influence. Enemy positions are represented by blue points and will send out negative influence. Robot is represented by a white point. When deciding moving direction, the robot will calculate a comprehensive value of positive and negative influence in each direction and choose the highest one. If there are multiple directions with the highest value, robot will randomly choose one to step.

To start the game, press *space button* to spread the influence and press *r button* to make robot move. Note that because the initial position of robot is far from positive influence source, if positive influence does not spread out enough, robot will not know where to move and will perform randomly. Therefore, it is better to start to guide robot when you can see the red influence is near it.

Followings are some screen shots of the application.

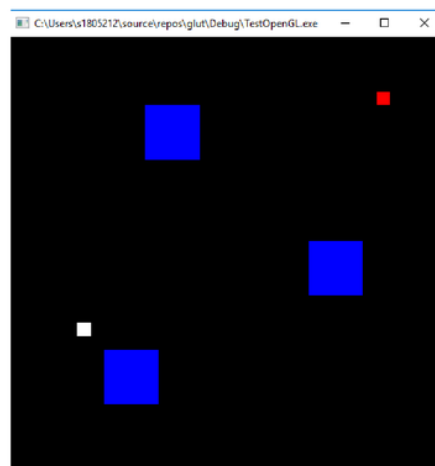


Figure 1: Influence map's initial status

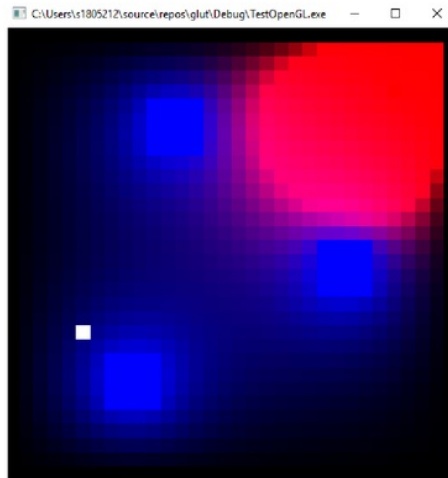


Figure 2: Influence has spread out

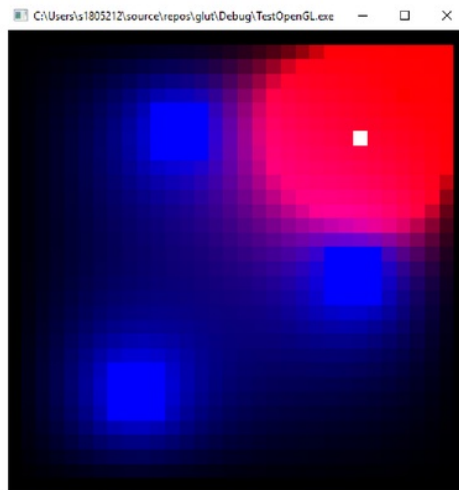


Figure 3: Robot has reached the destination

References

- [1] Wirth, Nathan, and Marcus Gallagher. "An influence map model for playing Ms. Pac-Man." 2008 IEEE Symposium on Computational Intelligence and Games. IEEE, 2008.
- [2] Svensson, Johan, and Stefan J. Johansson. "Influence Map-based controllers for Ms. PacMan and the ghosts." 2012 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2012.
- [3] Millington, I and Funge, J; Artificial Intelligence for Games, 2nd Ed; Charles River Media, 2009.
- [4] Krontiris, Athanasios, Kostas E. Bekris, and Mubbasir Kapadia. "ACUMEN: Activity-centric crowd authoring using influence maps." Proceedings of the 29th International Conference on Computer Animation and Social Agents. ACM, 2016.

- [5] Hagelbäck, Johan, and Stefan J. Johansson. "Using multi-agent potential fields in real-time strategy games." Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Appendix

Source.cpp file

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <cstdlib>
#include <fstream>
#include <string>
#include <sstream>
#include "glut.h"
#define GRIDSIZE 32
using namespace std;

typedef struct rgbtag {
    double r;
    double g;
    double b;
}rgb;

rgb grid[GRIDSIZE][GRIDSIZE];    // influence map
rgb gridtemp[GRIDSIZE][GRIDSIZE]; // store temporary comuting result
rgb gridInfSource[GRIDSIZE][GRIDSIZE]; // store influence source
double convFilter[3][3];
int robotRow;
int robotColumn;
//function prototypes
void init(void);
void display(void);
void drawBox(float w, float h, float l);
void reshape(int w, int h);
void updateTempMapPixel(int row, int column);
void setInfSource();
void robotMove();

void init()
// Initialization function
{
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);    // clears colour buffer
```

```

        glEnable(GL_DEPTH_TEST);
        glEnable(GL_CULL_FACE);
inside of poly's
        glFrontFace(GL_CCW);
wise polygons are out

        for (int i = 0; i < GRIDSIZE; i++) {
            for (int j = 0; j < GRIDSIZE; j++) {
                gridtemp[i][j].r = 0;
                gridtemp[i][j].g = 0;
                gridtemp[i][j].b = 0;
            }
        }

// read influence source from file
string line = "";
ifstream rfile;
rfile.open("InfluenceSource.txt");

if (!rfile.is_open()) {
    perror("Error open");
    exit(EXIT_FAILURE);
}

int i = 0;
while (getline(rfile, line)) {
    int j = 0;
    istringstream iss(line);
    do
    {
        //split string by whitespace
        string subs;
        iss >> subs;
        // transform string to double,
        // use red to represent positive influence, blue negative
        double lol = atof(subs.c_str());
        if (lol == 0) {
            gridInfSource[i][j].r = 0;
            gridInfSource[i][j].g = 0;
            gridInfSource[i][j].b = 0;
        }
        else if (lol > 0) {
            gridInfSource[i][j].r = lol;
            gridInfSource[i][j].g = 0;
            gridInfSource[i][j].b = 0;
        }
        else if (lol < 0) {
            gridInfSource[i][j].r = 0;
            gridInfSource[i][j].g = 0;
            gridInfSource[i][j].b = lol;
        }
    } while (iss.get() != '\n');
    i++;
}
// hidden surface removal
// do not calculate
// counter clock-

```

```

        gridInfSource[i][j].r = 0;
        gridInfSource[i][j].g = 0;
        gridInfSource[i][j].b = -lol;
    }
    j++;
} while (iss);
i++;
}

robotRow = -1;
robotColumn = -1;

// initialize Gaussian convlution filter
convFilter[0][0] = 0.0625; // 1/16
convFilter[0][1] = 0.125; // 1/8
convFilter[0][2] = 0.0625;
convFilter[1][0] = 0.125;
convFilter[1][1] = 0.25; // 1/4
convFilter[1][2] = 0.125;
convFilter[2][0] = 0.0625;
convFilter[2][1] = 0.125;
convFilter[2][2] = 0.0625;
}

void drawBox(float w, float h, float l)
//drawBox
//*****
**
//This function first translates a further x,y,z units then draws a box of
//width w, height h, length l
{
    glPushMatrix(); //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle of box
    //draw faces of box
    glBegin(GL_POLYGON);
    glVertex3f(w, l, h);
    glVertex3f(w, l, -h);
    glVertex3f(-w, l, -h);
    glVertex3f(-w, l, h);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex3f(w, l, h);
    glVertex3f(-w, l, h);
    glVertex3f(-w, -l, h);
    glVertex3f(w, -l, h);
    glEnd();
    glBegin(GL_POLYGON);

```

```

        glVertex3f(w, l, h);
        glVertex3f(w, -l, h);
        glVertex3f(w, -l, -h);
        glVertex3f(w, l, -h);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3f(-w, l, h);
        glVertex3f(-w, l, -h);
        glVertex3f(-w, -l, -h);
        glVertex3f(-w, -l, h);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3f(-w, -l, h);
        glVertex3f(-w, -l, -h);
        glVertex3f(w, -l, -h);
        glVertex3f(w, -l, h);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3f(w, -l, -h);
        glVertex3f(-w, -l, -h);
        glVertex3f(-w, l, -h);
        glVertex3f(w, l, -h);
        glEnd();
        glPopMatrix(); //restore previous modelview matrix so leaving things as you found
    }
}

// display
//*****
// desc: handles drawing of scene
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);           // clear
    screen and depth buffer
    glLoadIdentity();
    // reset modelview matrix

    gluLookAt(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); //reset camera position
    glColor3f(0, 0, 0); //set colour
    glTranslatef(0.5, 0.5, 0.5); //move by x,y and z units
    //tests grid by applying greyscale values to each position

    for (int i = 0; i < GRIDSIZE; i++) {
        for (int j = 0; j < GRIDSIZE; j++) {
            glColor3f(grid[i][j].r, 0, grid[i][j].b); //display influence

```

```

        if (i == robotRow && j == robotColumn) {
            glColor3f(1, 1, 1);    // display robot
        }

        glPushMatrix(); //save current position/orientation, etc.
        glTranslatef(i, j, 0);
        drawBox(1, 1, 1); //draw
        glPopMatrix(); //restore previous "current" position/orientation etc.
    }
}

glFlush(); //force drawing

glutSwapBuffers();           // bring backbuffer to foreground
//any errors then display error codes
GLenum errCode;
const GLubyte *errString;
if ((errCode = glGetError()) != GL_NO_ERROR)
{
    errString = gluErrorString(errCode);
    fprintf(stderr, "OpenGL error : %s\n", errString);
}

}

void keyboardread(unsigned char key, int x, int y)
{
    switch (key) {
        case 'r':
            robotMove(); //move robot
            glutPostRedisplay();
            break;
        case 't':

            for (int i = 0; i < GRIDSIZE; i++) {
                for (int j = 0; j < GRIDSIZE; j++) {
                    updateTempMapPixel(i, j);
                }
            }

            //copy calculated array into original
            for (int i = 0; i < GRIDSIZE; i++) {
                for (int j = 0; j < GRIDSIZE; j++) {
                    grid[i][j] = gridtemp[i][j];
                }
            }
    }
}

```

```

        setInfSource();

        glutPostRedisplay();
        break;
    }
}

void updateTempMapPixel(int row, int column)
// update the influence value of pixel specified by row and column, using convolution filter
{
    // not update edge pixels
    if (row == 0 || row == GRIDSIZE - 1)
    {
        return;
    }
    else if (column == 0 || column == GRIDSIZE - 1)
    {
        return;
    }
    else {
        double sumr = 0;
        double sumb = 0;
        int convCentre = 1;
        for (int i = -1; i < 2; i++) {
            for (int j = -1; j < 2; j++) {
                // convolutional caculation
                sumr += grid[row + i][column + j].r * convFilter[convCentre +
i][convCentre + j];
                sumb += grid[row + i][column + j].b * convFilter[convCentre +
i][convCentre + j];
            }
        }
        gridtemp[row][column].r = sumr;
        gridtemp[row][column].b = sumb;
    }
}

void setInfSource()
{
    // only set influence source values
    for (int i = 0; i < GRIDSIZE; i++) {
        for (int j = 0; j < GRIDSIZE; j++) {
            if (gridInfSource[i][j].r != 0 || gridInfSource[i][j].b != 0) {
                grid[i][j] = gridInfSource[i][j];
            }
        }
    }
}

```

```

    }
}

void robotMove() {
    if (robotRow == -1 || robotColumn == -1)
    {
        // init robot's position
        robotRow = 5;
        robotColumn = 10;
        return;
    }
    else
    {
        // use (weight*redValue-blueValue) as comprehensive influence value to
guide robot
        double redWeight = 100;
        // store comprehensive values in four directions#
        // initialize with very small value (if not being reset means can't go this
direction)
        // 0-up, 1-down, 2-left, 3-right
        double directionInf[4] = { -255, -255, -255, -255 };
        if (robotRow == 1 && robotColumn != 1 && robotColumn != GRIDSIZE - 2) {
            // can't go left
            directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
grid[robotRow][robotColumn + 1].b;
            directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
grid[robotRow][robotColumn - 1].b;
            directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
grid[robotRow + 1][robotColumn].b;
        }
        else if (robotRow == GRIDSIZE - 2 && robotColumn != 1 && robotColumn !=
GRIDSIZE - 2) {
            // can't go right
            directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
grid[robotRow][robotColumn + 1].b;
            directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
grid[robotRow][robotColumn - 1].b;
            directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
grid[robotRow - 1][robotColumn].b;
        }
        else if (robotColumn == 1 && robotRow != 1 && robotRow != GRIDSIZE - 2) {
            //can't go down
            directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
grid[robotRow][robotColumn + 1].b;
            directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
grid[robotRow - 1][robotColumn].b;

```

```

        directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
        grid[robotRow + 1][robotColumn].b;
    }
    else if (robotColumn == GRIDSIZE - 2 && robotRow != 1 && robotRow !=
    GRIDSIZE - 2) {
        // can't go up
        directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
        grid[robotRow][robotColumn - 1].b;
        directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
        grid[robotRow - 1][robotColumn].b;
        directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
        grid[robotRow + 1][robotColumn].b;
    }
    else if (robotRow == 1 && robotColumn == 1) {
        // can't go left and down
        directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
        grid[robotRow][robotColumn + 1].b;
        directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
        grid[robotRow + 1][robotColumn].b;
    }
    else if (robotRow == 1 && robotColumn == GRIDSIZE - 2) {
        // can't go left and up
        directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
        grid[robotRow][robotColumn - 1].b;
        directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
        grid[robotRow + 1][robotColumn].b;
    }
    else if (robotRow == GRIDSIZE - 2 && robotColumn == 1) {
        // can't go right and down
        directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
        grid[robotRow][robotColumn + 1].b;
        directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
        grid[robotRow - 1][robotColumn].b;
        //cout << "Inf: " << directionInf[0] << ", " << directionInf[1] << ", " <<
        directionInf[2] << ", " << directionInf[3] << endl;
    }
    else if (robotRow == GRIDSIZE - 2 && robotColumn == GRIDSIZE - 2) {
        // can't go right and up
        directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
        grid[robotRow][robotColumn - 1].b;
        directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
        grid[robotRow - 1][robotColumn].b;
    }
    else {
        //can move to all four directons
        directionInf[0] = redWeight * grid[robotRow][robotColumn + 1].r -
        grid[robotRow][robotColumn + 1].b;
    }

```

```

        directionInf[1] = redWeight * grid[robotRow][robotColumn - 1].r -
grid[robotRow][robotColumn - 1].b;
        directionInf[2] = redWeight * grid[robotRow - 1][robotColumn].r -
grid[robotRow - 1][robotColumn].b;
        directionInf[3] = redWeight * grid[robotRow + 1][robotColumn].r -
grid[robotRow + 1][robotColumn].b;
    }

    // find max value of directionInf array, if values are same, go randomly
    int maxDirection = 0;
    double maxValue = directionInf[0];
    for (int i = 1; i < 4; i++) {
        if (directionInf[i] > maxValue)
        {
            maxDirection = i;
            maxValue = directionInf[i];
        }
        else if (directionInf[i] == maxValue)
        {
            int randNumber = rand() % 100;
            if(randNumber<=50)
            {
                maxDirection = i;
            }
        }
    }

    // move robot
    if (maxDirection == 0) {
        robotColumn += 1;
    }
    else if (maxDirection == 1) {
        robotColumn -= 1;
    }
    else if (maxDirection == 2) {
        robotRow -= 1;
    }
    else if (maxDirection == 3) {
        robotRow += 1;
    }
    cout << "row: " << robotRow << ", column: " << robotColumn << endl;
    cout << "maxValue: " << maxValue << ", maxDirection: " << maxDirection <<
endl;
    cout << "directionInf: " << directionInf[0] << ", " << directionInf[1] << ", " <<
directionInf[2] << ", " << directionInf[3] << endl;
    cout << "red: " << grid[robotRow][robotColumn].r << endl;
    return;

```

```

    }

}

void reshape(int w, int h)
//reshape
//*****
//OpenGL reshape callback is called when the window is resized
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); //respecify size of view port to match new w
and h
    glMatrixMode(GL_PROJECTION); //go into projection mode
    glLoadIdentity(); //reset projection matrix
    glOrtho(0.0, GRIDSIZE, 0.0, GRIDSIZE, 0.0, 100.0); //defines orthoganal projection
(parallel
// viewing
volume
// window limits
of display
// left =0.0, right
=2.0,
// bottom = 0.0,
top = 2.0
// near =0.0, far
= 100.0
//gluPerspective(60.0,(GLfloat)w/(GLfloat)h,1.0,200.0); //redefine projection
matrix to match changes in w and h
    glMatrixMode(GL_MODELVIEW); //switch back to modelview mode
    glLoadIdentity(); //reset modelview matrix
    gluLookAt(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0); //reset camera position
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboardread);
    glutMainLoop();
    return 0;
}

```

Ren Li Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

79/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

GRADING FORM: SMU GENERIC_GJ

RESEARCH-INFORMED LITERATURE

7/10 Review of relevant literature undertaken and refs relevant and applicable. Good range of refs used here.

7**KNOWLEDGE AND UNDERSTANDING**

20/25 Good understanding of code and algorithms used to solve the problems set.

20**ANALYSIS**

16/20 Calculations carried out correctly. Well done. Analysis makes use of Big O notation and conclusions are appropriate for the problem specification. Good. Manhattan distance calculations covered here too. Well done. No evidence of test schedule in report though testing has been carried out.

16**PRACTICAL APPLICATION AND DEP.**

32/40 A good working application. Map loaded. AI Behaviour is reasonable. Both positive and negative

32.5

influences applied. Good. Test cases not documented.

SKILLS FOR PROFESSIONAL PRAC.



3.5/5 Refs good. Report structured. Some minor typos/grammar/style issues but overall v good.

3.5