

```

/*
* File: main.cpp
* Author: Michael Cooper
* Created on 5/5/22, 11:00 AM
* Purpose: Project 1 - Connect 4 - Version 4 FINAL BUILD
* Further expanded project with a text output that will keep track of players
* and their win counts, further optimized and made comments.
*/

```

```

#include "Game.h"
#include "Board.h"

```

```

int main() {
    // Initalizes and runs game
    Game game;
    game.start();
    game.run();
}

```

```

/*
* File: Board.h
* Author: Michael Cooper
* Purpose: Board Header File
* Created on 5/1/22
*/

```

```

#ifndef BOARD_H
#define BOARD_H

```

```

#include <string>
#include <array>
#include <set>    // Associative Container: set
#include <iterator>
#include <map>    // Associate Container: map
#include <list>

```

```

#include <algorithm>

```

```

#include "Player.h"

```

```

const int BOARD_SIZE = 8*4;

```

```

class Board {
public:
    std::list<char> board; // Sequences: list
    std::map<std::string, char> player_markers;
    Board(const std::list<Player>& players);
public:
    bool mark(std::string name, int col);
    char vertWin();
    char hortWin();
    char diagWin();
    char win();
}

```

```

    void printBoard();
};

#endif // BOARD_H

/*
 * File: Board.cpp
 * Author: Michael Cooper
 * Purpose: Board Source File
 * Created on 5/1/22
 */

#include "Board.h"

#include <set>
#include <iterator>
#include <iostream>

using namespace std;

std::array<char,3> TEMP = {'#', 'x', 'o'};

std::set<char> MARKERS(TEMP.begin(), TEMP.end());

static std::map<string, char>* _player_markers = nullptr;

// Function for stl for_each for assigning the proper
// information to all players for the board
bool fe(const Player& player) {
    static int li = 2;
    _player_markers->insert(std::pair<string, char>(player.getName(), *(next(MARKERS.cbegin(),li)))); //
    MARKERS.cbegin() is bidirectional
    li++;
    li % 3 == 0? li = 1: li*=1;
    return true;
}

// Constructor for the board
Board::Board(const std::list<Player>& players) {
    _player_markers = &player_markers;
    for_each(players.cbegin(),
        players.cend(), fe);
    _player_markers = nullptr;
    board.resize(BOARD_SIZE);
    fill(board.begin(),
        board.end(), *MARKERS.cbegin()); // Mutating Algorithm: fill()
}

// Checks if the user can mark the board
// and checks the far most bottom of
// a column
bool Board::mark(string name, int col) {
    if(col < 0 && col >= 8) {

```

```

    return false;
}
for(std::map<string, char>::iterator itr = player_markers.begin();
    itr != player_markers.end(); itr++) { // itr is bidirectional
    if(itr->first == name) {
        for(int i = 3; i >= 0; i--) {
            auto spot = next(board.begin(), (col + 8 * i));
            if(*spot != '#')
                continue;
            else {
                *spot = itr->second;
                break;
            }
        }
        return true;
    }
}
return false;
}

```

```

// iterates through the board and
// prints each element by a 8x4 grid
void Board::printBoard() {
    int i = 0;
    for(auto itr = board.begin(); itr != board.end(); itr++) {
        if(i % 8 == 0)
            cout << endl;
        cout << *itr << " ";
        i++;
    }
}

```

```

// Checks if a player has won in the
// horizontal direction
char Board::horWin() {
    int i = 0;
    char prev = *board.begin();
    for(auto itr = next(board.begin(), 1);
        itr != board.end(); itr++) {
        if(prev == *itr && prev != '#') {
            i++;
        }
        else
            i = 0;
        if(i == 3)
        {
            return prev;
        }
        prev = *itr;
    }
    return '#';
}

```

```

// Checks if a player has won in the
// vertical direction
char Board::vertWin() {
    int i = 0;
    char prev = '#';
    for(auto itr = board.begin();
        itr != board.end(); itr++) {
        prev = *itr;
        for(int i = 3; i >= 0; i--) {
            auto spot = next(itr, 8 * i);
            if(prev == *spot)
                i++;
            else
                i = 0;
            if(i == 4)
            {
                return prev;
            }
            prev = *spot;
        }
    }
    return '#';
}

```

```

// Helper function to return
// marker at board position
char markerAt(std::list<char>& board, int i) {
    return *next(board.begin(), i);
}

```

```

// Checks if a player has won from a
// a diagonal direction going from
// left to right and right to left
// from the board
char Board::diagWin() {
    int l = 0;
    char prev = '#';
    //descending from the left to right
    // Example:
    /*
    # # # # # * # # # Step 4
    # # # # # * # # Step 3
    # # # # # # * # Step 2
    # # # # # # # * Step 1
    */
    for(int i = 0; i <= 5; i++) {
        for(int j = 0; j < 4; j++) {
            if(markerAt(board, i + 8*j) == '#')
                break;
            if(prev == markerAt(board, i+8*j))
                l++;
            else
                l = 0;
        }
    }
}

```

```

        if(i == 4)
            return prev;
    }
}
l = 0;
prev = '#';
// ascending the right to left
// Example:
/*
# # # x # # # # Step 1
# # x # # # # # Step 2
# x # # # # # # Step 3
x # # # # # # # Step 4
*/
for(int i = 7; i >= 4; i--) {
    for(int j = 3; j >= 0; j--) {
        if(markerAt(board, i + 8*j) == '#')
            break;
        if(prev == markerAt(board, i+8*j))
            l++;
        else
            l = 0;
        if(i == 4)
            return prev;
    }
}
return '#';
}

```

// Checks for every case

// for a win

```
char Board::win() {
```

```
    char c = '#';
```

```
    c = vertWin();
```

```
    if(c != '#')
```

```
        return c;
```

```
    c = hortWin();
```

```
    if(c != '#')
```

```
        return c;
```

```
    c = diagWin();
```

```
    return c;
```

```
}
```

```
/*
```

```
* File: Game.h
```

```
* Author: Michael Cooper
```

```
* Purpose: Game Header File
```

```
* Created on 4/30/22
```

```
*/
```

```

#ifndef GAME_H
#define GAME_H

#include <stack>
#include <list>
#include "Player.h"

class GameState;

class Game {
public:
    std::stack<GameState*> gameState; // Container Adaptor: stack
    std::list<Player> players; // Sequence: list
public:
    Game();
    ~Game();
    void start();
    void run();
};

#endif

/*
 * File: Game.cpp
 * Author: Michael Cooper
 * Purpose: Game Source File
 * Created on 4/30/22
 */

#include "Game.h"
#include "GameState.h"

using namespace std;

//Constructor
Game::Game() {

}

// Deconstructor
Game::~~Game() {

}

// Should only be called once!
void Game::start() {
    GameState* menu = new MenuState(this);
    gameState.push(menu);
}

void Game::run() {
    gameState.top()->run();
}

```

```

/*
 * File: GameState.h
 * Author: Michael Cooper
 * Purpose: Game State Header File
 * Created on 4/29/22
 */

#ifndef GAMESTATE_H
#define GAMESTATE_H

#include "Game.h"

class Game;

class GameState {
public:
    virtual void run() = 0;
};

class MenuState : public GameState {
private:
    Game* game;
public:
    MenuState(Game* game);
    void run();
};

class CreateState : public GameState {
private:
    Game* game;
public:
    CreateState(Game* game);
    void run();
};

class PlayState : public GameState {
private:
    Game* game;
public:
    PlayState(Game* game);
    void run();
};

class RuleState : public GameState {
private:
    Game* game;
public:
    RuleState(Game* game);
    void run();
};

#endif // GAMESTATE_H

```

```

/*
 * File: GameState.cpp
 * Author: Michael Cooper
 * Purpose: Game State Source File
 * Created on 4/29/22
 */

#include "GameState.h"
#include "Board.h"
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <fstream>
#include <deque>

using namespace std;

// Prints the menu options
void print_menu() {
    std::cout << "Welcome to Connect 4!" << endl;
    std::cout << "Enter the following options..." << endl;

    std::cout << "1. Rules" << endl;
    std::cout << "2. Play" << endl;
    std::cout << "3. Exit" << endl;
}

MenuState::MenuState(Game* game) {
    this->game = game;
}

// Will print all the options the user
// can make and redirect them
// to the appropriate state
void MenuState::run() {
    while (true) {
        print_menu();
        string soption;
        int option = 3;
        getline(cin,soption);
        option = stoi(soption);
        if (option == 1) {
            game->gameState.push(new RuleState(game));
            game->gameState.top()->run();
        } else if(option == 2) {
            game->gameState.push(new CreateState(game));
            game->gameState.top()->run();
        } else {
            std::cout << "\nGoodbye!" << endl;
            break;
        }
    }
}

```



```

    }
    game->gameState.pop();
}

RuleState::RuleState(Game* game) {
    this->game = game;
}

// Give the player the information
// on how to play the game
void RuleState::run() {
    cout << "Welcome to Connect 4!\nHere are the rules." << endl;
    cout << "Players will go turn by turn adding their marker to the board." << endl;
    cout << "and in order for any player can win they must meet the following conditions"<<endl;

    cout << "A player must reach 4 spots diagonally\n";
    cout << "###x####" << endl;
    cout << "##xo####" << endl;
    cout << "#xoo####" << endl;
    cout << "xooo####" << endl;

    cout << "A player must reach 4 spots horizontally\n";
    cout << "#####" << endl;
    cout << "#####" << endl;
    cout << "#xxxx####" << endl;
    cout << "#ooxx####" << endl;

    cout << "A player must reach 4 spots vertically\n";
    cout << "#x#####" << endl;
    cout << "#x#####" << endl;
    cout << "#x#####" << endl;
    cout << "#xooo####" << endl;

    std::cout << "Enter the following options..." << endl;

    std::cout << "1. Play" << endl;
    std::cout << "2. Return to Menu" << endl;

    string soption;
    int option = 3;

    getline(cin,soption);
    option = stoi(soption);

    if (option == 1) {
        game->gameState.push(new CreateState(game));
        game->gameState.top()->run();
    }
    game->gameState.pop();
}

CreateState::CreateState(Game* game) {
    this->game = game;
}

```

```

}

bool comp(const Player& a, const Player& b) {
    return a.getWins() > b.getWins();
}

// Is responsible for making players
// or checking if there are any
// players in the save file
void CreateState::run() {

    vector<Player> players(game->players.begin(), game->players.end());

    ifstream myfile ("players.txt");
    // Checking if there is a save file
    // and if the save file is not empty
    if(myfile.is_open() || myfile.peek() != std::ifstream::traits_type::eof()) {
        string line;
        int i = 0;

        deque<string> names;
        deque<int> wins;

        // Goes through the file
        // and searches for names and wins
        while(getline(myfile, line)) {
            if(line.empty()) continue;
            if(i % 2 == 0) {
                names.push_back(line);
            } else {
                wins.push_back(stoi(line));
            }
            i++;
        }

        // Make new players according to the
        // names and wins found
        for(int i = 0; i < names.size(); i++) {
            players.push_back(Player(names[i], wins[i]));
        }

    }

    myfile.close();

    // Player creation loop if no players
    // were found in the save file
    // or the players chooses to make new players
    while(true) {
        if(players.size() == 0) {
            // No players were loaded from the save file
            string name;
            cout << "Enter Player's 1 name: ";

```

```

getline(cin, name);
game->players.push_back(Player(name,0));
cout << "Enter Player's 2 name: ";
getline(cin, name);
game->players.push_back(Player(name,0));
} else {
    // Uses the comp function to sort the
    // players by their win count
    sort(players.begin(), players.end(), comp);
    // game->players = list<Player>(players.begin(), players.end());
    int i = 0;
    // prints out all the players and assigns a number to them
    cout << "Leaderboards and selection" << endl;
    for(auto player: players) {
        cout << "-----" << endl;
        cout << "(" << i + 1 << ") "<<"Player: " << player.getName() << " Wins: " << player.getWins() << endl;
        cout << "-----" << endl;
        i++;
    }

    cout << "Would you like to create new players? (y/n)" << endl;
    string option;
    getline(cin, option);
    if(option == "y") {
        // Players get to choose their player save
        string name;
        cout << "Enter Player's 1 name: ";
        getline(cin, name);
        game->players.push_back(Player(name));
        cout << "Enter Player's 2 name: ";
        getline(cin, name);
        game->players.push_back(Player(name));
    } else {
        // Players get to make new Players to save

        cout << "Player 1 select your profile: ";
        getline(cin, option);

        Player player = *next(players.begin() , stoi(option) -1);

        game->players.push_back(player);

        cout << "Player 2 select your profile: ";
        getline(cin, option);

        player = *next(players.begin() , stoi(option) -1);

        game->players.push_back(player);
    }
}
game->gameState.push(new PlayState(game));
game->gameState.top()->run();
cout << "Play again? (y/n): ";

```

```

    string option;
    getline(cin, option);
    if(option == "n") {
        break;
    }
}
game->gameState.pop();
}

PlayState::PlayState(Game* game) {
    this->game = game;
}

// PlayState is responsible
// for all the game logic
void PlayState::run() {
    // Creating board
    Board board(game->players);
    int rounds = 1;
    const int MAX_ROUNDS = 32;
    cout<<"Playing..."<<endl;
    bool running = true;
    while(running) {
        queue<Player*> players;
        // Players go through a queue based
        // turn by turn
        for(auto& player: game->players) {
            players.push(&player);
        }

        while(!players.empty()) {
            auto player = players.front();
            if(rounds > MAX_ROUNDS) {
                cout << "No one won ... \n";
                break;
            }
            board.printBoard();
            cout << player->getName() <<"\s turn!" <<endl;
            cout << "Enter a number from 0-7\n";
            string option;
            getline(cin,option);
            int opt = stoi(option);
            board.mark(player->getName(), opt);
            char c = board.win();
            if(c != '#') {
                board.printBoard();
                cout << player->getName() << " WINS!" << endl;
                player->setWins(player->getWins() + 1);
                running = false;
                break;
            }
            rounds++;
            player = players.front();

```

```

    players.pop();
}
}

// Makes a copy of the save file
// In order to update each player's
// win counter
fstream myfile ("players.txt", std::fstream::in | std::fstream::out);
if(myfile.is_open()) {
    deque<Player> temp;
    string line;
    int i = 0;

    deque<string> names;
    deque<int> wins;

    // Goes through the file
    // and searches for names and wins
    while(getline(myfile, line)) {
        if(line.empty()) continue;
        if(i % 2 == 0) {
            names.push_back(line);
        } else {
            wins.push_back(stoi(line));
        }
        i++;
    }

    for(int i = 0; i < names.size(); i++) {
        temp.push_back(Player(names[i], wins[i]));
    }
    myfile.close();

    // Checks if any of the current players
    // need to update their
    // scores
    for(auto player : game->players) {
        bool found = false;
        for(int i = 0; i < names.size(); i++) {
            if(temp[i].getName() == player.getName()) {
                temp[i].setWins(player.getWins());
                found = true;
                break;
            }
        }
        // If the player was not found in the copy of
        // the save file then they are a new player
        // and need to be added in the save file
        if(!found) {
            Player nplayer(player.getName(), player.getWins());
            temp.push_back(nplayer);
        }
    }
}

```

```

    }

    myfile.close();
    // Close the file and open it in output mode
    fstream myfile2 ("players.txt", std::fstream::in | std::fstream::out);
    // Makes sure the file is not empty for formatting issues

    if(myfile2.peek() == std::ifstream::traits_type::eof()) {
        bool first = true;

        for(auto player : temp) {
            if(!first) {
                myfile2 << "\n";
            }
            first = false;
            myfile2 << player.data();
        }
    } else {
        for(auto& player : temp) {
            myfile2 << "\n";
            myfile2 << player.data();
        }
    }

    myfile2.close();
}
// clears all the players from the list
// in order to not have duplicate players
// in the save file
game->players.clear();
game->gameState.pop();
}

/*
 * File: Player.h
 * Author: Michael Cooper
 * Purpose: Player Header file
 * Created on 4/28/22
 */

#ifndef PLAYER_H
#define PLAYER_H

#include <string>

class Player {
private:
    std::string name;
    int wins;
public:
    Player(std::string name);
    Player(std::string name, int wins);

```

```

    Player(const Player& player);

    std::string getName() const;
    int getWins() const;

    void setName(std::string name);
    void setWins(int wins);

    std::string data();

    int getInput();
};

#endif // PLAYER_H

/*
 * File: Player.cpp
 * Author: Michael Cooper
 * Purpose: Player Source File
 * Created on 4/28/22
 */

#include "Player.h"
#include <iostream>

using namespace std;

// Constructors
Player::Player(string name) {
    this->name = name;
    wins = 0;
}

Player::Player(string name, int wins) {
    this->name = name;
    this->wins = wins;
}

Player::Player(const Player& player) {
    name = player.getName();
    wins = player.getWins();
}

// Getters
std::string Player::getName() const {
    return name;
}

int Player::getWins() const {
    return wins;
}

// Setters

```

```

void Player::setName(std::string name) {
    this->name = name;
}

void Player::setWins(int wins) {
    this->wins = wins;
}

// Used for saving player
// data into the save file
string Player::data() {
    return name + "\n" + to_string(wins);
}

// Used to check player input
// for marking the board
int Player::getInput() {
    while(true) {
        cout << getName() << "'s turn!" << endl;
        cout << "Enter a number from 0-7\n";
        string option;
        getline(cin, option);
        int opt = stoi(option);
        if(opt < 0 && opt > 7) {
            cout << "That is not a valid option!" << endl;
            cout << "Option has to be between 0 and 7" << endl;
            continue;
        }
        return opt;
    }
}

```