Assignment No 1

```
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */

using namespace std;

struct agent {
        int type;
        int xpos;
        int ypos;
        int direction;
        int score;
};

class world {
private:
        int height;     //The world is a 2D grid - hence the height and width
        int width;
public:
        int contents[4][4];     //make a 2D array to hold dirt

// Pick a random spot on the grid and fill it with dirt.
        inline void create_dirt () {
                contents[rand()%4][rand()%4] = 1;
        }

// Initializes the grid.  Sets height and weight to 4 and makes sure
// the grid is empty.
```

```
    void initialize() {
        height = 4;
        width = 4;
        for (int i=0;i<height;i++) {
            for (int j=0;j<width;j++) {
                contents[i][j] = 0;
            }
        }
    }
void display() {
        height = 4;
        width = 4;
        for (int i=0;i<height;i++) {
            for (int j=0;j<width;j++) {
                cout<<contents[i][j];
            }
                                    cout<<endl;
        }
    }
// This brings up a display of the grid and reports the agent's location
    void report_dims (agent x) {
        for (int i=0;i<height;i++) {
            for (int j=0;j<width;j++) {
                cout << contents[j][i];
            }
            cout << "\n";
        }
        cout << "Agent is currently in (" << x.xpos << ", " << x.ypos << ") and facing "
            << x.direction << " with score " << x.score << "\n";
        }

// Agent functions.  Report state of current square, turn left or
// right, move forward
        inline int report (agent x) {
                return (contents[x.ypos][x.xpos]);
        }
        inline void turn_right (agent x) {
                x.direction++;
                if (x.direction > 3)
                        x.direction = 0;
        }
        inline void turn_left (agent x) {
                x.direction--;
                if (x.direction < 0)
                        x.direction = 3;
        }
        void move_forward (agent x) {
                switch (x.direction) {
                case 0:
                        x.ypos++;
                case 1:
                        x.xpos++;
                case 2:
                        x.ypos--;
                case 3:
                        x.xpos--;
                }
        }
```

```
// Suck up dirt in the case of a vacuum, though could apply to
// anything eating anything else later
        inline void eat_here (agent x) {
//      This is where I deal with scoring based on the agent's type
//              switch (x.type) {
//              case 2:
//                      if (contents[x.ypos][x.xpos] = 1)
//                              x.score += 100;
//              }
                contents[x.ypos][x.xpos] = 0;
        }
};

//--A single move for a random agent--//
void run_rand_vac(agent x, world y) {
        switch (rand() % 5) {
        case 0:
                y.move_forward(x);
                cout << "Moving forward (1).\n";
        case 1:
                y.turn_left(x);
                cout << "Turning left.\n";
        case 2:
                y.turn_right(x);
                cout << "Turning right.\n";
        case 3:
                y.eat_here(x);
                cout << "Eating dirt. \n";
```

```
    case 4:
        y.move_forward(x);
        cout << "Moving forward (4).\n";
    }
}

//--A single move for an intelligent agent--// If report returns
//something other than 0, that means there's something to eat.  If
//there's nothing to eat, make a random move.
void run_smart_vac(agent x, world y) {
    if (y.report(x))
        y.eat_here(x); else
            switch (rand() % 5) {
            case 0:
                y.move_forward(x);
                cout << "Moving forward (0).\n";
            case 1:
                y.move_forward(x);
                cout << "Moving forward (1).\n";
            case 2:
                y.move_forward(x);
                cout << "Moving forward (2).\n";
            case 3:
                y.turn_left(x);
                cout << "Turning left.\n";
            case 4:
                y.turn_right(x);
                cout << "Turning right.\n";
            }
```

```
}
int main() {
 // int i;   // -jim added this, to fix variable i scoping


    world grid;                    //create a world named 'grid'
    grid.initialize();             //initialize and empty 'grid'
    agent randvac = {2, 0, 0, 2, 0};     //create a cleaner at
                          //(0,0) facing south
    for (int i=0;i<10;i++) {
     grid.create_dirt();            //for loop dumps out
                          //ten spots of dirt
                          //at random (allows
                          //overlap)
    }
    grid.report_dims(randvac);       //report the 'before' state of
                          //the grid
    for (int i=0;i<10;i++) {
        run_rand_vac(randvac, grid);    //run the cleaner 50 moves
    grid.display();          //report the 'after'
    }
    grid.report_dims(randvac);       //report the 'after'
                          //state of the grid
    return 0;
}
```