Object Oriented C++ report

MOD003197

SID 1910937

Contents

[1. Implementation 1](#_Toc70989456)

[1.1. Base Functionality 1](#_Toc70989457)

[1.2. Additional functionality 2](#_Toc70989458)

[1.3. Evaluation 2](#_Toc70989459)

[2. Class Diagram 3](#_Toc70989460)

[3. Appendix 4](#_Toc70989461)

[3.1. How to use 4](#_Toc70989462)

[3.1.1. Controls 4](#_Toc70989463)

[3.1.2. Symbols 4](#_Toc70989464)

[3.2. Source code 4](#_Toc70989465)

[3.2.1. Main 4](#_Toc70989466)

[3.2.2. Updater.h 5](#_Toc70989467)

[3.2.3. Updater.cpp 6](#_Toc70989468)

[3.2.4. Grid.h 11](#_Toc70989469)

[3.2.5. Grid.cpp 12](#_Toc70989470)

[3.2.6. Tree.h 12](#_Toc70989471)

[3.2.7. Tree.cpp 15](#_Toc70989472)

[3.2.8. Weather.h 17](#_Toc70989473)

[3.2.9. Weather.cpp 18](#_Toc70989474)

# Implementation

## Base Functionality

The base functionality that is required has been met. The grid is 21 x 21 with a boundary of 1 layer around the edge. The state of each cell in the grid is displayed using ascii characters. The enter key updates the states of the trees and redraws the grid displaying the trees. A fire can be started in the centre cell by hitting a specified key (listed in appendix).

The fire will spread based on probability. If a tree has no adjacent trees on fire, the tree will not catch fire. If a tree has one or more adjacent trees on fire, then the tree has a chance to catch fire. The probability of the tree catching fire has been changed by additional functionality. If a tree is on fire, then it will burn down. The rate of the tree burning down has been changed by additional functionality. The functionality of an empty cell has also been changed.

## Additional functionality

Wind and rain has been added to the program. Wind affects the chance of trees catching fire. If a tree is downwind of a tree on fire, then the tree is more likely to catch fire. If a tree is upwind of a tree on fire, then the tree is less likely to catch fire.

A second grid has been added for the wetness of the ground the trees are on. The ground gets wet every time it rains. The trees are then able to soak up the water in the ground. A tree with more water in it will take longer to burn down. The water in a tree has a probability of being ‘used up’ by the tree.

Trees can grow in empty cells. A tree can grow if there is water present in the ground.

## Evaluation

The program uses main to read the user input. The user input then calls the appropriate function in the updater class. The updater class is a central class that has the logic on how to use the other classes. The classes separate the different elements of the program. Splitting into classes lets the program use the class multiple times, such as making multiple trees.

By keeping the classes generic, they can be used in different ways. The program creates two separate grids, the two grids hold different information. The class can be used in two ways because the two grids require similar functionality. If the class was made specifically for one of the grids, then the other grid would need its own class.

The code has a lack of memory management. When a tree dies it could be deleted and its node removed from the linked list.

Some parts of the updater class could be split up into more functions as well. This would improve maintainability of the code if it had more, smaller functions.

# Class Diagram

A picture containing text, sky, screenshot

Description automatically generated

# Appendix

## How to use

### Controls

The program can be operated with button presses. Table 1 shows the buttons and their function.

|  |  |
| --- | --- |
| **Button** | **Function** |
| Enter | Advances the program on step |
| x | Closes the program |
| m | Opens the menu |
| s | Shows the grid with the ground wetness |
| t | Shows the grid with trees or fire |
| f | Tries to start a fire |
| l | Opens a list of all the trees in the linked list |

Table 1

### Symbols

Table 2 shows the symbols that can appear in the grid.

|  |  |
| --- | --- |
| Symbol | Explanation |
| - | Empty cell with no tree |
| T | Tree |
| # | Tree on fire |
| . | Dry ground |
| o | Slightly wet ground |
| @ | Fully wet ground |

## Source code

### Main

#include <iostream>

#include <conio.h>

#include "Updater.h"

int main(void)

{

Updater\* updater = new Updater;

char input;

updater->showDisplay(0);

while (true)

{

input = \_getch();

if (input == 'x') //seperate if as break function in a switch would just continue code, I want to break away

{

break;

}

switch (input)

{

case 'm': //for opening the menu

updater->menu();

break;

case 'l': //Used for debugging, woulld be left out a final program

updater->showTrees();

system("pause");

break;

case 't': //Shows the defualt screen of trees

updater->showDisplay(0);

break;

case 's': //Shows the hydration of soil

updater->showDisplay(1);

break;

case 'f': //starts a fire

updater->startFire();

updater->showDisplay(0);

break;

case '\r': //advances the program

updater->updateFrame();

updater->showDisplay(0);

break;

}

}

system("pause");

}

### Updater.h

#ifndef Updater\_h

#define Updater\_h

#include "Tree.h"

#include "Grid.h"

#include "Weather.h"

class Updater

{

private:

TreeList\* forest;

/// <summary>

/// Land trees are on

/// </summary>

Grid\* land;

/// <summary>

/// for wetness

/// </summary>

Grid\* soil;

Weather\* weather;

bool firstFire; //bool to see if a fire has been set before, first fire set is always in the centre

public:

/// <summary>

/// constructor

/// </summary>

Updater();

/// <summary>

/// sets up the 2 grids and all the trees

/// </summary>

void populate();

/// <summary>

/// starts a fire

/// </summary>

void startFire();

/// <summary>

/// make ground wetter

/// </summary>

void rain();

/// <summary>

/// updates the program to the next frame

/// </summary>

void updateFrame();

/// <summary>

/// Updates all the trees

/// </summary>

void updateTrees();

/// <summary>

/// updates the grids

/// </summary>

void updateGrid();

/// <summary>

/// shows the default display

/// </summary>

/// <param name="selector">chooses whether to show trees or soil</param>

void showDisplay(int selector);

/// <summary>

/// Gets list of all trees

/// </summary>

void showTrees();

/// <summary>

/// shows the menu for the user

/// </summary>

void menu();

};

#endif // !"Updater\_h"

### Updater.cpp

#include <iostream>

#include "Updater.h"

Updater::Updater()

{

forest = new TreeList;

land = new Grid;

soil = new Grid;

weather = new Weather;

firstFire = false;

populate();

srand(time(0)); //sets seed for rand()

}

void Updater::populate()

{

for (int y = 1; y < 20; y++) //starts at 1 and ends at 20 to keep perimeter of 1 plot around edge

{

for (int x = 1; x < 20; x++)

{

forest->addTree(x, y);

land->setChar(x, y, 'T'); //sets so tree is alive and not on fire

soil->setChar(x, y, '.'); //sets so soil is dry

}

}

}

void Updater::startFire()

{

if (!forest->isEmpty()) //if not empty

{

TreeNode\* current = forest->getStart();

int x = rand() % 20 + 1; //makes random x value

int y = rand() % 20 + 1; //makes random y value

//only for first fire, all afterwards is random

//sets fire to tree and update land

if (land->getChar(10, 10) == 'T' && firstFire == false) //could have just first fire = false but this way I know a tree is being set on fire and not fire or ground

{

while (current != NULL) //loops till current is null

{

if (current->getTree()->getX() == 10 && current->getTree()->getY() == 10) //loops till current is the desired tree

{

current->getTree()->setStatus(2); //status 2 is on fire

land->setChar(10, 10, '#'); //# is on fire

break; //break out of while loop

}

current = current->getNext();

}

firstFire = true;

}

else if (land->getChar(x, y) == 'T') //checks to see if random coordinate is a tree, afterwards is just same as above

{

while (current != NULL)

{

if (current->getTree()->getX() == x && current->getTree()->getY() == y) //loops till current is the desired tree

{

current->getTree()->setStatus(2);

land->setChar(x, y, '#');

break;

}

current = current->getNext();

}

}

}

}

void Updater::rain()

{

for (int y = 1; y < 20; y++)

{

for (int x = 1; x < 20; x++)

{

char current = soil->getChar(x, y);

//checks two soil levels that can be increased and increases them

if (current == '.')

{

soil->setChar(x, y, 'o');

}

else if (current == 'o')

{

soil->setChar(x, y, '@');

}

}

}

}

void Updater::updateTrees()

{

TreeNode\* current;

current = forest->getStart();

while (current != NULL)

{

//update tree

int xpos = current->getTree()->getX();

int ypos = current->getTree()->getY();

int rnd = rand() % 100 + 1;

int chance = 0;

if (land->getChar(xpos, ypos) == 'T') // See if tree will set fire

{

//each if is an adjacent tree (or border)

if (land->getChar(xpos - 1, ypos) == '#')

{

chance = chance + 40 + weather->getWindX();

}

if (land->getChar(xpos + 1, ypos) == '#')

{

chance = chance + 40 - weather->getWindX();

}

if (land->getChar(xpos, ypos - 1) == '#')

{

chance = chance + 40 + weather->getWindY();

}

if (land->getChar(xpos, ypos + 1) == '#')

{

chance = chance + 40 - weather->getWindY();

}

if (rnd <= chance) //if rnd is less than chance set tree to be on fire

{

current->getTree()->setStatus(2);

}

else // if tree is not set on fire update trees hydration

{

if (current->getTree()->getHydration() < 2)

{

if (soil->getChar(xpos, ypos) == '@' && 10 > rand() % 100) // 10% chance of hydration on wet soil

{

current->getTree()->setHydration(current->getTree()->getHydration() + 1);

soil->setChar(xpos, ypos, 'o'); //reduce soil wetness level

}

else if ((soil->getChar(xpos, ypos) == 'o' && 5 > rand() % 100)) // 5% chance of hydration on moist soil

{

current->getTree()->setHydration(current->getTree()->getHydration() + 1);

soil->setChar(xpos, ypos, '.');

}

}

if (current->getTree()->getHydration() > 0 && 10 > rand() % 100) //10 chance of using current hydration

{

current->getTree()->setHydration(current->getTree()->getHydration() - 1);

}

}

}

else if (land->getChar(xpos, ypos) == '-') // See if tree will grow in empty soil

{

if (soil->getChar(xpos, ypos) == '@')

{

if (3 > rand() % 100) // 3% chance to grow on fully wet soil

{

land->setChar(xpos, ypos, 'T');

current->getTree()->setStatus(1);

current->getTree()->setHydration(0);

soil->setChar(xpos, ypos, '.');

}

}

else if (soil->getChar(xpos, ypos) == 'o')

{

if (1 > rand() % 100) // 1% chance to grow on half wet soil

{

land->setChar(xpos, ypos, 'T');

current->getTree()->setStatus(1);

current->getTree()->setHydration(0);

soil->setChar(xpos, ypos, '.');

}

}

}

else if (land->getChar(xpos, ypos) == '#') // See if tree burns down

{

if (current->getTree()->getHydration() <= 0)

{

current->getTree()->setStatus(0);

soil->setChar(xpos, ypos, '.');

}

else

{

current->getTree()->setHydration(current->getTree()->getHydration() - 1);

}

}

current = current->getNext();

}

}

void Updater::updateGrid()

{

TreeNode\* current;

current = forest->getStart();

while (current != NULL)

{

//get trees coordinates

int xpos = current->getTree()->getX();

int ypos = current->getTree()->getY();

if (current->getTree()->getStatus() == 1) //if a tree is alive and not on fire

{

land->setChar(xpos, ypos, 'T');

}

else if (current->getTree()->getStatus() == 2) //if a tree is on fire

{

land->setChar(xpos, ypos, '#');

}

else

{

land->setChar(xpos, ypos, '-');

}

current = current->getNext();

}

}

void Updater::updateFrame()

{

if (!forest->isEmpty())

{

if (weather->getRainChance() >= 20)

{

rain();

weather->resetRainChance();

}

updateTrees(); //update each tree, chance to set on fire if neighbors are on fire

updateGrid(); //update land so each tree is correctly displayed

//all trees need status updated BEFORE the grid so fire doesn't spread or die out too soon

weather->weatherChange(); //update weather

}

}

void Updater::showDisplay(int selector)

{

system("cls");

Grid\* toDisplay;

if (selector == 0) //choose what grid to display

{

toDisplay = land;

}

else if (selector == 1)

{

toDisplay = soil;

}

else // just in case

{

toDisplay = land;

}

for (int y = 0; y < 21; y++) //creates the display

{

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

{

if (x == 20)

{

std::cout << toDisplay->getChar(x, y) << std::endl;

}

else

{

std::cout << toDisplay->getChar(x, y) << " ";

}

}

}

std::cout << "wind speed is currently: " << weather->getWindX() << " on the x axis" << std::endl;

std::cout << "and " << weather->getWindY() << " on the y axis" << std::endl << std::endl;

std::cout << "Press m to bring up menu or" << std::endl;

std::cout << "Press enter to continue ";

}

void Updater::showTrees()

{

system("cls");

forest->listAll();

}

void Updater::menu()

{

system("cls");

std::cout << "List of options:" << std::endl;

std::cout << "m - Menu button" << std::endl;

std::cout << "s - Show hydration of soil" << std::endl;

std::cout << "t - Show trees" << std::endl;

std::cout << "f - chance to start fire" << std::endl;

std::cout << "x - close program" << std::endl;

std::cout << "Enter key - for progressing program" << std::endl << std::endl;

}

### Grid.h

#ifndef Grid\_h

#define Grid\_h

/// <summary>

/// class 2d array grids of 21 by 21

/// </summary>

class Grid

{

private:

char plot[21][21];

public:

/// <summary>

/// constructor

/// </summary>

Grid();

/// <summary>

/// get the char at specified coordinates

/// </summary>

/// <param name="x">x coordinate</param>

/// <param name="y">y coordinate</param>

/// <returns></returns>

char getChar(int x, int y);

/// <summary>

/// sets the char at specified coordinate

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

/// <param name="letter">char to be set</param>

void setChar(int x, int y, char letter);

};

#endif // !Grid\_h

### Grid.cpp

#include "Grid.h"

Grid::Grid()

{

//Loops though all available x and y coordinates and sets to -

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

{

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

{

plot[x][y] = '-';

}

}

}

char Grid::getChar(int x, int y)

{

return plot[x][y];

}

void Grid::setChar(int x, int y, char letter)

{

plot[x][y] = letter;

}

### Tree.h

#ifndef Tree\_h

#define Tree\_h

/// <summary>

/// Each tree object

/// </summary>

class Tree

{

private:

int status; //status of tree, 0 = no tre, 1 = tree, 2 = on fire

int xcoord; //trees x coordinate

int ycoord; //trees y coordinate

int hydration; //hydration level of tree from 0 to 2

public:

/// <summary>

/// Construtor for the tree

/// </summary>

/// <param name="x">trees x coordinate</param>

/// <param name="y">trees y coordinate</param>

Tree(int x, int y);

/// <summary>

/// Get the trees x coordinate

/// </summary>

/// <returns></returns>

int getX();

/// <summary>

/// get the trees y coordinate

/// </summary>

/// <returns></returns>

int getY();

/// <summary>

/// get the trees hydration level

/// </summary>

/// <returns></returns>

int getHydration();

/// <summary>

/// set the trees hydration level

/// </summary>

/// <param name="hydro"></param>

void setHydration(int hydro);

/// <summary>

/// get the trees current status

/// </summary>

/// <returns></returns>

int getStatus();

/// <summary>

/// set the trees current status

/// </summary>

/// <param name="newStatus"></param>

void setStatus(int newStatus);

};

/// <summary>

/// Create a node for trees for a linked list

/// </summary>

class TreeNode

{

private:

Tree\* tree;

TreeNode\* next;

public:

/// <summary>

/// construtor

/// </summary>

/// <param name="x">trees x coordinate</param>

/// <param name="y">trees y coordinate</param>

TreeNode(int x, int y);

//get the tree object the node has

Tree\* getTree();

/// <summary>

/// get the next node in the linked list

/// </summary>

/// <returns></returns>

TreeNode\* getNext();

/// <summary>

/// set the next node in the linked list

/// </summary>

/// <param name="nextNode"></param>

void setNext(TreeNode\* nextNode);

};

/// <summary>

/// Linked list of trees

/// </summary>

class TreeList

{

private:

TreeNode\* start;

TreeNode\* end;

public:

/// <summary>

/// Constructor

/// </summary>

TreeList();

/// <summary>

/// Find out if the linked list is empty

/// </summary>

/// <returns></returns>

bool isEmpty();

/// <summary>

/// add a tree to the linked list

/// </summary>

/// <param name="x">trees x coordinate</param>

/// <param name="y">trees y coordinate</param>

void addTree(int x, int y);

/// <summary>

/// list all trees in linked list

/// </summary>

void listAll();

/// <summary>

/// get the first node in the linked list

/// </summary>

/// <returns></returns>

TreeNode\* getStart();

};

#endif // !Tree\_h

### Tree.cpp

#include "Tree.h"

#include <iostream>

Tree::Tree(int x, int y)

{

xcoord = x;

ycoord = y;

status = 1;

hydration = 1; //when constructed the tree has a hydration of 1

}

int Tree::getX()

{

return xcoord;

}

int Tree::getY()

{

return ycoord;

}

int Tree::getHydration()

{

return hydration;

}

void Tree::setHydration(int hydro)

{

hydration = hydro;

}

int Tree::getStatus()

{

return status;

}

void Tree::setStatus(int newStatus)

{

status = newStatus;

}

TreeNode::TreeNode(int x, int y)

{

tree = new Tree(x, y);

next = NULL;

}

Tree\* TreeNode::getTree()

{

return tree;

}

TreeNode\* TreeNode::getNext()

{

return next;

}

void TreeNode::setNext(TreeNode\* nextNode)

{

next = nextNode;

}

TreeList::TreeList()

{

start = NULL;

end = NULL;

}

bool TreeList::isEmpty()

{

return start == NULL;

}

void TreeList::addTree(int x, int y)

{

TreeNode\* current;

current = new TreeNode(x, y);

if (end == NULL) //if theres no trees the current tree will be set for the start and the end

{

start = current;

end = current;

}

else

{

end->setNext(current); //sets nodes currently held in end next node to be current

end = current; //sets current to node at end of list

}

}

void TreeList::listAll()

{

TreeNode\* current;

if (!isEmpty()) // if not empty

{

std::cout << "start of list: " << std::endl;

current = start;

while (current != NULL) //will loop while current is an object

{

std::cout << "Tree is " << current->getTree()->getX() << ", " << current->getTree()->getY() << std::endl;

current = current->getNext();

}

std::cout << "End of list" << std::endl;

}

else

{

std::cout << "list empty" << std::endl;

}

}

TreeNode\* TreeList::getStart()

{

return start;

}

### Weather.h

#ifndef Weather\_h

#define Weather\_h

class Weather

{

private:

int windX;

int windY;

int rainChance;

public:

/// <summary>

/// constructor

/// </summary>

Weather();

/// <summary>

/// updates the weather

/// </summary>

void weatherChange();

/// <summary>

/// gets the winds x speed

/// </summary>

/// <returns></returns>

int getWindX();

/// <summary>

/// gets the winds y speed

/// </summary>

/// <returns></returns>

int getWindY();

/// <summary>

/// brings rain chance back to 0

/// </summary>

void resetRainChance();

/// <summary>

/// gets the current chance of rain

/// </summary>

/// <returns></returns>

int getRainChance();

};

#endif // !Weather\_h

### Weather.cpp

#include <iostream>

#include "Weather.h"

Weather::Weather()

{

windX = 0;

windY = 0;

srand(time(0));

rainChance = 0;

}

void Weather::weatherChange()

{

rainChance = rainChance + rand() % 3; //increases chance of rain by 0 - 3

//changes both winds x and y, can change by -5 to +5 units

int xChange = rand() % 11 + (-5);

int yChange = rand() % 11 + (-5);

if (windX + xChange >= 20) //if wind will end up faster than 20

{

windX = 20; //set wind to 20

}

else if (windX + xChange <= -20) //same as above but -20

{

windX = -20;

}

else

{

windX = windX + xChange;

}

if (windY + yChange >= 20) //same as above but on y axis

{

windY = 20;

}

else if (windY + yChange <= -20)

{

windY = -20;

}

else

{

windY = windY + yChange;

}

}

int Weather::getWindX()

{

return windX;

}

int Weather::getWindY()

{

return windY;

}

void Weather::resetRainChance()

{

rainChance = 0;

}

int Weather::getRainChance()

{

return rainChance;

}