

InfluenceMaps_OliverJordan_P1 44558

by Oliver Jordan

FILE	2911_OLIVER_JORDAN_INFLUENCEMAPS_OLIVERJORDAN_P144558_177048_2101026603.DOCX (112.38K)		
TIME SUBMITTED	22-MAR-2019 03:20PM (UTC+0000)	WORD COUNT	9687
SUBMISSION ID	103021883	CHARACTER COUNT	60377

Influence Maps

Oliver Jordan (P144558)

Agent AI and Machine Learning (M6X01080)

Gordon Dickers

Contents

1. Influence Map Theory	3
1.1. Convolution filters.....	3
1.2. Distance-Based Influences	5
2. Implementation	7
3. Appendix	8
3.1. Code	8
4. References	49

1. Influence Map Theory

Influence maps can be implemented in multiple ways, with each approach providing its own strengths. This section covers two of those approaches – convolution filters, and distance-based influences.

This section includes a worked example and pseudo code for each approach, as required for this paper.

1.1. Convolution filters

Convolution filters spread influence by “blurring” existing influence across a map. A map starts with enemy locations, and the only defined influence at the inception of the map is where these enemies are. For each pass, we look at a tile on the map, then use its neighbouring tiles to calculate the new influence for the tile we are looking at [1].

A convolution filter defines exactly how the neighbouring tiles affect the new influence value. A simple convolution example could be defined as the following:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

This filter would look at each neighbouring tile and take 1/9 of its influence value. The new influence value is the sum of all these – in this case resulting in the new influence being the mean of the surrounding tiles and the tile being assessed.

One popular convolution filter comes in the form a Gaussian filter, which has useful mathematical properties for programming [1]. The weightings in these filters are defined by the values on a Gaussian curve [2].

The following gaussian filter and influence map has been provided for this assignment:

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

	C 1	C 2	C 3	C 4	C 5	C 6
R 1	32	32	32	34	34	36
R 2	26	26	26	26	30	36
R 3	18	18	18	24	24	24
R 4	14	16	16	22	22	24
R 5	14	14	14	20	20	18
R 6	16	20	20	22	24	26
R 7	16	22	22	24	26	30
R 8	18	24	24	26	28	32

The following calculations detail how to find the new influence value for cell (R 3, C 4) using the provided filter.

Tile	Influence Value	Filter Weighting	Filtered Value
Tile Above Target (R 2, C 4)	26	2/16	$(26 \times 2) / 16 = 3.25$
Tile Above and Left of Target (R 2, C 3)	26	1/16	$26 / 16 = 1.625$
Tile Left of Target (R 3, C 3)	18	2/16	$(18 \times 2) / 16 = 2.25$
Tile Below and Left of Target (R 4, C 3)	16	1/16	$16 / 16 = 1$
Tile Below Target (R 4, C 4)	22	2/16	$(22 \times 2) / 16 = 2.75$
Tile Below and Right of Target (R 4, C 5)	22	1/16	$22 / 16 = 1.375$
Tile Right of Target (R 3, C 5)	24	2/16	$(24 \times 2) / 16 = 3$
Tile Above and Right of Target (R 2, C 5)	30	1/16	$30 / 16 = 1.875$
Target (R 3, C 4)	24	4/16	$(24 \times 4) / 16 = 6$
		SUM OF ALL FILTERED VALUES, NEW INFLUENCE VALUE FOR (R 3, C 4)	23.125  2

The following is a brief pseudo code example for how this might be implemented:

```
function spreadInfluence(float Tile[])
{
    For (each element i in Tile)
    {
        Float total = 0;
        total += (Tile[i] Influence Value * 4) / 16
        total += (Tile[i + width] Influence Value * 2) / 16
        total += Tile[i + width - 1] Influence Value / 16
        total += (Tile[i - 1] Influence Value * 2) / 16
    }
}
```

```

        total += Tile[i - 1 - width] Influence Value/16
        total += (Tile[i - width] Influence Value * 2)/16
        total += Tile[i + 1 - width] Influence Value/16
        total += (Tile[i + 1] Influence Value * 2)/16
        total += Tile[i + 1 + width] Influence Value/16
        NewTile[i] Influence Value = total
    }
}

Return NewTile
}

```

In this example, the following data structures are present:

- Tile – A one-dimensional array containing the influence values for each tile. To access a tile at position (x, y) you would access Tile[x + (y*width)]. The max size of the array would be Tile[mapWidth * mapHeight].
- NewTile – An identical array to Tile, so as not overwrite influence data before the function is finished with it. This is returned and would become the new primary tile array.
- Influence Value – a float associated with each tile containing the Influence value.
- Total – Used to add the filtered values and then place into the NewTile influence values.

This example has a timing of $O(N)$, where N is the number of tiles in the data set [3].

Using convolution filters provides a smooth blurring of influence values, however multiple passes are required to spread influence fully across the map. This can be counteracted by applying the filter every frame in a graphics program but adds some work to each frame [1].

1.2. Distance-Based Influences

A second approach uses a tile's distance from any influence emitter to calculate its own influence value [1]. An immediate apparent issue with this approach arises from the need to consider every emitter for every tile, giving any general algorithm implementing this approach a run time of $O(nm)$, where n is the number of emitters and m the number of tiles.

A modern RTS game may have thousands upon thousands of units each with their own influences, making this approach somewhat slow [1].

This can be offset by limiting the range of each unit's influence, only updating the influence values for tiles within a set range of each emitter. This would reduce the complexity down to $O(nr)$, where r is the number of tiles within the average radius of a unit's influence [1]. This, however, causes interactions arising from influences to occur only within that set radius of each unit – a group of units influence may travel very far when grouped up in the first approach, but when limited to a radius, the influence can only travel past the outermost units in a group for that set radius.

The following equation for distance-based influence calculations and influence map data have been provided for this assignment:

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

Influence Source	Position	Influence Value
Io1	(1,1)	3.5
Io2	(1,6)	4.3

The following calculations provide the influence for position (2,4), given emitters have an influence radius r of 5 (sqrt = Square Root):

Considered Influence source	Distance from Source	Influence of Source at (2,4)
Io1	Dist = sqrt $((2-1)^2 + (4-1)^2)$ Dist = sqrt (10) = 3.162 (to 3dp)	$Id = \frac{3.5}{(e^{(3.162-5)} + 1)}$ = 3.019 (to 3dp)
Io2	Dist = sqrt $((2-1)^2 + (4-6)^2)$ Dist = sqrt (5) = 2.236 (to 3dp)	$Id = \frac{4.3}{(e^{(2.236-5)} + 1)}$ = 4.045 (to 3dp)
	TOTAL INFLUENCE OF ALL CONSIDERED SOURCES, I.E INFLUENCE VALUE AT (2,4)	7.064

The following is a brief pseudo code example of this implementation:

Function calcInfluence(array Emitters, array Tiles)

```

{
    For(each element i in Tiles)
    {
        Tiles[i].influence = 0
        For (each element i in Emitters)
        {
            float e = 2.718
            dist = sqrt((Tiles[i].x - Emitter[i].x) ^ 2 + (Tiles[i].y - Emitter[i].y) ^ 2)
            float influence = Emitter[i].influence/(e^(dist-Emitters[i].range)+1)
            Tiles[i].influence += influence
        }
    }
}
return
}

```

Calculating the influence of every tile for every emitter this way results in complexity of $O(nm)$, where n is the number of tiles and m is the number of emitters, as stated above. A quicker approach would instead be to only calculate the influence of every tile within r tiles of each emitter, where r is the limiting range on the influence [1].

2. Implementation

The code provided for this assignment implements a distance-based calculation for influence, using a limiting radius for enemy AI and allowing the influence from the map's goal to be unlimited.

The player AI will attempt to follow the goals influence towards the goal; however, obstacles may prevent it from approaching the goal by simply looking for higher influence surrounding itself. To prevent the AI (quite literally) banging its head into a wall, a heuristic heavily inspired by Skorin-Kapov's Tabu Search [4] as detailed in Taillard's implementation [5]. Once a tile has been visited a defined number of times, the AI assumes that this location will not lead to any path to the goal and avoids moving to that tile regardless of its influence value, unless absolutely necessary.

Enemy AI wander the map randomly, forcing the player AI to flee if they get too close. The AI player succeeds and pauses once it steps on the same tile the goal is on.

3. Appendix

3.1. Code

3.1.1. AIcontroller.h

```
#include "Map.h"
#include "AIStates.h"
class State;
//STATE MACHINE AI CONTROLLER
class AIcontroller
{
private:
    //stores current state
    State* pCurrentState;

public:
    //constructor using map to initialise
    AIcontroller(Map* inMap, bool isEnemy);
    //default constructor sets up as if it's an enemy AI
    AIcontroller();
    //stores loaded map
    Map* map;

    //world pause will keep a Player from executing in searching state
    bool worldPause;
    //enemy pause will keep an Enemy from executing in enemy state
    bool enemyPause;
    //variable describing if AI is player or enemy
    bool enemy;

    //position structure
    struct position
    {
        int x = 0;
        int y = 0;
    };

    //stores AI position
    position Pos;
    //stores goal position, used for win condition checking
    position Goal;
    void Update(); //call this each time step

    //state functions
    //checks if goal is found
    bool foundGoal();
    //checks for danger
    bool dangerNear();
    //checks if out of danger
    bool escapedDanger();

    //function for changing state
    void ChangeState(State* nextState);

    //additional logic functions
    //moves AI toward goal
    void moveTowardGoal();
    //moves AI randomly (used only by enemy AI)
    void moveRandomly();
    //runs from danger
    void flee();
```

```
};
```

3.1.2. AIStates.h

```
#pragma once
#include "State.h"
//holds state information
class AIcontroller;

//each class is described by its name
//contains constructor, execute, enter and exit functions
//execute runs logic required while in that state, enter runs any code required for
that state when entering, exit runs code that may need to run when done with that
state
class searchingForGoal : public State
{
private:

public:
    searchingForGoal() {};
    //singleton

    static searchingForGoal* Instance();

    virtual void Execute(AIcontroller*);
    virtual void Enter(AIcontroller*);
    virtual void Exit(AIcontroller*);
};

class fleeingFromDanger : public State
{
private:

public:
    fleeingFromDanger() {};

    static fleeingFromDanger* Instance();

    virtual void Execute(AIcontroller*);
    virtual void Enter(AIcontroller*);
    virtual void Exit(AIcontroller*);
};

class foundGoal : public State
{
private:
public:
    foundGoal() {};

    static foundGoal* Instance();

    virtual void Execute(AIcontroller*);
    virtual void Enter(AIcontroller*);
    virtual void Exit(AIcontroller*);
};

class enemyState : public State
{
private:

public:
```

```

        enemyState() {};  

        //singleton  

        static enemyState* Instance();  

        virtual void Execute(AIcontroller*);  

        virtual void Enter(AIcontroller*);  

        virtual void Exit(AIcontroller*);  

};

```

3.1.3. Globals.h

```

#pragma once  

  

//stores common headers  

#include "Map.h"  

#include "MapLoader.h"  

#include "GL\glut.h"  

#include "obj.h"  

#include "objNewReader.h"  

#include "Matrix.h"  

#include "Vector.h"  

#include "MathLib.h"  

#include <cmath>  

#include "AIcontroller.h"

```

3.1.4. Map.h

```

#pragma once  

//map holds level data  

class Map  

{  

public:  

    //creates object, requires variablesso normally initialised with 1,1,1,1  

    Map(int width, int height, int infCount, int enemyCount);  

    ~Map();  

  

    //size of map  

    int width;  

    int height;  

    //number of enemies on the map  

    int enemies;  

    //total number of influences on the map  

    int totInf;  

    //position structure  

    struct pos  

    {  

        int x;  

        int y;  

    };  

    //enemy structure  

    struct enemy  

    {  

        pos position;  

    };  

    //goal structure  

    struct goal  

    {  

        pos position;  

    };  

    //holds goal information  

    goal goal;  

    //tile structure, contains information about each tile in the map

```



```

float lupCos[1024];
float lupSin[1024];

float conversionAngRad(float degrees);
float conversionAngDeg(float Radians);

float getCos(float angleInDegrees);
float getSin(float angleInDegrees);
Vector multiplyMatVect(Matrix inMat, Vector inVect);
void setIDMat(Matrix * inMat);
void setTransMat(Matrix * inMat, float x, float y, float z);
void setRotXMat(Matrix * inMat, float angle);
void setRotYMat(Matrix * inMat, float angle);
void setRotZMat(Matrix * inMat, float angle);
void setUniSc1Mat(Matrix * inMat, float scale);
void setSc1Mat(Matrix * inMat, float scaleX, float scaleY, float scaleZ);
Matrix multiplyMat(Matrix m1, Matrix m2);
Matrix addMat(Matrix m1, Matrix m2);

Matrix transposeMat(Matrix inputMat);
bool checkOrth(Matrix inputMat);

float dotProd(Vector v1, Vector v2);
float vectorLength(Vector inVect);

private:
    const float PI = 3.14;
};

```

3.1.7. Matrix.h

```

#pragma once

////////////////////////////////////
////////////////////////////////////
////

////
////    MATRIX WRITTEN FOR PREVIOUS ASSIGNMENT IN SECOND YEAR. REUSED AS IS WITH NO
CHANGES FROM THERE. ALL WORK MINE.    ////
////

////
////////////////////////////////////
////////////////////////////////////
class Matrix
{
public:
    Matrix();
    ~Matrix();

    float mtrx[4][4];

private:

};

```

3.1.8. Obj.h

```
#pragma once

//stores object details
class obj
{
private:
public:
    obj(int inVSize, int inVtSize, int inVNSize, int inFSize);
    ~obj();

    //stores count of each data type
    int vSize;
    int vtSize;
    int vnSize;
    int fSize;

    struct Vertex
    {
        float x, y, z;
    };

    struct vTexture
    {
        float x, y;
    };

    struct vNormal
    {
        float x, y, z;
    };

    struct Face
    {
        int v1, v2, v3;
        int vt1, vt2, vt3;
        int vn1, vn2, vn3;
    };

    //lists for each type
    Vertex * vList;
    vTexture * vtList;
    vNormal * vnList;
    Face * fList;
};
```

3.1.9. objNewreader.h

```
#pragma once
#include "obj.h"
#include <iostream>
#include <string>

////////////////////////////////////
////////////////////////////////////
////

////
//// OBJREADER WRITTEN AS AN ALL PURPOSE LIBRARY, USED IN MULTIPLE THIRD YEAR
PROJECTS. ALL WORK MINE.
////
```

```

////

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
class objNewReader
{
private:

public:
    objNewReader();
    ~objNewReader();

    //read file, pass back resulting obj from file
    obj readObjFile(char * inFile);
};

```

3.1.10. State.h

```

#pragma once
class AIcontroller;
//sets virtual functions for states
class State
{
public:

    virtual void Execute(AIcontroller*) = 0; //0 sets virtual void as not
    implemented

    virtual void Enter(AIcontroller*) = 0;

    virtual void Exit(AIcontroller*) = 0;
};

```

3.1.11. Vector.h

```

#pragma once

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

////
//// VECTOR WRITTEN FOR PREVIOUS ASSIGNMENT IN SECOND YEAR. REUSED AS IS WITH NO
CHANGES FROM THERE. ALL WORK MINE.
////

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
class Vector
{
public:
    Vector(float x, float y, float z);
    ~Vector();
    float vect[4];
private:
};

```

3.1.12. AIcontroller.cpp

```
#include "AIcontroller.h"
#include "AIStates.h"
#include <stdlib.h>
#include <time.h>
//sets appropriate attributes for Player AI
AIcontroller::AIcontroller(Map* inMap, bool isEnemy)
{
    map = inMap;
    pCurrentState = searchingForGoal::Instance();
    worldPause = true;
    enemy = isEnemy;
    //set seed for random functions
    srand(time(NULL));
}

//default constructor sets up Enemy AI
AIcontroller::AIcontroller()
{
    pCurrentState = enemyState::Instance();
    enemyPause = true;
    //set seed for random functions
    srand(time(NULL));
}

//update executes current state's execute function
void AIcontroller::Update()
{
    pCurrentState->Execute(this);
}

//checks if player is on same tile as goal
bool AIcontroller::foundGoal()
{
    if (Pos.x == Goal.x && Pos.y == Goal.y)
    {
        return true;
    }
    else
    {
        return false;
    }
}

//checks every surrounding tile to see if 1) enemy influence is high enough to be a
//danger and 2) it's not a wall. returns true if in danger
bool AIcontroller::dangerNear()
{
    if (map->tileList[(Pos.y - 1)*map->width + Pos.x].infIndex[1] > 90 && map->tileList[(Pos.y - 1)*map->width + Pos.x].infIndex[0] != 1000) //up
    {
        return true;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x - 1].infIndex[1] > 90 && map->tileList[(Pos.y - 1)*map->width + Pos.x - 1].infIndex[0] != 1000) //up left
    {
        return true;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x + 1].infIndex[1] > 90 && map->tileList[(Pos.y - 1)*map->width + Pos.x + 1].infIndex[0] != 1000) //up right
    {
        return true;
    }
}
```

```

    }
    else if (map->tileList[Pos.y*map->width + Pos.x - 1].infIndex[1] > 90 && map-
>tileList[Pos.y*map->width + Pos.x - 1].infIndex[0] != 1000) //left
    {
        return true;
    }
    else if (map->tileList[Pos.y*map->width + Pos.x + 1].infIndex[1] > 90 && map-
>tileList[Pos.y*map->width + Pos.x + 1].infIndex[0] != 1000) //right
    {
        return true;
    }
    else if (map->tileList[(Pos.y + 1)*map->width + Pos.x].infIndex[1] > 90 && map-
>tileList[(Pos.y + 1)*map->width + Pos.x].infIndex[0] != 1000) //down
    {
        return true;
    }
    else if (map->tileList[(Pos.y + 1)*map->width + Pos.x + 1].infIndex[1] > 90 &&
map->tileList[(Pos.y + 1)*map->width + Pos.x + 1].infIndex[0] != 1000) //up right
    {
        return true;
    }
    else if (map->tileList[(Pos.y + 1)*map->width + Pos.x - 1].infIndex[1] > 90 &&
map->tileList[(Pos.y + 1)*map->width + Pos.x - 1].infIndex[0] != 1000) //up left
    {
        return true;
    }
    else
    {
        return false;
    }
}

//checks every surrounding tile to see if danger is not near anymore, or if there is a
wall in the way. Returns true if there's no longer danger
bool AIcontroller::escapedDanger()
{
    if (map->tileList[(Pos.y - 1)*map->width + Pos.x].infIndex[1] > 90 && map-
>tileList[(Pos.y - 1)*map->width + Pos.x].infIndex[0] != 1000) //up
    {
        return false;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x - 1].infIndex[1] > 90 &&
map->tileList[(Pos.y - 1)*map->width + Pos.x - 1].infIndex[0] != 1000) //up left
    {
        return false;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x + 1].infIndex[1] > 90 &&
map->tileList[(Pos.y - 1)*map->width + Pos.x + 1].infIndex[0] != 1000) //up right
    {
        return false;
    }
    else if (map->tileList[Pos.y*map->width + Pos.x - 1].infIndex[1] > 90 && map-
>tileList[Pos.y*map->width + Pos.x - 1].infIndex[0] != 1000) //left
    {
        return false;
    }
    else if (map->tileList[Pos.y*map->width + Pos.x + 1].infIndex[1] > 90 && map-
>tileList[Pos.y*map->width + Pos.x + 1].infIndex[0] != 1000) //right
    {
        return false;
    }
}

```

```

        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x].infIndex[1] > 90 && map->tileList[(Pos.y + 1)*map->width + Pos.x].infIndex[0] != 1000) //down
        {
            return false;
        }
        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x + 1].infIndex[1] > 90 && map->tileList[(Pos.y + 1)*map->width + Pos.x + 1].infIndex[0] != 1000) //up right
        {
            return false;
        }
        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x - 1].infIndex[1] > 90 && map->tileList[(Pos.y + 1)*map->width + Pos.x - 1].infIndex[0] != 1000) //up left
        {
            return false;
        }
        else
        {
            return true;
        }
    }

    //changes to specified state, runs exit of current state and enter of new state
    void AIcontroller::ChangeState(State* nextState)
    {
        pCurrentState->Exit(this);
        pCurrentState = nextState;
        pCurrentState->Enter(this);
    }

    //will move toward goal based on goal's influence.
    void AIcontroller::moveTowardGoal()
    {
        //storing each tile data to be able to sort
        Map::tile * tiles[8];
        tiles[0] = &map->tileList[(Pos.y - 1)*map->width + Pos.x];
        tiles[1] = &map->tileList[(Pos.y - 1)*map->width + Pos.x - 1];
        tiles[2] = &map->tileList[(Pos.y - 1)*map->width + Pos.x + 1];
        tiles[3] = &map->tileList[Pos.y*map->width + Pos.x - 1];
        tiles[4] = &map->tileList[Pos.y*map->width + Pos.x + 1];
        tiles[5] = &map->tileList[(Pos.y + 1)*map->width + Pos.x];
        tiles[6] = &map->tileList[(Pos.y + 1)*map->width + Pos.x + 1];
        tiles[7] = &map->tileList[(Pos.y + 1)*map->width + Pos.x - 1];
        //bubblesort least to most visited
        while (1)
        {
            bool swapped = false;
            for (int i = 0; i < 7; i++)
            {
                Map::tile* temp;
                if (tiles[i + 1]->visited < tiles[i]->visited)
                {
                    temp = tiles[i + 1];
                    tiles[i + 1] = tiles[i];
                    tiles[i] = temp;
                    swapped = true;
                }
            }
            if (!swapped)
            {
                break;
            }
        }
        //End bubble sort
    }

```

```

        //bool maybeStuck flips true if there isn't a tile the AI can reach by
        following the goal's influence
        bool maybeStuck = false;
        //index pointing to which tile in our lil array is the least visited and not a
        wall
        int leastNotWall;
        for (int i = 0; i < 7; i++)
        {
            if (tiles[i]->infIndex[0] != 1000)
            {
                //sets least visited and not wall index
                leastNotWall = i;
                break;
            }
        }
        //flips to true provided the AI has found a tile to move to
        bool moved = false;
        //for each surrounding tile...
        for (int i = 0; i < 8; i++)
        {
            //if the tile's goal influence value is higher than the tile we're on...
            if ((tiles[i]->infIndex[2] - tiles[i]->infIndex[0]) > map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[2])
            {
                //and that tile has not yet been visited more than 50 times...
                if (tiles[i]->visited < 50)
                {
                    //step off current tile and increment visited count
                    map->tileList[(Pos.y)*map->width + Pos.x].visited++;
                    Pos.x = tiles[i]->position.x;
                    Pos.y = tiles[i]->position.y;
                    //move has been made, flip to true
                    moved = true;
                    //we can break early out of the loop if moving based on
                    influence. We always visit the least visited tile with higher influence by sorting
                    tiles before moving based on influence.
                    break;
                }
            }
        }
        //if none of the tiles were eligible...
        if (!moved)
        {
            //check each tile...
            for (int i = 0; i < 8; i++)
            {
                //if it's not a wall...
                if (tiles[i]->infIndex[0] != 1000)
                {
                    //move to this tile and increment visited counts
                    map->tileList[(Pos.y)*map->width + Pos.x].visited++;
                    Pos.x = tiles[i]->position.x;
                    Pos.y = tiles[i]->position.y;
                    //break out here... since tiles are sorted, when no
                    influence based move is made, we step onto the least visited tile regardless of it's
                    influence value
                    break;
                }
            }
        }
    }
}

```

```

//function picks a random direction, checks it's not a wall, and moves an AI to it.
void AIcontroller::moveRandomly()
{
    bool success = false;
    while (!success)
    {
        int direction = rand() % 8 + 1;
        switch (direction)
        {
            case 1: // up
                if (map->tileList[((Pos.y - 1) * map->width) + Pos.x].infIndex[0]
!= 1000)
                {
                    Pos.y = Pos.y - 1;
                    success = true;
                }
                break;
            case 2: //up right
                if (map->tileList[((Pos.y - 1) * map->width) + Pos.x +
1].infIndex[0] != 1000)
                {
                    Pos.y = Pos.y - 1;
                    Pos.x = Pos.x + 1;
                    success = true;
                }
                break;
            case 3: // right
                if (map->tileList[(Pos.y * map->width) + Pos.x + 1].infIndex[0]
!= 1000)
                {
                    Pos.x = Pos.x + 1;
                    success = true;
                }
                break;
            case 4: //right down
                if (map->tileList[((Pos.y + 1) * map->width) + Pos.x +
1].infIndex[0] != 1000)
                {
                    Pos.x = Pos.x + 1;
                    Pos.y = Pos.y + 1;
                    success = true;
                }
                break;
            case 5: //down
                if (map->tileList[((Pos.y + 1) * map->width) + Pos.x].infIndex[0]
!= 1000)
                {
                    Pos.y = Pos.y + 1;
                    success = true;
                }
                break;
            case 6: //down left
                if (map->tileList[((Pos.y + 1) * map->width) + Pos.x -
1].infIndex[0] != 1000)
                {
                    Pos.y = Pos.y + 1;
                    Pos.x = Pos.x - 1;
                    success = true;
                }
                break;
            case 7: // left

```

```

        if (map->tileList[(Pos.y * map->width) + Pos.x - 1].infIndex[0]
!= 1000)
        {
            Pos.x = Pos.x - 1;
            success = true;
        }
        break;
    case 8: //up left
        if (map->tileList[((Pos.y - 1) * map->width) + Pos.x -
1].infIndex[0] != 1000)
        {
            Pos.x = Pos.x - 1;
            Pos.y = Pos.y - 1;
            success = true;
        }
        break;
    }
}

//checks surrounding tiles for an enemy influence lower than the one it's on, check
it's not a wall, move to that tile.
void AIcontroller::flee()
{
    if (map->tileList[(Pos.y - 1)*map->width + Pos.x].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y - 1)*map-
>width + Pos.x].infIndex[0] != 1000) //up
    {
        map->tileList[(Pos.y)*map->width + Pos.x].visited++;
        Pos.y = Pos.y - 1;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x - 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y - 1)*map-
>width + Pos.x - 1].infIndex[0] != 1000) //up left
    {
        map->tileList[(Pos.y)*map->width + Pos.x].visited++;
        Pos.y = Pos.y - 1;
        Pos.x = Pos.x - 1;
    }
    else if (map->tileList[(Pos.y - 1)*map->width + Pos.x + 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y - 1)*map-
>width + Pos.x + 1].infIndex[0] != 1000) //up right
    {
        map->tileList[(Pos.y)*map->width + Pos.x].visited++;
        Pos.y = Pos.y - 1;
        Pos.x = Pos.x + 1;
    }
    else if (map->tileList[Pos.y*map->width + Pos.x - 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[Pos.y*map->width +
Pos.x - 1].infIndex[0] != 1000) //left
    {
        map->tileList[(Pos.y)*map->width + Pos.x].visited++;
        Pos.x = Pos.x - 1;
    }
    else if (map->tileList[Pos.y*map->width + Pos.x + 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[Pos.y*map->width +
Pos.x + 1].infIndex[0] != 1000) //right
    {
        map->tileList[(Pos.y)*map->width + Pos.x].visited++;
        Pos.x = Pos.x + 1;
    }
}

```

```

        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y + 1)*map-
>width + Pos.x].infIndex[0] != 1000) //down
        {
            map->tileList[(Pos.y)*map->width + Pos.x].visited++;
            Pos.y = Pos.y + 1;
        }
        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x + 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y + 1)*map-
>width + Pos.x + 1].infIndex[1] != 1000) //up right
        {
            map->tileList[(Pos.y)*map->width + Pos.x].visited++;
            Pos.y = Pos.y + 1;
            Pos.x = Pos.x + 1;
        }
        else if (map->tileList[(Pos.y + 1)*map->width + Pos.x - 1].infIndex[1] < map-
>tileList[(Pos.y)*map->width + Pos.x].infIndex[1] && map->tileList[(Pos.y + 1)*map-
>width + Pos.x - 1].infIndex[0] != 1000) //up left
        {
            map->tileList[(Pos.y)*map->width + Pos.x].visited++;
            Pos.y = Pos.y + 1;
            Pos.x = Pos.x - 1;
        }
    }
}

```

3.1.13. AIStates.cpp

```

#include "AIStates.h"
#include <stdio.h>
#include "Globals.h"
//creates one global state to use
searchingForGoal * searchingForGoal::Instance()
{
    static searchingForGoal instance;
    return &instance;
}
//checks if state should change based on factors, if not paused, move towards the
goal.
void searchingForGoal::Execute(AIcontroller* bot)
{
    if (bot->foundGoal())
    {
        bot->ChangeState(foundGoal::Instance());
    }
    else if (bot->dangerNear())
    {
        bot->ChangeState(fleeingFromDanger::Instance());
    }
    else if (!bot->worldPause)
    {
        bot->moveTowardGoal();
    }
}
//telling us what it's doing in this state
void searchingForGoal::Enter(AIcontroller* bot)
{
    printf("It's time to search for the goal!\n");
}
//telling us it's leaving the state
void searchingForGoal::Exit(AIcontroller* bot)
{
    printf("Something has gotten in the way of my search!\n");
}

```

```

}
//creates one global state to use
fleeingFromDanger * fleeingFromDanger::Instance()
{
    static fleeingFromDanger instance;
    return &instance;
}
//checks if we're out of danger, or if we've stumbled onto the goal while fleeing,
otherwise will continue to flee
void fleeingFromDanger::Execute(AIcontroller* bot)
{
    if (bot->escapedDanger())
    {
        bot->ChangeState(searchingForGoal::Instance());
    }
    else if (bot->foundGoal())
    {
        bot->ChangeState(foundGoal::Instance());
    }
    else
    {
        bot->flee();
    }
}
//tells us it's being a coward and running away
void fleeingFromDanger::Enter(AIcontroller* bot)
{
    printf("Discovered enemy nearby, relocating...\n");
}
//when it's safe, lets us know
void fleeingFromDanger::Exit(AIcontroller* bot)
{
    printf("Phew, that was close...\n");
}
//global instance
foundGoal * foundGoal::Instance()
{
    static foundGoal instance;
    return &instance;
}
//pauses itself when it finds the goal
void foundGoal::Execute(AIcontroller * bot)
{
    if (!bot->worldPause)
    {
        bot->worldPause = true;
    }
}
//tells us the goal is found and where it is
void foundGoal::Enter(AIcontroller * bot)
{
    printf("Found the goal! It's at the following co-ords: (%d, %d)\n", bot->Pos.x,
bot->Pos.y);
}
//for this project, goal state is program end, so we never need to exit
void foundGoal::Exit(AIcontroller * bot)
{
}
//global state
enemyState * enemyState::Instance()
{
    static enemyState instance;

```

```

        return &instance;
    }
    //enemy state for enemy AIs only. AIs move randomly when unpaused
    void enemyState::Execute(AIcontroller * bot)
    {
        if (!bot->enemyPause)
        {
            bot->moveRandomly();
        }
    }
    //enemy AIs never enter or exit this state, no code required.
    void enemyState::Enter(AIcontroller * bot)
    {
    }

    void enemyState::Exit(AIcontroller * bot)
    {
    }

```

3.1.14. Main.cpp

```

#include "Globals.h"

//set const math number
const int SCREEN_HEIGHT = 1080;
const int SCREEN_WIDTH = 1920;
int TIMER = 10;
const float e = 2.178;
//infViewMode defines which influences are being shown
int infViewMode = 0;
//setting up blank objects ready to be loaded into
obj Ground(1, 1, 1, 1);
obj Wall(1, 1, 1, 1);
obj Entity(1, 1, 1, 1);
obj Goal(1, 1, 1, 1);
//setting up blank map ready to be loaded into
Map map(1, 1, 1, 1);
//AIcontroller for player initialised with pointer to the map and a flag detailing if
it's an enemy set to false (not truly used for anything, but available for possible
expansion)
AIcontroller AIPlayer(&map, false);
//set an array of AIs to be used for enemy entities
AIcontroller * enemies;

//camera control variables...
float strafe = 0.0f;
float fly = 0.0f;
float lookAtX = 1.0f;
float lookAtY = 1.0f;
float lookAtZ = 1.0f;
float posX = 20.0f;
float posY = 10.0f;
float posZ = 20.0f;

//matricies for rotating forward facing vector to find left and right vector for
strafing
Matrix rot90;
Matrix rot270;
//math lib (used from previous work) and vectors to hold relevant directions
MathLib maths;

```

```

//setting vectors to store left right and forward vectors
Vector right(1,1,1);
Vector left(1,1,1);
Vector fwd(1, 1, 1);
//setting vectors to store rotation values
int yAxisRotTot = 0;
int xAxisRotTot = 0;
//general rotation matrix to be used to create infrequently used rotation matrices
Matrix rotation;
Vector newLookAt(1, 1, 1);
//float to hold length for normalisation
float length;
//function prototypes for influence calculations
void calcInf(Map* inMap);
void cleanEnemyInf(Map* inMap);
//prototype function for game loop, called repeatedly using glutTimerFunc
void round(int n);
//function called when mouse moves, rotates camera
void rotCam(int x, int y);
//bool used to lock and unlock the mouse from camera controls
bool lockMouse = true;

//round is the game loop, called repeatedly from the moment the program starts. AI
updates and influence calculation happen in here.
void round(int n)
{
    //clean enemy influence from map
    cleanEnemyInf(&map);
    //calculate influences again
    calcInf(&map);
    //update player
    AIPlayer.Update();
    //update every enemy
    for (int i = 0; i < map.enemies; i++)
    {
        enemies[i].Update();
    }
    if (lockMouse)
    {
        glutWarpPointer(SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2);
    }
    //redisplay
    glutPostRedisplay();
    //call again after 10ms
    glutTimerFunc(TIMER, round, 0);
}

void rotCam(int x, int y)
{
    if (lockMouse)
    {
        int distMidX = (SCREEN_WIDTH / 2) - x;
        if (yAxisRotTot != 360 || yAxisRotTot < 360)
        {
            yAxisRotTot = ((yAxisRotTot + distMidX) % 360);
        }
        else
        {
            yAxisRotTot = 0;
        }
    }
}

```

```

        int distMidY = (SCREEN_HEIGHT / 2) - y;

        if (xAxisRotTot >= -90 && xAxisRotTot <= 90)
        {
            xAxisRotTot = (xAxisRotTot + distMidY) % 360;
        }

        if(xAxisRotTot < -90)
        {
            xAxisRotTot = -90;
        }
        else if (xAxisRotTot > 90)
        {
            xAxisRotTot = 90;
        }

        lookAtX = sin(float(yAxisRotTot)*(0.0175)) + posX;
        lookAtY = sin(float(xAxisRotTot)*(0.0175)) + posY;
        lookAtZ = cos(float(yAxisRotTot)*(0.0175)) *
cos(float(xAxisRotTot)*(0.0175)) + posZ;
    }

}

//openGL keyboard read, containing input control
void keyboardread(unsigned char key, int x, int y)
{
    switch (key) {
        // 'a' strafes the camera left
        case 'a':
            right = maths.multiplyMatVect(rot270, Vector(lookAtX - posX, 0, lookAtZ
- posZ));
            length = maths.vectorLength(right);
            right.vect[0] = right.vect[0] / length;
            right.vect[1] = right.vect[1] / length;
            right.vect[2] = right.vect[2] / length;
            posX = posX - right.vect[0];
            posZ = posZ - right.vect[2];
            lookAtX = lookAtX - right.vect[0];
            lookAtZ = lookAtZ - right.vect[2];
            //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
            glutPostRedisplay();
            break;
        case 'A':
            right = maths.multiplyMatVect(rot270, Vector(lookAtX - posX, 0, lookAtZ
- posZ));
            length = maths.vectorLength(right);
            right.vect[0] = right.vect[0] / length;
            right.vect[1] = right.vect[1] / length;
            right.vect[2] = right.vect[2] / length;
            posX = posX - right.vect[0];
            posZ = posZ - right.vect[2];
            lookAtX = lookAtX - right.vect[0];
            lookAtZ = lookAtZ - right.vect[2];
            //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
            glutPostRedisplay();
            break;
        // 'd' strafes the camera right
        case 'd':
    
```

```

posZ));
    right = maths.multiplyMatVect(rot90, Vector(lookAtX - posX, 0, lookAtZ -
length = maths.vectorLength(right);
right.vect[0] = right.vect[0] / length;
right.vect[1] = right.vect[1] / length;
right.vect[2] = right.vect[2] / length;
posX = posX - right.vect[0];
posZ = posZ - right.vect[2];
lookAtX = lookAtX - right.vect[0];
lookAtZ = lookAtZ - right.vect[2];
//printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
glutPostRedisplay();
glutPostRedisplay();
break;
case 'D':
    right = maths.multiplyMatVect(rot90, Vector(lookAtX - posX, 0, lookAtZ -
posZ));
    length = maths.vectorLength(right);
    right.vect[0] = right.vect[0] / length;
    right.vect[1] = right.vect[1] / length;
    right.vect[2] = right.vect[2] / length;
    posX = posX - right.vect[0];
    posZ = posZ - right.vect[2];
    lookAtX = lookAtX - right.vect[0];
    lookAtZ = lookAtZ - right.vect[2];
    //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
    glutPostRedisplay();
    glutPostRedisplay();
    break;
    //'s' moves the camera backwards
case 's':
    fwd = Vector(lookAtX - posX, lookAtY - posY, lookAtZ - posZ);
    length = maths.vectorLength(fwd);
    fwd.vect[0] = fwd.vect[0] / length;
    fwd.vect[1] = fwd.vect[1] / length;
    fwd.vect[2] = fwd.vect[2] / length;
    posX = posX - fwd.vect[0];
    posY = posY - fwd.vect[1];
    posZ = posZ - fwd.vect[2];
    lookAtX = lookAtX - fwd.vect[0];
    lookAtY = lookAtY - fwd.vect[1];
    lookAtZ = lookAtZ - fwd.vect[2];
    //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
    glutPostRedisplay();
    break;
case 'S':
    fwd = Vector(lookAtX - posX, lookAtY - posY, lookAtZ - posZ);
    length = maths.vectorLength(fwd);
    fwd.vect[0] = fwd.vect[0] / length;
    fwd.vect[1] = fwd.vect[1] / length;
    fwd.vect[2] = fwd.vect[2] / length;
    posX = posX + fwd.vect[0];
    posY = posY + fwd.vect[1];
    posZ = posZ + fwd.vect[2];
    lookAtX = lookAtX - fwd.vect[0];
    lookAtY = lookAtY - fwd.vect[1];
    lookAtZ = lookAtZ - fwd.vect[2];
    //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
    glutPostRedisplay();
    break;
    //'w' moves the camera forward
case 'w':

```

```

        fwd = Vector(lookAtX - posX, lookAtY - posY, lookAtZ - posZ);
        length = maths.vectorLength(fwd);
        fwd.vect[0] = fwd.vect[0] / length;
        fwd.vect[1] = fwd.vect[1] / length;
        fwd.vect[2] = fwd.vect[2] / length;
        posX = posX + fwd.vect[0];
        posY = posY + fwd.vect[1];
        posZ = posZ + fwd.vect[2];
        lookAtX = lookAtX + fwd.vect[0];
        lookAtY = lookAtY + fwd.vect[1];
        lookAtZ = lookAtZ + fwd.vect[2];
        //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
        glutPostRedisplay();
        break;
    case 'W':
        fwd = Vector(lookAtX - posX, lookAtY - posY, lookAtZ - posZ);
        length = maths.vectorLength(fwd);
        fwd.vect[0] = fwd.vect[0] / length;
        fwd.vect[1] = fwd.vect[1] / length;
        fwd.vect[2] = fwd.vect[2] / length;
        posX = posX + fwd.vect[0];
        posY = posY + fwd.vect[1];
        posZ = posZ + fwd.vect[2];
        lookAtX = lookAtX + fwd.vect[0];
        lookAtY = lookAtY + fwd.vect[1];
        lookAtZ = lookAtZ + fwd.vect[2];
        //printf("POS X: %f Y: %f Z: %f\n", posX, posY, posZ);
        glutPostRedisplay();
        break;
    // 'y' is the pause toggle for enemy AI - pausing the AI essentially just
    stops them moving
    case 'y':
        for (int i = 0; i < map.enemies; i++)
        {
            if (enemies[i].enemyPause)
            {
                enemies[i].enemyPause = false;
            }
            else
            {
                enemies[i].enemyPause = true;
            }
        }
        glutPostRedisplay();
        break;
    case 'Y':
        for (int i = 0; i < map.enemies; i++)
        {
            if (enemies[i].enemyPause)
            {
                enemies[i].enemyPause = false;
            }
            else
            {
                enemies[i].enemyPause = true;
            }
        }
        glutPostRedisplay();
        break;
    // p toggles the cursor being active
    case 'p':
        if (lockMouse)

```

```

        {
            lockMouse = false;
        }
        else
        {
            lockMouse = true;
        }
        glutPostRedisplay();
        break;
case 'P':
    if (lockMouse)
    {
        lockMouse = false;
    }
    else
    {
        lockMouse = true;
    }
    glutPostRedisplay();
    break;
//t toggles the player AI on and off
case 't':
    if (AIPlayer.worldPause)
    {
        AIPlayer.worldPause = false;
    }
    else
    {
        AIPlayer.worldPause = true;
    }
    glutPostRedisplay();
    break;
case 'T':
    if (AIPlayer.worldPause)
    {
        AIPlayer.worldPause = false;
    }
    else
    {
        AIPlayer.worldPause = true;
    }
    glutPostRedisplay();
    break;
//# changes which influence is being displayed on the map
case '#':
    if (infViewMode != 4)
    {
        infViewMode++;
    }
    else
    {
        infViewMode = 0;
    }
    glutPostRedisplay();
    break;
//[ and ] increase and decrease the time between ai updates
case '[':
    if (TIMER != 10)
    {
        TIMER--;
    }
    break;

```

```

        case ']':
            TIMER++;
            break;
            // . immediately resets update timer to lowest value
        case '.':
            TIMER = 10;
            break;
            // , positions the camera above the map so that you can see the majority
of it, in case the camera controls are too cumbersome
        case ',':
            lockMouse = false;
            posX = (map.width / 2) * 5;
            posZ = (map.height / 2) * 5;
            posY = 100;
            lookAtX = posX;
            lookAtZ = posZ;
            lookAtY = 100;
            maths.setRotZMat(&rotation, 270);
            newLookAt = Vector(lookAtX, lookAtY, lookAtZ);
            length = maths.vectorLength(newLookAt);
            newLookAt = maths.multiplyMatVect(rotation, newLookAt);
            lookAtX = newLookAt.vect[0];
            lookAtY = newLookAt.vect[1];
            lookAtZ = newLookAt.vect[2];
            break;
    }
}

//openGL init function
void init(void)
{
    //initialise rotation matrices that are used frequently
    maths.setRotYMat(&rot90, 90);
    maths.setRotYMat(&rot270, 270);
    //OpenGL initialisation goes here
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glutPassiveMotionFunc(rotCam);
    GLfloat mat_specular[] = { 1, 1, 1.0, 1.0 };
    GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_ambient[] = { 1, 1, 1, 1.0 };
    GLfloat mat_shininess[] = { 12.8 };
    //glEnable(GL_LIGHTING);
    //glEnable(GL_LIGHT0);
    GLfloat light_position[] = { 0.5, 5, 0.5, 0.0 };

    glClearColor(0.0, 0.0, 0.0, 0.0); //reset "empty" background colour
    glShadeModel(GL_SMOOTH);

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

    //glShadeModel(GL_SMOOTH);
    glEnable(GL_CULL_FACE);

```

```

        glCullFace(GL_FRONT);
        glFrontFace(GL_CCW);
        glClearDepth(1.0f);
        glutInitDisplayMode(GLUT_DEPTH);
        glEnable(GL_DEPTH_TEST);
        glDepthFunc(GL_LEQUAL);
        //call looping function at start of program to keep running throughout
        glutTimerFunc(1, round, 0);
    }

    //draw object from loaded obj object (assumes triangulation already occurred)
    void drawObj(obj object)
    {
        for (int i = 0; i < object.fSize; i++)
        {
            glBegin(GL_TRIANGLES);
            glVertex3f(object.vList[object.fList[i].v1].x,
            object.vList[object.fList[i].v1].y, object.vList[object.fList[i].v1].z);
            glVertex3f(object.vList[object.fList[i].v3].x,
            object.vList[object.fList[i].v3].y, object.vList[object.fList[i].v3].z);
            glVertex3f(object.vList[object.fList[i].v2].x,
            object.vList[object.fList[i].v2].y, object.vList[object.fList[i].v2].z);
            glEnd();
        }
    }

    //draw object at a specified position
    void drawObjAt(obj object, float xPos, float yPos, float zPos)
    {
        for (int i = 0; i < object.fSize; i++)
        {
            glBegin(GL_TRIANGLES);
            glVertex3f(object.vList[object.fList[i].v1].x + xPos,
            object.vList[object.fList[i].v1].y + yPos, object.vList[object.fList[i].v1].z + zPos);
            glVertex3f(object.vList[object.fList[i].v3].x + xPos,
            object.vList[object.fList[i].v3].y + yPos, object.vList[object.fList[i].v3].z + zPos);
            glVertex3f(object.vList[object.fList[i].v2].x + xPos,
            object.vList[object.fList[i].v2].y + yPos, object.vList[object.fList[i].v2].z + zPos);
            glEnd();
        }
    }

    //draws a loaded map, specific function to read codes and draw appropriate object
    void drawMap(Map inMap)
    {
        //for each row
        for (int i = 0; i < inMap.height; i++)
        {
            //for each column
            for (int j = 0; j < inMap.width; j++)
            {
                //influence trackers
                float inf = 0;
                float inf2 = 0;
                //ground is 5 units wide
                switch (inMap.tileList[(i*inMap.width) + j].code)
                {
                    default:
                        break;
                    //code 0 is a wall
                    case 0:
                        //set wall colour
                        glColor3f(1, 0, 1);
                }
            }
        }
    }

```

```

//draw at current position multiplied by 5 to accomodate
object size
drawObjAt(Wall, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
break;
//code 1 is a floor tile
case 1:
//depending on which influence we wish to view, or none at
all, set colour of floor tile according to the tiles influence value
if (infViewMode == 0)
{
//all solid red for no influence
glColor3f(1, 0, 0);
}
else if (infViewMode == 1)
{
//according to enemy influence
inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
glColor3f(inf, 0, 0);
}
else if (infViewMode == 3)
{
//according to both enemy and goal influence
inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
inf2 = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
glColor3f(inf, 0, inf2);
}
else if (infViewMode == 4)
{
//according to visited tile influence
inf = (inMap.tileList[(i*inMap.width) + j].visited
* 2) / 100.0f;
if (inMap.tileList[(i*inMap.width) + j].visited >=
50)
{
//once tile has been visited 50 times, colour
red
glColor3f(1, 0, 0);
}
else
{
//depending how close to visited 50 times,
colour a shade of yellow
glColor3f(inf, inf, 0);
}
}
else
{
//according to goal's influence
inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
glColor3f(inf, 0, 0);
}
//draw the ground tile
drawObjAt(Ground, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
break;
//code 2 is a player object, however when drawing the map
it is treated the same as a ground tile as above

```

```

        case 2:
            if (infViewMode == 0)
            {
                glColor3f(1, 0, 0);
            }
            else if (infViewMode == 1)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                glColor3f(inf, 0, 0);
            }
            else if (infViewMode == 3)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                inf2 = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
                glColor3f(inf, 0, inf2);
            }
            else if (infViewMode == 4)
            {
                inf = (inMap.tileList[(i*inMap.width) + j].visited
* 2) / 100.0f;
                glColor3f(inf, inf, 0);
            }
            else
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
                glColor3f(inf, 0, 0);
            }
            drawObjAt(Ground, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
            break;
            //code 3 is an enemy object, however when drawing the map
            it is treated the same as a ground object as above
        case 3:
            if (infViewMode == 0)
            {
                glColor3f(1, 0, 0);
            }
            else if (infViewMode == 1)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                glColor3f(inf, 0, 0);
            }
            else if (infViewMode == 3)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                inf2 = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
                glColor3f(inf, 0, inf2);
            }
            else if (infViewMode == 4)
            {
                inf = (inMap.tileList[(i*inMap.width) + j].visited
* 2) / 100.0f;
                glColor3f(inf, inf, 0);
            }
            else

```

```

        {
            inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
            glColor3f(inf, 0, 0);
        }
        drawObjAt(Ground, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
        break;
        //code 4 is a goal, we draw a tile object as above and
then draw the goal object on top
        case 4:
            //drawing goal object first
            glColor3f(0, 1, 0);
            drawObjAt(Goal, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
            //then drawing tile object (same as above)
            if (infViewMode == 0)
            {
                glColor3f(1, 0, 0);
            }
            else if (infViewMode == 1)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                glColor3f(inf, 0, 0);
            }
            else if (infViewMode == 3)
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[1] / 100.0f);
                inf2 = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
                glColor3f(inf, 0, inf2);
            }
            else if (infViewMode == 4)
            {
                inf = (inMap.tileList[(i*inMap.width) + j].visited
* 2) / 100.0f;
                glColor3f(inf, inf, 0);
            }
            else
            {
                inf = (inMap.tileList[(i*inMap.width) +
j].infIndex[2] / 100.0f);
                glColor3f(inf, 0, 0);
            }
            drawObjAt(Ground, inMap.tileList[(i*inMap.width) +
j].position.x * 5, 0, inMap.tileList[(i*inMap.width) + j].position.y * 5);
            break;
        }
    }
}

//search for player object in a loaded map to pass appropriate values to an AI
controller
void searchPlayer(Map inMap, AIcontroller* inBot)
{
    for (int i = 0; i < inMap.height; i++)
    {
        for (int j = 0; j < inMap.width; j++)
        {

```

```

        //pass in tile code to switch
        switch (inMap.tileList[(i*inMap.width) + j].code)
        {
        default:
            break;
            //if tile code matches player code, we found the player
        case 2:
            //pass position into the provided controller
            inBot->Pos.x = j;
            inBot->Pos.y = i;
            break;
        }
    }
}

//search for goal object to pass into bot
void searchGoal(Map inMap, AIcontroller* inBot)
{
    for (int i = 0; i < inMap.height; i++)
    {
        for (int j = 0; j < inMap.width; j++)
        {
            switch (inMap.tileList[(i*inMap.width) + j].code)
            {
            default:
                break;
                //if tile code is goal code, goal is found
            case 4:
                //pass goal position to provided bot
                inBot->Goal.x = j;
                inBot->Goal.y = i;
                break;
            }
        }
    }
}

//openGL reshape function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 2000.0);
    //l,r,bottom,top,near,far
    glMatrixMode(GL_MODELVIEW);
}

//calc influence looks at all influence emitters and spreads influence to surrounding
tiles appropriately
void calcInf(Map* inMap)
{
    //two loops to step through every influence on every tile
    for (int i = 0; i < inMap->height * inMap->width; i++)
    {
        for (int j = 0; j < inMap->totInf; j++)
        {
            //pass loop counter into switch to work for each influence
            float newinf = 0;
            switch (j)

```

```

        {
        default:
            break;
        case 0:
            break;
            //for enemy influence...
        case 1:
            //for every enemy in the map...
            for (int k = 0; k < inMap->enemies; k++)
            {
                //work out the distance from the enemy we're
                looking at
                float distance = sqrt(((inMap->tileList[i].position.x - enemies[k].Pos.x)*(inMap->tileList[i].position.x -
enemies[k].Pos.x)) + ((inMap->tileList[i].position.y - enemies[k].Pos.y)*(inMap->tileList[i].position.y - enemies[k].Pos.y)));
                //calculate influence using a formula that lets
                influence drop off drastically after a few tiles, add it to the tiles current
                influence
                //this way the enemy influence accumulates
                depending on how many enemies are near enough to it
                newInf = newInf + ( 100 / (pow(e, distance - 3) +
1));

            }
            //place influence value on this tile
            inMap->tileList[i].infIndex[1] = newInf;
            break;
            //for the goal's influence...
        case 2:
            //work out the tile's distance from the goal
            float distance = sqrt(((inMap->tileList[i].position.x -
inMap->goal.position.x) * (inMap->tileList[i].position.x - inMap->goal.position.x)) +
((inMap->tileList[i].position.y - inMap->goal.position.y) * (inMap->tileList[i].position.y - inMap->goal.position.y)));
            float newInf = 0;
            //if the tile isn't the tile the goal is on...
            if (distance != 0)
            {
                //tile's influence is the reciprocal of (distance +
0.05). Influence is not clamped so it can spread across the whole map
                newInf = 250 * (1 / (distance+0.05));
            }
            else
            {
                //the tile the goal is on is full influence
                newInf = 250;
            }
            //place this influence on this tile
            inMap->tileList[i].infIndex[2] = newInf;
            break;
        }
    }
}

//openGL display function
void display()
{
    //clear background colour, so as last drawn buffer isn't shown
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

```

```

        glLoadIdentity();
        //gluLookAt(camDist*sin(float(theta)*0.0175) + strafe, camDist,
        camDist*cos(float(theta)*0.0175), lookAtX + strafe, lookAtY, lookAtZ, 0.0, 1.0, 0.0);
        gluLookAt(posX, posY, posZ, lookAtX + strafe, lookAtY, lookAtZ, 0.0, 1.0, 0.0);
        //first three - place camera, second three, aim camera at, last three, up
        vector

        //we can add any functions that draw objects here
        //calculate the influence...
        calcInf(&map);
        //draw the map...
        drawMap(map);
        //set colour to draw friend entity
        glColor3f(0, 1, 0);
        //draw player entity
        drawObjAt(Entity, AIPlayer.Pos.x * 5, 0, AIPlayer.Pos.y * 5);
        //set colour for enemy entities
        glColor3f(1, 1, 0);
        //draw all enemies
        for (int i = 0; i < map.enemies; i++)
        {
            drawObjAt(Entity, enemies[i].Pos.x * 5, 0, enemies[i].Pos.y * 5);
        }

        glutSwapBuffers();
    }

    //clear enemy influences so we can spread new influences based on position
    void cleanEnemyInf(Map* inMap)
    {
        //step through each tile...
        for (int i = 0; i < inMap->height * inMap->width; i++)
        {
            //make enemy influence 0...
            inMap->tileList[i].infIndex[1] = 0;
            for (int j = 0; j < inMap->enemies; j++)
            {
                //make every tile an enemy is stood on the full value...
                if (enemies[j].Pos.x == inMap->tileList[i].position.x &&
                    enemies[j].Pos.y == inMap->tileList[i].position.y)
                {
                    inMap->tileList[i].infIndex[1] = 100;
                }
            }
        }
    }

    void main(int argc, char** argv)
    {
        //MAP LOADING
        MapLoader loader;
        //load map
        map = loader.LoadMap("TestMap.txt");
        //set array of enemies to hold the right amount of enemies, default constructor
        //of an AI controller assumes we are creating an enemy AI
        enemies = new AIcontroller[map.enemies];
        //set enemy AI attributes accordingly
        for (int i = 0; i < map.enemies; i++)
        {
            enemies[i].enemy = true;
        }
    }

```

```

        enemies[i].map = &map;
        enemies[i].Pos.x = map.enemyList[i].position.x;
        enemies[i].Pos.y = map.enemyList[i].position.y;
        enemies[i].enemyPause = true;
    }

    //OBJ LOADING
    objNewReader objReader;
    //loading in objects for visual representation
    Ground = objReader.readObjFile("ground.obj");
    Wall = objReader.readObjFile("wall.obj");
    Entity = objReader.readObjFile("entity.obj");
    Goal = objReader.readObjFile("goalNut.obj");

    //search for player and goal in map codes to set up AI correctly
    searchPlayer(map, &AIPlayer);
    searchGoal(map, &AIPlayer);

    //OpenGL Setup
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);
    glutCreateWindow("Influence Maps");
    init();
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboardread);
    glutMainLoop();
}

```

3.1.15. Map.cpp

```

#include "Map.h"
//initlaizes lists and fills them with blank objects ready to be written to
Map::Map(int width, int height, int infCount, int enemyCount)
{
    tileList = new tile[width*height];
    enemyList = new enemy[enemyCount];
    this->width = width;
    this->height = height;
    totInf = 0;
    enemies = enemyCount;
    goal.position.x = 0;
    goal.position.y = 0;
    tile clean;
    clean.code = 'q';
    clean.position.x = 0;
    clean.position.y = 0;
    clean.visited = 0;
    for (int i = 0; i < infCount; i++)
    {
        clean.infIndex[i] = 0;
    }

    for (int j = 0; j < width*height; j++)
    {
        tileList[j] = clean;
    }
}

```

```

Map::~Map()
{
}
3.1.16. MapLoader.cpp

#include "MapLoader.h"

MapLoader::MapLoader()
{
}

MapLoader::~MapLoader()
{
}

Map MapLoader::LoadMap(char* mapLocation)
{
    //THE MAP FILES BEING USED STORE METADATA. THIS LINE IS STORED AS "#DATA"
    FOLLOWED BY THREE VALUES - HEIGHT WIDTH AND THE NUMBER OF INFLUENCE EMITTER TYPES
    char buffer[100];
    FILE * mapFile;
    int width;
    int height;
    int infCount;
    int enemyCount;

    //open file
    mapFile = fopen(mapLocation, "r");
    //scan in first line
    fscanf(mapFile, "%s", buffer);

    //loop till eof
    //seems a lil wasteful to have to scan the file twice for just the metadata-
    FIX THIS IF POSSIBLE AND TIME ALLOWS
    while (!feof(mapFile))
    {
        if (buffer != "")
        {
            if (strcmp(buffer, "#data") == 0)
            {
                //we're in metadata, we know what the next three scans
                should be
                fscanf(mapFile, "%d %d %d %d", &height, &width, &infCount,
                &enemyCount);
            }
            fscanf(mapFile, "%s", buffer);
        }

        //init Map
        Map outMap(width, height, infCount, enemyCount);
        //return to start of file
        rewind(mapFile);
        int wCounter = 0;
        int hCounter = 0;
        outMap.totInf = infCount;
        fscanf(mapFile, "%s", buffer);
        int enemyCounter = 0;
        while (!feof(mapFile))
        {

```

```

        if (buffer != "")
        {
            //a '~' signifies a line follows that is a map line, otherwise we
            ignore it...
            if (strcmp(buffer, "~") == 0)
            {
                //for each character in the line read in, check character
                and set map's tile code accordingly...
                fscanf(mapFile, "%s", buffer);
                for (int i = 0; i < width; i++)
                {
                    switch (buffer[i])
                    {
                        //character x is a wall
                        case 'x':
                            outMap.tileList[(hCounter * width) +
                                wCounter].code = 0;
                            outMap.tileList[(hCounter * width) +
                                wCounter].infIndex[0] = 1000;
                            break;
                        //character . is a floor tile
                        case '.':
                            outMap.tileList[(hCounter * width) +
                                wCounter].code = 1;
                            break;
                        //character p is a player
                        case 'p':
                            outMap.tileList[(hCounter * width) +
                                wCounter].code = 2;
                            break;
                        //character e is an enemy
                        case 'e':
                            outMap.tileList[(hCounter * width) +
                                wCounter].code = 3;
                            outMap.tileList[(hCounter * width) +
                                wCounter].infIndex[1] = 100;
                            wCounter;
                            hCounter;
                            outMap.enemyList[enemyCounter].position.x =
                                outMap.enemyList[enemyCounter].position.y =
                                    enemyCounter++;
                            break;
                        //character g is a goal
                        case 'g':
                            outMap.tileList[(hCounter * width) +
                                wCounter].code = 4;
                            outMap.tileList[(hCounter * width) +
                                wCounter].infIndex[2] = 250;
                            outMap.goal.position.x = wCounter;
                            outMap.goal.position.y = hCounter;
                            break;
                    }
                    //place tile's position correctly
                    outMap.tileList[(hCounter * width) +
                        wCounter].position.x = wCounter;
                    outMap.tileList[(hCounter * width) +
                        wCounter].position.y = hCounter;
                    wCounter++;
                }
                hCounter++;
                wCounter = 0;
            }
        }
    }
}

```

```

        }
        fscanf(mapFile, "%s", buffer);
    }

    return outMap;
}

3.1.17. MathLib.cpp

#include "MathLib.h"

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

////
//// MATHLIB WRITTEN FOR PREVIOUS ASSIGNMENT IN SECOND YEAR. REUSED AS IS WITH NO
CHANGES FROM THERE. ALL WORK MINE.
////
////

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
MathLib::MathLib()
{
    for (int i = 0; i < 1024; i++)
    {
        float scaledDeg = (360.0f / 1024.0f)*(static_cast<float>(i) + 1.0f);
        float scaledRad = MathLib::conversionAngRad(scaledDeg);
        lupSin[i] = sin(scaledRad);
        lupCos[i] = cos(scaledRad);
        //printf("COS %d: %f\n", i, lupCos[i]);
        //printf("SIN %d: %f\n", i, lupSin[i]);
    }
}

MathLib::~MathLib()
{
}

float MathLib::conversionAngRad(float degrees)
{
    float rad = degrees*(PI / 180);
    return rad;
}

float MathLib::conversionAngDeg(float Radians)
{
    float deg = Radians*(180 / PI);
    return deg;
}

float MathLib::getCos(float angleInDegrees)
{
    int slot = static_cast<int>((angleInDegrees * 1024) / 360);
    return lupCos[slot];
}

float MathLib::getSin(float angleInDegrees)
{
    int slot = static_cast<int>((angleInDegrees * 1024) / 360);
    return lupSin[slot];
}

```

```

}

Vector MathLib::multiplyMatVect(Matrix inMat, Vector inVect)
{
    Vector outVect(0,0,0);

    outVect.vect[0] = (inMat.mtrx[0][0] * inVect.vect[0]) + (inMat.mtrx[0][1] *
inVect.vect[1]) + (inMat.mtrx[0][2] * inVect.vect[2]) + (inMat.mtrx[0][3] *
inVect.vect[3]);
    outVect.vect[1] = (inMat.mtrx[1][0] * inVect.vect[0]) + (inMat.mtrx[1][1] *
inVect.vect[1]) + (inMat.mtrx[1][2] * inVect.vect[2]) + (inMat.mtrx[1][3] *
inVect.vect[3]);
    outVect.vect[2] = (inMat.mtrx[2][0] * inVect.vect[0]) + (inMat.mtrx[2][1] *
inVect.vect[1]) + (inMat.mtrx[2][2] * inVect.vect[2]) + (inMat.mtrx[2][3] *
inVect.vect[3]);
    outVect.vect[3] = (inMat.mtrx[3][0] * inVect.vect[0]) + (inMat.mtrx[3][1] *
inVect.vect[1]) + (inMat.mtrx[3][2] * inVect.vect[2]) + (inMat.mtrx[3][3] *
inVect.vect[3]);

    return outVect;
}

void MathLib::setIDMat(Matrix* inMat)
{
    for (int i = 0; i < 4; i++)
    {
        inMat->mtrx[i][i] = 1;
    }
}

void MathLib::setTransMat(Matrix* inMat, float x, float y, float z)
{
    inMat->mtrx[0][3] = x;
    inMat->mtrx[1][3] = y;
    inMat->mtrx[2][3] = z;
}

void MathLib::setRotXMat(Matrix* inMat, float angle)
{
    inMat->mtrx[1][1] = getCos(angle);
    inMat->mtrx[2][1] = getSin(angle);
    inMat->mtrx[1][2] = -(getSin(angle));
    inMat->mtrx[2][2] = getCos(angle);
}

void MathLib::setRotYMat(Matrix * inMat, float angle)
{
    inMat->mtrx[0][0] = getCos(angle);
    inMat->mtrx[2][0] = -(getSin(angle));
    inMat->mtrx[0][2] = getSin(angle);
    inMat->mtrx[2][2] = getCos(angle);
}

void MathLib::setRotZMat(Matrix* inMat, float angle)
{
    inMat->mtrx[0][0] = getCos(angle);
    inMat->mtrx[1][0] = getSin(angle);
    inMat->mtrx[0][1] = -(getSin(angle));
    inMat->mtrx[1][1] = getCos(angle);
}

```

```

void MathLib::setUniSclMat(Matrix* inMat, float scale)
{
    inMat->mtx[3][3] = scale * 0.5;
}

void MathLib::setSclMat(Matrix* inMat, float scaleX, float scaleY, float scaleZ)
{
    inMat->mtx[0][0] = scaleX;
    inMat->mtx[1][1] = scaleY;
    inMat->mtx[2][2] = scaleZ;
}

Matrix MathLib::multiplyMat(Matrix m1, Matrix m2)
{
    Matrix outMat;
    float currentTotal;
    currentTotal = (m1.mtx[0][0] * m2.mtx[0][0]) + (m1.mtx[0][1] *
m2.mtx[1][0]) + (m1.mtx[0][2] * m2.mtx[2][0]) + (m1.mtx[0][3] * m2.mtx[3][0]);
    outMat.mtx[0][0] = currentTotal;
    currentTotal = (m1.mtx[0][0] * m2.mtx[0][1]) + (m1.mtx[0][1] *
m2.mtx[1][1]) + (m1.mtx[0][2] * m2.mtx[2][1]) + (m1.mtx[0][3] * m2.mtx[3][1]);
    outMat.mtx[0][1] = currentTotal;
    currentTotal = (m1.mtx[0][0] * m2.mtx[0][2]) + (m1.mtx[0][1] *
m2.mtx[1][2]) + (m1.mtx[0][2] * m2.mtx[2][2]) + (m1.mtx[0][3] * m2.mtx[3][2]);
    outMat.mtx[0][2] = currentTotal;
    currentTotal = (m1.mtx[0][0] * m2.mtx[0][3]) + (m1.mtx[0][1] *
m2.mtx[1][3]) + (m1.mtx[0][2] * m2.mtx[2][3]) + (m1.mtx[0][3] * m2.mtx[3][3]);
    outMat.mtx[0][3] = currentTotal;

    currentTotal = (m1.mtx[1][0] * m2.mtx[0][0]) + (m1.mtx[1][1] *
m2.mtx[1][0]) + (m1.mtx[1][2] * m2.mtx[2][0]) + (m1.mtx[1][3] * m2.mtx[3][0]);
    outMat.mtx[1][0] = currentTotal;
    currentTotal = (m1.mtx[1][0] * m2.mtx[0][1]) + (m1.mtx[1][1] *
m2.mtx[1][1]) + (m1.mtx[1][2] * m2.mtx[2][1]) + (m1.mtx[1][3] * m2.mtx[3][1]);
    outMat.mtx[1][1] = currentTotal;
    currentTotal = (m1.mtx[1][0] * m2.mtx[0][2]) + (m1.mtx[1][1] *
m2.mtx[1][2]) + (m1.mtx[1][2] * m2.mtx[2][2]) + (m1.mtx[1][3] * m2.mtx[3][2]);
    outMat.mtx[1][2] = currentTotal;
    currentTotal = (m1.mtx[1][0] * m2.mtx[0][3]) + (m1.mtx[1][1] *
m2.mtx[1][3]) + (m1.mtx[1][2] * m2.mtx[2][3]) + (m1.mtx[1][3] * m2.mtx[3][3]);
    outMat.mtx[1][3] = currentTotal;

    currentTotal = (m1.mtx[2][0] * m2.mtx[0][0]) + (m1.mtx[2][1] *
m2.mtx[1][0]) + (m1.mtx[2][2] * m2.mtx[2][0]) + (m1.mtx[2][3] * m2.mtx[3][0]);
    outMat.mtx[2][0] = currentTotal;
    currentTotal = (m1.mtx[2][0] * m2.mtx[0][1]) + (m1.mtx[2][1] *
m2.mtx[1][1]) + (m1.mtx[2][2] * m2.mtx[2][1]) + (m1.mtx[2][3] * m2.mtx[3][1]);
    outMat.mtx[2][1] = currentTotal;
    currentTotal = (m1.mtx[2][0] * m2.mtx[0][2]) + (m1.mtx[2][1] *
m2.mtx[1][2]) + (m1.mtx[2][2] * m2.mtx[2][2]) + (m1.mtx[2][3] * m2.mtx[3][2]);
    outMat.mtx[2][2] = currentTotal;
    currentTotal = (m1.mtx[2][0] * m2.mtx[0][3]) + (m1.mtx[2][1] *
m2.mtx[1][3]) + (m1.mtx[2][2] * m2.mtx[2][3]) + (m1.mtx[2][3] * m2.mtx[3][3]);
    outMat.mtx[2][3] = currentTotal;

    currentTotal = (m1.mtx[3][0] * m2.mtx[0][0]) + (m1.mtx[3][1] *
m2.mtx[1][0]) + (m1.mtx[3][2] * m2.mtx[2][0]) + (m1.mtx[3][3] * m2.mtx[3][0]);
    outMat.mtx[3][0] = currentTotal;
}

```

```

        currentTotal = (m1.mtrx[3][0] * m2.mtrx[0][1]) + (m1.mtrx[3][1] *
m2.mtrx[1][1]) + (m1.mtrx[3][2] * m2.mtrx[2][1]) + (m1.mtrx[3][3] * m2.mtrx[3][1]);
        outMat.mtrx[3][1] = currentTotal;
        currentTotal = (m1.mtrx[3][0] * m2.mtrx[0][2]) + (m1.mtrx[3][1] *
m2.mtrx[1][2]) + (m1.mtrx[3][2] * m2.mtrx[2][2]) + (m1.mtrx[3][3] * m2.mtrx[3][2]);
        outMat.mtrx[3][2] = currentTotal;
        currentTotal = (m1.mtrx[3][0] * m2.mtrx[0][3]) + (m1.mtrx[3][1] *
m2.mtrx[1][3]) + (m1.mtrx[3][2] * m2.mtrx[2][3]) + (m1.mtrx[3][3] * m2.mtrx[3][3]);
        outMat.mtrx[3][3] = currentTotal;
        return outMat;
    }

Matrix MathLib::addMat(Matrix m1, Matrix m2)
{
    Matrix outMat;
    outMat.mtrx[0][0] = m1.mtrx[0][0] + m2.mtrx[0][0];
    outMat.mtrx[0][1] = m1.mtrx[0][1] + m2.mtrx[0][1];
    outMat.mtrx[0][2] = m1.mtrx[0][2] + m2.mtrx[0][2];
    outMat.mtrx[0][3] = m1.mtrx[0][3] + m2.mtrx[0][3];

    outMat.mtrx[1][0] = m1.mtrx[1][0] + m2.mtrx[1][0];
    outMat.mtrx[1][1] = m1.mtrx[1][1] + m2.mtrx[1][1];
    outMat.mtrx[1][2] = m1.mtrx[1][2] + m2.mtrx[1][2];
    outMat.mtrx[1][3] = m1.mtrx[1][3] + m2.mtrx[1][3];

    outMat.mtrx[2][0] = m1.mtrx[2][0] + m2.mtrx[2][0];
    outMat.mtrx[2][1] = m1.mtrx[2][1] + m2.mtrx[2][1];
    outMat.mtrx[2][2] = m1.mtrx[2][2] + m2.mtrx[2][2];
    outMat.mtrx[2][3] = m1.mtrx[2][3] + m2.mtrx[2][3];

    outMat.mtrx[3][0] = m1.mtrx[3][0] + m2.mtrx[3][0];
    outMat.mtrx[3][1] = m1.mtrx[3][1] + m2.mtrx[3][1];
    outMat.mtrx[3][2] = m1.mtrx[3][2] + m2.mtrx[3][2];
    outMat.mtrx[3][3] = m1.mtrx[3][3] + m2.mtrx[3][3];
    return outMat;
}

Matrix MathLib::transposeMat(Matrix inputMat)
{
    Matrix outMat;
    outMat.mtrx[0][0] = inputMat.mtrx[0][0];
    outMat.mtrx[0][1] = inputMat.mtrx[1][0];
    outMat.mtrx[0][2] = inputMat.mtrx[2][0];
    outMat.mtrx[0][3] = inputMat.mtrx[3][0];

    outMat.mtrx[1][0] = inputMat.mtrx[0][1];
    outMat.mtrx[1][1] = inputMat.mtrx[1][1];
    outMat.mtrx[1][2] = inputMat.mtrx[2][1];
    outMat.mtrx[1][3] = inputMat.mtrx[3][1];

    outMat.mtrx[2][0] = inputMat.mtrx[0][2];
    outMat.mtrx[2][1] = inputMat.mtrx[1][2];
    outMat.mtrx[2][2] = inputMat.mtrx[2][2];
    outMat.mtrx[2][3] = inputMat.mtrx[3][2];

    outMat.mtrx[3][0] = inputMat.mtrx[0][3];
    outMat.mtrx[3][1] = inputMat.mtrx[1][3];
    outMat.mtrx[3][2] = inputMat.mtrx[2][3];
    outMat.mtrx[3][3] = inputMat.mtrx[3][3];

    return outMat;
}

```

```

bool MathLib::checkOrth(Matrix inputMat)
{
    Matrix transMat = MathLib::transposeMat(inputMat);
    Matrix orthCheck = MathLib::multiplyMat(inputMat, transMat);
    Matrix idCheck;
    MathLib::setIDMat(&idCheck);
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (orthCheck.mtrx[i][j] != idCheck.mtrx[i][j])
            {
                return false;
            }
        }
    }
    return true;
}

float MathLib::dotProd(Vector v1, Vector v2)
{
    float dotProd = (v1.vect[0] * v2.vect[0]) + (v1.vect[1] * v2.vect[1]) +
(v1.vect[2] * v2.vect[2]) + (v1.vect[3] * v2.vect[3]);
    return dotProd;
}

float MathLib::vectorLength(Vector inVect)
{
    float length;
    length = (inVect.vect[0] * inVect.vect[0]) + (inVect.vect[1] * inVect.vect[1])
+ (inVect.vect[2] * inVect.vect[2]) + (inVect.vect[3] * inVect.vect[3]);
    return sqrt(length);
}

```

3.1.18. Matrix.cpp

```

////////////////////////////////////
////////////////////////////////////
////

////
////  MATRIX WRITTEN FOR PREVIOUS ASSIGNMENT IN SECOND YEAR. REUSED AS IS WITH NO
CHANGES FROM THERE. ALL WORK MINE.  ////
////

////
////////////////////////////////////
////////////////////////////////////
#include "Matrix.h"

Matrix::Matrix()
{
}

Matrix::~Matrix()
{
}

```

3.1.19. Obj.cpp

```
#include "obj.h"

obj::obj(int inVSize, int inVtSize, int inVnSize, int inFSize)
{
    //set sizes
    vSize = inVSize;
    vtSize = inVtSize;
    vnSize = inVnSize;
    fSize = inFSize;

    //set up lists
    vList = new Vertex[vSize];
    vtList = new vTexture[vtSize];
    vnList = new vNormal[vnSize];
    fList = new Face[fSize];

    //filling vertex list with empties
    for (int i = 0; i < vSize; i++)
    {
        Vertex clean;
        clean.x = 0.0f;
        clean.y = 0.0f;
        clean.z = 0.0f;
        vList[i] = clean;
    }
}

obj::~obj()
{
}
```

3.1.20. objNewReader.cpp

```
#include "objNewReader.h"
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

////
////  OBJREADER WRITTEN AS AN ALL PURPOSE LIBRARY, USED IN MULTIPLE THIRD YEAR
PROJECTS. ALL WORK MINE.      ////
////

////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
objNewReader::objNewReader()
{
}

objNewReader::~objNewReader()
{
}

obj objNewReader::readObjFile(char * inFile)
{
    //buffer for reading
    char buffer[50];
```

```

FILE * objFile;
int vCount = 0;
int vtCount = 0;
int vnCount = 0;
int fCount = 0;
//unsure if used
int faceCornerCount;
bool inFace = false;
bool didRead = false;

//open file
objFile = fopen(inFile, "r");
//scan in first line
fscanf(objFile, "%s", buffer);

//looping while not at EOF
while (!feof(objFile))
{
    //if buffer isn't whitespace, move on
    if (buffer != "")
    {
        //if buffer matches certain keys, take action
        if (strcmp(buffer, "v") == 0)
        {
            vCount++;
        }
        else if (strcmp(buffer, "vt") == 0)
        {
            vtCount++;
        }
        else if (strcmp(buffer, "vn") == 0)
        {
            vnCount++;
        }
        else if (strcmp(buffer, "f") == 0)
        {
            fCount++;
        }
        //here we would also check for mtls, but we're only fussed with
        mesh data right now
    }
    fscanf(objFile, "%s", buffer);
}

//internal obj to hold data and output when we're done
obj object(vCount, vtCount, vnCount, fCount);
//would also have read in mtlfile name from file in first pass and open it here

//return to start of objfile
rewind(objFile);

//we're tracking data being put in to make sure it matches what was first
counted here
//also tracks which slot in the object array to dump data in
int vIn = 0;
int vtIn = 0;
int vnIn = 0;
int fIn = 0;

//here we would usually read in all the mtls and assign an ID to them. No need
this time
//read ahead in objfile

```

```

fscanf(objFile, "%s", buffer);
//while not at EOF again
while (!feof(objFile))
{
    //if string matches, copy data over
    //Wanted to use a case switch instead for this in previous version??
    Check this
    if (strcmp(buffer, "v") == 0)
    {
        //if we see a v we know the next three are point data
        fscanf(objFile, "%f %f %f", &object.vList[vIn].x,
        &object.vList[vIn].y, &object.vList[vIn].z);
        //increase count
        vIn++;
    }
    else if (strcmp(buffer, "vt") == 0)
    {
        fscanf(objFile, "%f %f", &object.vtList[vtIn].x,
        &object.vtList[vtIn].y);
        vtIn++;
    }
    else if (strcmp(buffer, "vn") == 0)
    {
        fscanf(objFile, "%f %f %f", &object.vnList[vnIn].x,
        &object.vnList[vnIn].y, &object.vnList[vnIn].z);
        vnIn++;
    }
    //a check for usemtl exists in previous version - no need here
    else if (strcmp(buffer, "f") == 0)
    {
        //HEY! WOULD IT BE TOO MUCH TO DECOUPLE THIS?
        //previous version worked with quads - working with tris here
        //previous versions assigned mtlindex to faces - only concerned
        with mesh for now, not mtl

        //face data is formatted as f int/int/int int/int/int int/int/int
        //each set of three is v/vt/vn, refers to each vertex of a tri
        //following the f tag in file, we know three sets occur
        //so we read in each set, split that set, copy data

        fscanf(objFile, "%s", buffer);
        //strtok needs to output data to somewhere, so we make that
        somewhere
        char * split = strtok(buffer, "/");
        //then copy this!
        //gotta convert the data to int tho, and drop by one to count
        from 0. yay for zeroes.
        object.fList[fIn].v1 = atoi(split) - 1;
        //strtok remembers the set it's splitting - we use NULL to let it
        carry on through the data
        //overwriting split for each new strtok operation since we've
        gotten the data we need from it
        split = strtok(NULL, "/");
        object.fList[fIn].vn1 = atoi(split) - 1;
        split = strtok(NULL, "/");
        object.fList[fIn].vt1 = atoi(split) - 1;

        //and now we do it all again for sets 2 and 3
        fscanf(objFile, "%s", buffer);
        split = strtok(buffer, "/");
        object.fList[fIn].v2 = atoi(split) - 1;
        split = strtok(NULL, "/");

```

```

        object.fList[fIn].vn2 = atoi(split) - 1;
        split = strtok(NULL, "/");
        object.fList[fIn].vt2 = atoi(split) - 1;

        fscanf(objFile, "%s", buffer);
        split = strtok(buffer, "/");
        object.fList[fIn].v3 = atoi(split) - 1;
        split = strtok(NULL, "/");
        object.fList[fIn].vn3 = atoi(split) - 1;
        split = strtok(NULL, "/");
        object.fList[fIn].vt3 = atoi(split) - 1;

        //put that count up
        fIn++;
        //here previous version basically checked if an object was using
quads and triangulated it into a new face
        //project isn't as fussed about that right now. Not bothering
    }
    fscanf(objFile, "%s", buffer);
}
//close that file
fclose(objFile);
//get that object out into the world!
//i.e. return object
return object;
//HEY. YOU'VE COUNTED HOW MANY DATA YOU COPIED. OUGHT TO ERROR CHECK IT BEFORE
OUTPUT
}

```

3.1.21. Vector.cpp

```

#include "Vector.h"
////////////////////////////////////
////////////////////////////////////
////

////
//// VECTOR WRITTEN FOR PREVIOUS ASSIGNMENT IN SECOND YEAR. REUSED AS IS WITH NO
CHANGES FROM THERE. ALL WORK MINE.
////

////
////////////////////////////////////
////////////////////////////////////
Vector::Vector(float x, float y, float z)
{
    vect[0] = x;
    vect[1] = y;
    vect[2] = z;
    vect[3] = 1.0f;
}

Vector::~Vector()
{
}

```

4. References

- [1] I. Millington, *Artificial Intelligence for Games*, Burlington: Morgan Kaufmann Publishers, 2009.
- [2] "Gaussian Kernel Calculator | The Devil in the Details," 30 January 2014. [Online]. Available: <http://dev.theomader.com/gaussian-kernel-calculator/>. [Accessed 20 March 2019].
- [3] "A Beginners Guide to Big O Notation," 23 June 2009. [Online]. Available: <https://robbell.net/2009/06/a-beginners-guide-to-big-o-notation/>. [Accessed 20 March 2019].
- [4] J. Skorin-Kapov, "Tabu Search Applied To The Quadratic Assignment Problem," *ORSA J. Comput.*, vol. 2, no. 1, pp. 33-45, 1990.
- [5] E. Taillard, "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, vol. 17, pp. 443-455, 1991.

FINAL GRADE

GENERAL COMMENTS

Instructor

82/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

Convolution kernels can also sharpen images and more.

PAGE 4



Comment 2

good. Correct

PAGE 5

PAGE 6



Comment 3

Correct. Well done

PAGE 7



Comment 4

Good

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

GRADING FORM: SMU GENERIC_GJ

OLIVER JORDAN

82

RESEARCH-INFORMED LITERATURE

 7.5/10 References generally used appropriately and include standard texts. Could be more varied. Tabu search ref - why not use a std text book here. One or two formatting/style issues.

7.5

KNOWLEDGE AND UNDERSTANDING

 19/25 Good understanding of methods used to calculate influence. Discussion of complexity applied to algorithms, though the descriptions could be a little clearer in places.

19

ANALYSIS

 16/20 Complexity of algorithms reviewed and calculations correctly applied.

16

PRACTICAL APPLICATION AND DEP.

 36/40. Some good work here. Graphical interface well developed. Objects contribute to map. Influence map file loaded. State machine used to control behaviours- very well done! Sophisticated AI behaviours demonstrated.

36

SKILLS FOR PROFESSIONAL PRAC.

 3.5/10 Well structured report. Refs appropriate.

3.5