

**Object Orientated C++ (MOD003197)**

Element 010: Documentation Report

**SID: 1839616**

**Due Date: Turnitin, 2pm, 4th May 2021**

**Faculty: Science and Technology**

**Department: Computing & Technology**

**Module Code: MOD003197**

**Academic Year: 2020/21**

**Trimester: 2**

**Contents**

1. Introduction………………………………………………………………………. 3

2. Implementation and Evaluation………………………………………………... 3

3. Class Diagrams………......……………………………………………………... 4

4. Appendix………………………………………………………………………...... 6

1. **Introduction**

In this documentation report, I will show the approach I have taken in designing and implementing a program that models the spread of fire in a 2D forest environment and will also show the strengths and limitations while designing the program.

1. **Implementation and Evaluation**

The program built executes cleanly and follows all the timestep rules with no run-time errors. The program also includes enhancements such as weather conditions (Normal or windy), Ground moisture; East of the forest is initialised to be damper side and hence harder to burn. Wind speed is measured in miles per hour (mph) that ranges between 10-30 mph with 10-17 mph being normal/low and 17-30 mph being high speed, wind speed determines how long the simulation is to run and wind direction is North, South, East, and West. New tree growth, where after a tree is burnt, a new tree grows with a probability of 50%.

The program consists of 2 classes and the main function. A Cell class that consists of cell objects that fill up a 2D array called forest with attributes such as cell state, which is set to be 1 (tree) except for the centre burning cell, which is initiated to 2 (Burning tree) and a getStateValue() method which returns the current state of the cell, and a setStateValue(). A Grid class that consists of various methods such as Grid(), spread(), applySpread(), IsAnyCellOnFire(), DrawGrid(), Regrowth(), and Info(). Lastly, a main() function is the entry point of the program.

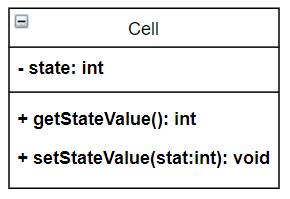
The program starts by first creating a 2D array of cell objects which are represented on a 2D array of chars for display. The cell in the centre of the grid is initialised to be 2 (burning tree) marked with an "x". The probability of any cell (tree) to burn depends on parameters such as weather, windDirection, windspeed, newCellGrowth, and groundMoisture. Only the groundMoisture is kept constant throughout the simulation, and other parameters are selected at random at run-time. Example scenario: if the weather is normal, then all the cells have a probability of burning 50%. Example scenario 2: If the weather is windy, and if the direction of the wind is east which is also the area of the forest that is damp, the probability of burning is 40%. If the weather is windy, and the direction of the wind is anything other east, then the location of the cell is checked, and if the location of the cell is in the direction of the wind, then the probability of burning of cells in that direction is 80% and cells that are not in the direction of the wind have a probability of burning of 50%.

* **Limitations**

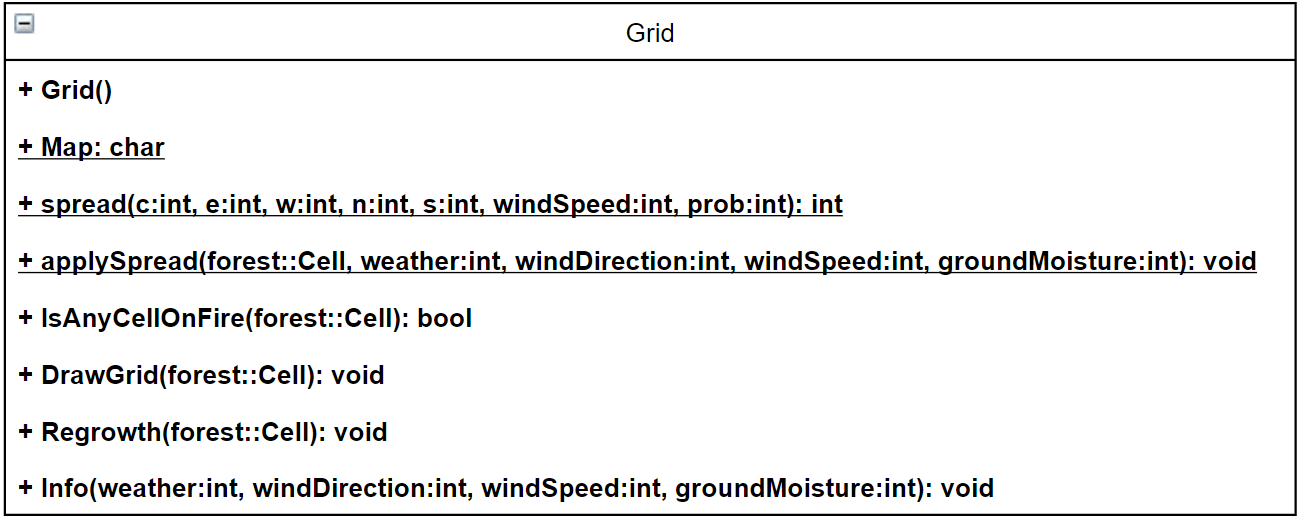
It is assumed that the simulation finishes when no cell is on fire. The simulation **may** end as soon as the first timestep. This is because we start reading the array of objects cells from the east towards the west and north towards the south. When a cell is read, and if it is a burning tree, it will turn into an empty site before we update its immediate west and south neighbours. So ideally, the centre burning cell is only exposed to an east and north neighbour. The west and south neighbours see an empty site instead of a burning tree because it's updated before them, and hence this causes the simulation to end. I can say this is the limitation of having an array.

1. **Class Diagrams**

The diagram for the “Cell” class:



The diagram for “Grid” class:



1. **Appendix**

* **Instructions**

As described in the report about the limitation, this can cause the simulation to end as soon as the first timestep so please do run the simulation a couple times to see the desired output.

* **Listing of Source Code**

**.h files**

* **Cell.h**

#ifndef CELL\_H\_INCLUDED

#define CELL\_H\_INCLUDED

class Cell

{

int state; // state of the cell; 0: Empty, 1: Tree, 2: Burning tree

public:

Cell();

int getStateValue();

void setStateValue(int stat);

};

#endif // CELL\_H\_INCLUDED

* **Grid.h**

#ifndef GRID\_H\_INCLUDED

#define GRID\_H\_INCLUDED

class Grid

{

public:

Grid();

static char Map[21][21];

static int spread(int c, int e , int w , int n , int s, int windSpeed, int prob); //take all current states and return new state

static void applySpread(Cell forest[21][21], int weather, int windDirection, int windSpeed, int groundMoisture);

bool IsAnyCellOnFire(Cell forest[21][21]);

void DrawGrid(Cell forest[21][21]);

void Regrowth(Cell forest[21][21]);

void Info(int weather, int windDirection, int windSpeed, int groundMoisture);

};

#endif // GRID\_H\_INCLUDED

* **.cpp files**
* **Cell.cpp**

#include<iostream>

#include "Cell.h"

using namespace std;

Cell::Cell()

{

state = 1;

}

int Cell::getStateValue()

{

return state;

}

void Cell::setStateValue(int stat)

{

state = stat;

}

* **Grid.cpp**

#include<iostream>

#include "Cell.h"

#include "Grid.h"

using namespace std;

char Grid::Map[21][21];

Grid::Grid()

{

for(int i = 0 ; i < 21; i++)

{

for(int j = 0; j < 21; j++)

{

Map[i][j] = '&';

}

}

}

int Grid::spread(int c, int e , int w , int n , int s, int windSpeed, int probability ) //take all current states and return new state

{

if(c == 0 || c == 2)

return 0;

if(c == 1)

{

if(e != 2 && w != 2 && n != 2 && s != 2)

return 1;

else

{

int seconds = 0;

if(windSpeed < 17)

seconds = 1000;

else

seconds = 500;

//Sleep(seconds);

//srand( (unsigned)time( NULL ) );

int m = 2;

if(probability == 40 || probability == 80)

m = 3;

else if(probability == 50)

m = 2;

//choose random number between 0 & 1, 0 for a tree and 1 for a burning tree

int random = rand() % m;

if(probability == 40 && (random == 0 || random == 2))

return 1;

else if(probability == 40 && random == 1)

return 2;

else if(probability == 80 && (random == 1 || random == 2))

return 2;

else if(probability == 80 && random == 0)

return 1;

else if(probability == 50 && random == 0)

return 1;

else if(probability == 50 && random == 1)

return 2;

}

}

return 9;

}

void Grid::applySpread(Cell forest[21][21], int weather, int windDirection, int windSpeed, int groundMoisture)

{

int newState, startRow,startCol,endRow,endCol,probability = 50;

for(int i = 0 ; i < 21; i++)

{

for(int j = 0; j < 21; j++)

{

if(weather == 0)

probability = 50;

else

{

switch(windDirection)

{

case 0 : //east

if(groundMoisture == windDirection)

probability = 40;

else if(j >= 0 && j <= 10)

probability = 80;

else

probability = 40;

break;

case 1 : //west

if(groundMoisture == windDirection)

probability = 40;

else if(j >= 10 && j < 21)

probability = 80;

else

probability = 40;

break;

case 2 : //south

if(groundMoisture == windDirection)

probability = 40;

else if(i >= 10 && i < 21)

probability = 80;

else

probability = 40;

break;

case 3 : //north

if(groundMoisture == windDirection)

probability = 40;

else if(i >= 0 && i <= 10)

probability = 80;

else

probability = 40;

break;

}

}

int east, west, south, north;

//checking for the trees at the edges

if(i == 0)

north = 0;

else

north = forest[i-1][j].getStateValue();

if(j == 0)

east = 0;

else

east = forest[i][j-1].getStateValue();

if (i == 20)

south = 0;

else

south = forest[i+1][j].getStateValue();

if (j == 20)

west = 0;

else

west = forest[i][j+1].getStateValue();

newState = spread(forest[i][j].getStateValue(), east, west, south, north, windSpeed, probability);

//apply new state to the forest;

forest[i][j].setStateValue(newState);

//update the Map

switch(newState)

{

case 0 : Grid::Map[i][j] = ' ';

break;

case 1 : Grid::Map[i][j] = '&';

break;

case 2 : Grid::Map[i][j] = 'x';

break;

}

}

}

}

bool Grid::IsAnyCellOnFire(Cell forest[21][21])

{

int i, j;

bool flag = false;

for( i = 0 ; i < 21; i++)

{

for( j = 0; j < 21; j++)

{

if(forest[i][j].getStateValue() == 2)

{

flag = true;

}

}

}

if(flag)

return true;

else

return false;

}

void Grid::DrawGrid(Cell forest[21][21])

{

cout << endl << endl;

for(int i = 0 ; i < 21; i++)

{

for(int j = 0; j < 21; j++)

{

cout << Grid::Map[i][j] << " ";

}

cout << endl;

}

}

void Grid:: Regrowth(Cell forest[21][21])

{

for(int i = 0 ; i < 21; i++)

{

for(int j = 0; j < 21; j++)

{

if(forest[i][j].getStateValue() == 0)

{

int random = rand() % 2;

//apply new state to the forest;

forest[i][j].setStateValue(random);

//update the Map

switch(random)

{

case 0 : Grid::Map[i][j] = ' ';

break;

case 1 : Grid::Map[i][j] = '&';

break;

}

}

}

}

}

void Grid::Info(int weather, int windDirection, int windSpeed, int groundMoisture)

{

cout << endl;

cout << " Weather conditions :\t";

switch(weather)

{

case 0 : cout << "Normal" << endl;

break;

case 1 : cout << "Windy" << endl;

break;

}

cout << " Wind Direction :\t" ;

switch(windDirection)

{

case 0 : cout << "East" << endl;

break;

case 1 : cout << "West" << endl;

break;

case 2 : cout << "South" << endl;

break;

case 3 : cout << "North" << endl;

break;

}

cout << " Wind Speed :\t\t" << windSpeed << "mph (miles per hour)" << endl;

cout << " Forest damp area :\t";

switch(groundMoisture)

{

case 0 : cout << "East" << endl;

break;

case 1 : cout << "West" << endl;

break;

case 2 : cout << "South" << endl;

break;

case 3 : cout << "North" << endl;

break;

}

}