Ella Ley

CIS 250

Week 17

Memory Match Final Project

## Requirements

Design and develop a program that using one of three possible categories, displays a gameboard in which a memory match game can be played, with encapsulation in use for all necessary game functions

# Code:

## MemoryMatch.hpp

/\*\*

\* @file MemoryMatch.hpp

\* @author Ella Ley

\* @brief Memory Matching Game

\* @version 1.0

\* @date 2024-11-08

\*

\*

\*/

//Requirements

//3 Themes, each based on one of the three txt files in \dir themes

//Have each themeFile have a term that describes it

//

//USER MENU

//Level of Play: 4x4, 6x6, 8x8

//Speed of Play: How long in seconds words in the grids will be shown: 2, 4, 6

//

//At game start, themeFile name should be shown in all grid spaces

//

//MECHANICS

//Amount of words needed is gridSize / 2

//Each word will be present twice on the board

//Store the words and their respective locations in a 2D array

//

//GAMEPLAY LOOP

// 1. User selects first square, themeFile term is replaced by its respective word from the word array

// 2. User selects second square, themeFile term replaced by respective word from game array

// 3. IF SAME: Keep terms displayed, those terms are no longer selectable

// IF DIFFERENT: Terms remain on screen for selected difficulty time, then revert to themeFile name

#pragma once

#include <iostream>

#include <iomanip>

class MemoryMatch{

private:

std::string AllWords[51];

std::string themeFile;

std::string themeName;

std::string displayThemeName;

std::string \* SelectedWords;

std::string \* ptrChosenWords = nullptr;

int boardSize;

int difficultyTime;

int row;

bool wordsSelected;

int totalWords;

public:

MemoryMatch();

int sizeSelect();

int timeSelect();

//themeFile Selection, loads words into array

bool themeSelect();

//void boardInit();

std::string \* getWords();

int getWordCount();

int getSize();

std::string getDisplayTheme();

~MemoryMatch(){};

};

## MemoryMatch.cpp

#include "MemoryMatch.hpp"

#include <iostream>

#include <iomanip>

#include <fstream>

#include <cmath>

/\*\*

\* @brief Construct a new Memory Match:: Memory Match object

\*

\*/

MemoryMatch::MemoryMatch(){

themeFile = "none";

boardSize = 0;

difficultyTime = 0;

row = 0;

displayThemeName = "none";

}

/\*\*

\* @brief Selects size for the gameboard for later use

\*

\* @return int Dimension of one side of the gameboard

\*/

int MemoryMatch::sizeSelect(){

bool done = true;

while(done){

int userSelection;

std::cout << "Select Board Size" << std::endl;

std::cout << "1. 4x4" << std::endl;

std::cout << "2. 6x6" << std::endl;

std::cout << "3. 8x8\n" << std::endl;

std::cout << "Enter Selection Number: ";

std::cin >> userSelection;

switch(userSelection){

case 1:

boardSize = 4;

done = false;

break;

case 2:

boardSize = 6;

done = false;

break;

case 3:

boardSize = 8;

done = false;

break;

}

}

return boardSize;

}

/\*\*

\* @brief Selects how long the words will be displayed if not matching

\*

\* @return int How long the user can see the words before the board is reset

\*/

int MemoryMatch::timeSelect(){

int userSelection;

bool done = true;

while(done){

std::cout << "Select how long words are displayed when pair do not match" << std::endl;

std::cout << "1. 6 Seconds" << std::endl;

std::cout << "2. 4 Seconds" << std::endl;

std::cout << "3. 2 Seconds" << std::endl;

std::cout << "Enter Selection Number: ";

std::cin >> userSelection;

switch(userSelection){

case 1:

difficultyTime = 6;

done = false;

break;

case 2:

difficultyTime = 4;

done = false;

break;

case 3:

difficultyTime = 2;

done = false;

break;

}

}

return difficultyTime;

}

/\*\*

\* @brief Selects theme for 3 possible choices corresponding to its respective .txt file

\*

\* @return true When a theme has been chosen and words are loaded into the @var AllWords array

\* @return false Does not occur as the theme will only exit upon successful loading of a theme

\*/

bool MemoryMatch::themeSelect(){

bool done = true;

int userSelection;

while(done){

std::cout << "1. Human Proteins" << std::endl;

std::cout << "2. US Military Aircraft" << std::endl;

std::cout << "3. x86 Assembly Instructions" << std::endl;

std::cout << "Select Theme: ";

std::cin >> userSelection;

switch(userSelection){

case 1:

themeFile = "./themes/humanProteins.txt";

themeName = "Human Proteins";

displayThemeName = "Protein";

done = false;

//std::cout << "DEBUG: Proteins" << std::endl;

break;

case 2:

themeFile = "./themes/USmilitaryaircraft.txt";

themeName = "US Military Aircraft";

displayThemeName = "Aircraft";

done = false;

//std::cout << "DEBUG: Aircraft" << std::endl;

break;

case 3:

themeFile = "./themes/x86assembly.txt";

themeName = "x86 Assembly";

displayThemeName = "x86";

done = false;

//std::cout << "DEBUG: Assembly" << std::endl;

break;

}

}

//Import File Contents

std::ifstream inputFile(themeFile);

std::string importWord;

int i = 0;

while(getline(inputFile, importWord)){

AllWords[i] = importWord;

i++;

}

inputFile.close();

return true;

}

/\*\*

\* @brief Sets up the array that selects which words are loaded into the gameboard

\*

\* @return std::string\* Pointer to the array of strings which contains all of the words loaded into gameboard

\*/

std::string \* MemoryMatch::getWords(){

int neededWords = (pow(boardSize, 2) / 2);

totalWords = neededWords;

std::string ChosenWords[neededWords];

ptrChosenWords = ChosenWords;

//Default Values for all array members

for(int j = 0; j < ((neededWords) / 2) - 1; j++){

ptrChosenWords[j] = "none";

}

std::string newWord;

int randomNum = 0;

int openIndex = 0;

bool inArray = true;

std::string potentialWord;

while(openIndex < neededWords){

randomNum = rand() % 50;

potentialWord = AllWords[randomNum];

for(int i = 0; i <= openIndex; i++){

if(potentialWord != ptrChosenWords[i]){

inArray = false;

}

else{

inArray = true;

//std::cout << "Word Already in Array" << std::endl;

}

}

if(inArray){

break;

}

else{

ptrChosenWords[openIndex] = potentialWord;

openIndex++;

}

}

wordsSelected = true;

return ptrChosenWords;

}

/\*\*

\* @brief Simple Getter for the total word count

\*

\* @return int Count of how many words will be needed in the gameboard

\*/

int MemoryMatch::getWordCount(){

return totalWords;

}

/\*\*

\* @brief Simple Getter for gameboard size

\*

\* @return int Size of one side of the gameboard

\*/

int MemoryMatch::getSize(){

return boardSize;

}

/\*\*

\* @brief Simple Getter for the theme name

\*

\* @return std::string Abbreviated name of the theme for use in the gameboard

\*/

std::string MemoryMatch::getDisplayTheme(){

return displayThemeName;

}

## game.hpp

#pragma once

#include "MemoryMatch.hpp"

#include <iostream>

class game : public MemoryMatch{

private:

MemoryMatch internalGame;

std::string displayName;

bool themeSelected, firstPlace, secondPlace;

int rows, cols, totalCells, randomRow, randomCol, wordIndex, timeDiff;

//Gameboard coordinates

int x1, x2, y1, y2;

int totalWords = 0;

int matchedWords;

const int cellWidth = 8;

std::string \*\* board = nullptr;

std::string \* chosenWords = nullptr;

std::string \* wordPool = nullptr;

bool \* revealed = nullptr;

bool finished;

public:

game();

bool start();

int gameLoop();

void printBoard(std::string\*\*, const bool\*, const int, int, int);

int checkAndRevealWords(std::string\*\*, bool \*, int, int, int, int, int, int);

~game();

};

## game.cpp

#include "MemoryMatch.hpp"

#include "game.hpp"

#include <iostream>

#include <unistd.h>

#include <random>

#include <iomanip>

//For windows machines

//#include <windows.h>

game::game(){

themeSelected = internalGame.themeSelect();

displayName = internalGame.getDisplayTheme();

timeDiff = internalGame.timeSelect();

rows = internalGame.sizeSelect();

cols = internalGame.getSize();

totalCells = rows \* cols;

revealed = new bool[rows \* cols]{false};

matchedWords = 0;

finished = false;

}

/\*\*

\* @brief Startup script for game

\*

\* @return true If there are enough words to fill board, board size is allocated, and words are placed into board array

\* @return false In case enough words are somehow not chosen for the board

\*/

bool game::start(){

//Checking if there are enough words to fill the whole board

if(totalCells > internalGame.getWordCount()){

std::cout << "GameError: Not enough words to fill board" << std::endl;

return false;

}

//Allocating board size dynamically

board = new std::string \*[rows];

for(int i = 0; i < rows; ++i){

board[i] = new std::string[cols];

}

//Fetching Words from chosenWords array

chosenWords = internalGame.getWords();

wordPool = new std::string[totalCells];

for(int i = 0; i < totalCells / 2; ++i){

wordPool[i] = chosenWords[i];

totalWords++;

}

//Placing the word randomly into the array twice

wordIndex = 0;

for(int i = 0; i < totalCells / 2; i++) {

firstPlace = false;

secondPlace = false;

while(!firstPlace){

//Generating the random numbers for the array indexes

randomRow = rand() % rows;

randomCol = rand() % rows;

//Checks if the array has a blank value, replaces it with word if it does and sets firstPlace to true

if(board[randomRow][randomCol] == ""){

board[randomRow][randomCol] = wordPool[wordIndex];

firstPlace = true;

} else{

std::cout << "Array at row " << randomRow << " and column " << randomCol << " filled. Retrying" << std::endl;

}

}

//If secondPlace is false, then if the first word has been placed

while(!secondPlace) {

if(firstPlace){

randomRow = rand() % rows;

randomCol = rand() % rows;

if(board[randomRow][randomCol] == "") {

board[randomRow][randomCol] = wordPool[wordIndex];

wordIndex++;

secondPlace = true;

} else{

std::cout << "Array at row " << randomRow << " and column " << randomCol << " filled. Retrying" << std::endl;

}

}

}

}

return true;

}

/\*\*

\* @brief Main game loop, contains the logic for finding the cells in the 2d array

\*

\* @return int Returns 0 if all the words are found, else returns 1 as a catch condition

\*/

int game::gameLoop(){

system("clear");

while(!finished) {

printBoard(board, revealed, cellWidth, rows, cols);

std::cout << "First Word" << std::endl;

std::cout << "Enter Row: ";

std::cin >> x1;

std::cout << "Enter Column: ";

std::cin >> y1;

std::cout << "First Word: " << board[x1 - 1][y1 - 1] << std::endl;

int index1 = (x1 - 1) \* cols + (y1 - 1);

revealed[index1] = true;

printBoard(board, revealed, cellWidth, rows, cols);

revealed[index1] = false;

std::cout << "Second Word" << std::endl;

std::cout << "Enter Row: ";

std::cin >> x2;

std::cout << "Enter Column: ";

std::cin >> y2;

if(x1 < 1 || x1 > rows || y1 < 1 || y1 > cols){

std::cout << "Invalid Coordinates. Try Again \n";

continue;

}

if (revealed[index1]) {

std::cout << "Tile already revealed. Try another.\n";

continue;

}

if(x2 < 1 || x2 > rows || y2 < 1 || y2 > cols ) {

std::cout << "Invalid coordinates. Try again. \n";

continue;

}

int index2 = (x2 - 1) \* cols + (y2 - 1);

if(revealed[index2]) {

std::cout << "Tile already revealed. Try again. \n";

continue;

}

if(x1 == x2 && y1 == y2) {

std::cout << "The same tile was selected twice. Try again. \n";

continue;

}

matchedWords += checkAndRevealWords(board, revealed, rows, cols, x1, y1, x2, y2);

if(matchedWords >= rows \* 2){

std::cout << "All Words Found!" << std::endl;

finished = true;

return 0;

}

}

return 1;

}

/\*\*

\* @brief Prints out the board, with cells containing either the placeholder or the shown word

\*

\* @param board Poiter to the gameboard array

\* @param revealed Pointer that stores which of the values in the gameboard are revealed

\* @param cellWidth Controls the width of the cells in the board, can be changed

\* @param rows Rows needed to display the gameboard

\* @param cols Columns needed to display the gameboard

\*/

void game::printBoard(std::string\*\* board, const bool\* revealed, const int cellWidth, int rows, int cols) {

std::cout << " ";

for (int col = 0; col < cols; ++col)

std::cout << std::setw(cellWidth) << col + 1;

std::cout << std::endl;

std::cout << " +";

for (int col = 0; col < cols; ++col)

std::cout << std::string(cellWidth - 1, '-') << "+";

std::cout << std::endl;

for (int row = 0; row < rows; ++row) {

std::cout << row + 1 << " |";

for (int col = 0; col < cols; ++col) {

int index = row \* cols + col;

if (revealed[index]) {

std::cout << std::setw(cellWidth - 2) << board[row][col] << std::setw(cellWidth - 6) << "|";

} else {

std::cout << std::setw(cellWidth - 2) << "■" << std::setw(cellWidth - 4) << "|";

}

}

std::cout << std::endl;

std::cout << " +";

for (int col = 0; col < cols; ++col)

std::cout << std::string(cellWidth - 1, '-') << "+";

std::cout << std::endl;

}

}

/\*\*

\* @brief Checks if the words at the coordinates given by the user are the same

\*

\* @param board Pointer to the 2d array which contains all of the words in the gameboard

\* @param revealed Pointer to a 1d array which stores whether a word has been revealed as a boolean

\* @param rows Rows necessary to display gameboard, Needed for printBoard()

\* @param cols Columns necessary to display gameboard, Needed for printBoard()

\* @param x1 User chosen x coordinate in gameboard of First Word

\* @param y1 User chosen y coordinate in gameboard of First Word

\* @param x2 User chosen x coordinate in gameboard for Second Word

\* @param y2 User chosen y coordinate in gameboard for Second Word

\* @return int Returns 0 if the words do not match, else returns 1 since words match which updates @var matchedWords

\*/

int game::checkAndRevealWords(std::string \*\* board, bool \* revealed, int rows, int cols, int x1, int y1, int x2, int y2){

int index1 = (x1 - 1) \* cols + (y1 - 1);

int index2 = (x2 - 1) \* cols + (y2 - 1);

revealed[index1] = true;

revealed[index2] = true;

printBoard(board, revealed, 8, rows, cols);

if(board[x1- 1][y1 - 1] != board[x2 - 1][y2 - 1]){

std::cout << "Words do not match" << std::endl;

sleep(timeDiff);

revealed[index1] = false;

revealed[index2] = false;

system("clear");

return 0;

} else{

std::cout << "The Words Match" << std::endl;

system("clear");

return 1;

}

}

/\*\*

\* @brief Destroy the game::game object

\*

\*/

game::~game(){

for(int i = 0; i < rows; ++i){

delete[] board[i];

}

delete[] board;

delete[] revealed;

delete[] wordPool;

}

## main.cpp

#include "MemoryMatch.hpp"

#include "game.hpp"

#include <iostream>

//For Linux Machines

#include <unistd.h>

#include <random>

//For Windows Machines

//#include <windows.h>

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

int main() {

bool done = false;

game mainGame;

while(!done){

done = mainGame.start();

}

mainGame.gameLoop();

mainGame.~game();

return 0;

}

## Terminal Output:







