CSCI 1300 Recitation HW 10

Board.cpp

```
#include "Board.h"
#define RED "\033[48;2;230;10;10m"
#define GREEN "\033[48;2;34;139;34m"  /* Grassy Green (34,139,34) */
#define BLUE "\033[48;2;10;10;230m"
#define PINK "\033[48;2;255;105;180m"
#define BROWN "\033[48;2;139;69;19m"
#define PURPLE "\033[48;2;128;0;128m"
#define ORANGE "\033[48;2;230;115;0m" /* Orange (230,115,0) */
#define GREY "\033[48;2;128;128;128m" /* Grey (128,128,128) */
#define RESET "\033[0m"

void Board::initializeBoard()
{
    // Seed random number generator in your main function once
    for (int i = 0; i < 2; i++)
    {
        initializeTiles(i);  // This ensures each lane has a unique tile distribution
    }
}

#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()
#include <iostream>

using namespace std;

void Board::initializeTiles(int player_index)
{
    Tile temp;
    int green_count = 0;
    int total_tiles = _BOARD_SIZE;


    srand(time(0));

    // Keep track of green tile positions to ensure we place exactly 30 greens
    for (int i = 0; i < total_tiles; i++)
    {

        if (i == total_tiles - 1) {
            // Set the last tile as Orange for "Pride Rock"
            temp.color = 'O';
        }
        else if (i == 0) {
            // Set the last tile as Orange for "Pride Rock"
            temp.color = 'Y';
        }
        else if (green_count < 30 && (rand() % (total_tiles - i) < 30 - green_count)) {
            temp.color = 'G';
            green_count++;
        }
        else
        {
            // Randomly assign one of the other colors: Blue, Pink, Brown, Red, Purple
            int color_choice = (rand()*(player_index+1)*13)%7;
            if(green_count < 15){
                switch (color_choice)
                {
                    case 0:
                        temp.color = 'B'; // Blue
                        break;
                    case 1:
                        temp.color = 'P'; // Pink
                        break;
                    case 2:
                        temp.color = 'N'; // Brown
                        break;
                    case 3:
                        temp.color = 'N'; // Brown
                        break;
                    case 4:
                        temp.color = 'R'; // Red
                        break;
                    case 5:
                        temp.color = 'R'; // Red
                        break;
                    case 6:
                        temp.color = 'U'; // Purple
                        break;
                }
            }
            else{
                switch (color_choice)
                {
                case 0:
                        temp.color = 'B'; // Blue
                        break;
                    case 1:
                        temp.color = 'P'; // Pink
                        break;
                    case 2:
                        temp.color = 'B'; // Brown
                        break;
                    case 3:
                        temp.color = 'N'; // Brown
                        break;
                    case 4:
                        temp.color = 'R'; // Red
                        break;
                    case 5:
                        temp.color = 'U'; // Red
                        break;
                    case 6:
```

```
                        temp.color = 'U'; // Purple
                        break;
                }
            }
        }

        // Assign the tile to the board for the specified lane
        _tiles[player_index][i] = temp;
    }
}

Board::Board()
{
   _player_count = 1;
   // Initialize player position
   _player_position[0] = 0;

   // Initialize tiles
   initializeTiles(1); //temp
}

Board::Board(int player_count)
{
    if (player_count > _MAX_PLAYERS)
    {
        _player_count = _MAX_PLAYERS;
    }
    else
    {
        _player_count = player_count;
    }

    // Initialize player position
    for (int i = 0; i < _player_count; i++)
    {
        _player_position[i] = 0;
    }

    // Initialize tiles

    initializeBoard();
}

bool Board::isPlayerOnTile(int player_index, int pos)
{
    if (_player_position[player_index] == pos)
    {
        return true;
    }
    return false;
}

void Board::displayTile(int player_index, int pos)
{
    // string space = "                                       ";
    string color = "";
    int player = isPlayerOnTile(player_index, pos);

    // Template for displaying a tile: <line filler space> <color start> |<player symbol or blank spa

    // Determine color to display

    if (_tiles[player_index][pos].color == 'R')
    {
        color = RED;
    }
    else if (_tiles[player_index][pos].color == 'G')
    {
        color = GREEN;
    }
    else if (_tiles[player_index][pos].color == 'B')
    {
        color = BLUE;
    }
    else if (_tiles[player_index][pos].color == 'U')
    {
        color = PURPLE;
    }
    else if (_tiles[player_index][pos].color == 'N')
    {
        color = BROWN;
    }
    else if (_tiles[player_index][pos].color == 'P')
    {
        color = PINK;
    }
    else if (_tiles[player_index][pos].color == 'O')
    {
        color = ORANGE;
    }
    else if (_tiles[player_index][pos].color == 'Y')
    {
        color = GREY;
    }
     if (player == true)
    {
        cout << color << "|" << (player_index + 1) << "|" << RESET;
    }
    else
    {
        cout << color << "| |" << RESET;
    }
}

void Board::displayTrack(int player_index)
{
    for (int i = 0; i < _BOARD_SIZE; i++)
    {
        displayTile(player_index, i);
    }
    cout << endl;
}

void Board::displayBoard()
{
    for (int i = 0; i < 2; i++)
    {
        displayTrack(i);
        if (i == 0) {
            cout << endl;  // Add an extra line between the two lanes
        }
    }
}

bool Board::movePlayer(int player_index)
{
    // Increment player position
    _player_position[player_index]++;
    if (_player_position[player_index] == _BOARD_SIZE - 1){
// Player reached last tile
        return true;
    }
    return false;
}

int Board::getPlayerPosition(int player_index) const
{
    if (player_index >= 0 && player_index <= _player_count)
    {
        return _player_position[player_index];
    }
   return -1;
}

char Board::getTile(int player_index, int pos) const{
    return _tiles[player_index][pos].color;
}
```

board.h

```
#ifndef BOARD_H
#define BOARD_H
#include "Tile.h"

class Board
{
private:
    static const int _BOARD_SIZE = 52;
    Tile _tiles[2][_BOARD_SIZE];
    static const int _MAX_PLAYERS = 2;
    int _player_count;
    int _player_position[_MAX_PLAYERS];
    void displayTile(int player_index, int pos);
    void initializeTiles(int player_index);
    bool isPlayerOnTile(int player_index, int pos);

public:
    Board();
    Board(int player_count);
    void displayTrack(int player_index);
    void initializeBoard();
    void displayBoard();
    bool movePlayer(int player_index);
    int getPlayerPosition(int player_index) const;
    char getTile(int player_index, int pos) const;
};

#endif
```

tile.h

```
struct Tile
{
    char color;
};
```

recitation13.cpp

```
#include "Board.h"
#include <iostream>

using namespace std;

void PrintMenu(){

    cout << "Main Menu: Select an option to continue" << endl;

    cout << "1. Check Player Progress (1)" << endl;

    cout << "2. Review Character (2)" << endl;

    cout << "3. Check Position (3)" << endl;

    cout << "4. Review your Advisor (4)" << endl;
    cout << "5. Move Forward (5)" << endl;

    cout << "Please choose an option using the corresponding number:" << endl;
}

int main(){
    Board board(2);
    char input;
    bool playing = true;
    bool isTurn = true;
    char currentTileColor= ' ';

    for (int i = 0; i < 2; i++){
        board.initializeBoard();
    }
    board.displayBoard();

    while(playing){
        for(int i = 0; i < 2; i++){
            isTurn = true;
            while(isTurn){
                cout << "player " << i+1 << " s turn" << endl;
                PrintMenu();
                cin >> input;

                switch (input)
                {
                case '1':
                    /* code */
                    break;
                case '2':
                    /* code */
                    break;
                case '3':
                    board.displayBoard();
                    break;
                case '4':
                    /* code */
                    break;
                case '5':
                    board.movePlayer(i);
                    isTurn = false;
                    break;

                default:
                    playing = false;
                    isTurn = false;
                    break;
                }
                board.displayBoard();
                currentTileColor = board.getTile(i, board.getPlayerPosition(i));

                switch (currentTileColor)
                {
                case 'B':
                    cout << "on a blue tile" << endl;
                    break;
                case 'P':
                    cout << "on a pink tile" << endl;
                    break;
                case 'N':
                    cout << "on a brown tile" << endl;
                    break;
                case 'R':
                    cout << "on a red tile" << endl;
                    break;
                case 'U':
                    cout << "on a purple tile" << endl;
                    break;
                case 'G':
                    cout << "on a green tile" << endl;
                    break;
                default:
                    break;
                }
                cout << endl;
            }
        }
    }
}
```