***OS Lab final project***

***Game***

***Potion Explosion***

***Basic Implementation oop based***

***Os concepts***

***Threading***

***Synchoronization***

***Scheduling***

***Multi Player***

Source code:

#include<iostream>

#include<random>

#include<algorithm> // for std::swap

#include<string>

#include <cstring> // Add this line

#include<mutex>

#include<thread>

using namespace std;

void player\_disp(char p[]);

char p1[25];

char p2[25];

int p1\_c = 0;

int p2\_c = 0;

int score1 = 0, score2 = 0;

int Healing\_score = 2;

int Courage\_score = 2;

int invisibility\_score = 2;

int strength\_score = 2;

string name1 = "Healing", name2 = "Courage", name3 = "invisibility", name4 = "strength";

mutex mtx;

void player\_disp(char p[]);

int random\_no();

class Potion {

public:

char Healing[2];

char Courage[2];

char invisibility[2];

char strength[2];

mutex potionMutex;

Potion() {

for (int i = 0; i < 2; ++i)

Healing[i] = ' ';

for (int i = 0; i < 2; ++i)

Courage[i] = ' ';

for (int i = 0; i < 2; ++i)

invisibility[i] = ' ';

for (int i = 0; i < 2; ++i)

strength[i] = ' ';

}

// Modify the disp\_potion function to accept the size of the array

void disp\_potion(char arr[], int size) {

lock\_guard<mutex> lock(potionMutex);

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

cout << arr[i] << " ";

}

}

// Inside the Potion class, update the call to disp\_potion

void disp\_potion(char arr[]) {

int size = 0;

if (strcmp(arr, Healing) == 0) {

size = sizeof(Healing) / sizeof(Healing[0]);

}

else if (arr == Courage) {

size = sizeof(Courage) / sizeof(Courage[0]);

}

else if (arr == invisibility) {

size = sizeof(invisibility) / sizeof(invisibility[0]);

}

else if (arr == strength) {

size = sizeof(strength) / sizeof(strength[0]);

}

disp\_potion(arr, size);

}

void calculate\_score(const char\* potion\_name, int player\_number) {

lock\_guard<mutex> lock(potionMutex);

if (player\_number == 1) {

if (strcmp(potion\_name, "Healing") == 0) {

score1 += Healing\_score;

}

else if (strcmp(potion\_name, "Courage") == 0) {

score1 += Courage\_score;

}

}

else if (player\_number == 2) {

if (strcmp(potion\_name, "invisibility") == 0) {

score2 += invisibility\_score;

}

else if (strcmp(potion\_name, "strength") == 0) {

score2 += strength\_score;

}

}

}

bool isFull() {

lock\_guard<mutex> lock(potionMutex);

if (strcmp(Healing, " ") == 0 && strcmp(Courage, " ") == 0) {

return true; // Both Healing and Courage are full

}

else if (strcmp(invisibility, " ") == 0 && strcmp(strength, " ") == 0) {

return true; // Both Invisibility and Strength are full

}

else {

return false; // Neither combination is full

}

}

};

class Gameboard {

public:

Potion potion1;

Potion potion2;

// Store which player has which potion

int player1\_potion\_index1;

int player1\_potion\_index2;

int player2\_potion\_index1;

int player2\_potion\_index2;

Gameboard() : player1\_potion\_index1(0), player1\_potion\_index2(0),

player2\_potion\_index1(0), player2\_potion\_index2(0) {}

void add\_marble\_to\_potion(char p[], int& player\_count, int player\_number) {

int choice;

char\* selected\_potion = nullptr;

int\* selected\_index = nullptr;

Potion\* current\_potion = nullptr;

if (player\_number == 1) {

std::cout << "Player " << player\_number << ", select a potion to add your marble to:" << std::endl;

std::cout << "1. Healing" << std::endl;

std::cout << "2. Courage" << std::endl;

std::cin >> choice;

if (choice == 1) {

selected\_index = &player1\_potion\_index1;

current\_potion = &potion1;

selected\_potion = potion1.Healing;

}

else if (choice == 2) {

selected\_index = &player1\_potion\_index2;

current\_potion = &potion1;

selected\_potion = potion1.Courage;

}

else {

std::cout << "Invalid choice. Marble not added to any potion." << std::endl;

return;

}

}

else if (player\_number == 2) {

std::cout << "Player " << player\_number << ", select a potion to add your marble to:" << std::endl;

std::cout << "1. Invisibility" << std::endl;

std::cout << "2. Strength" << std::endl;

std::cin >> choice;

if (choice == 1) {

selected\_index = &player2\_potion\_index1;

current\_potion = &potion2;

selected\_potion = potion2.invisibility;

}

else if (choice == 2) {

selected\_index = &player2\_potion\_index2;

current\_potion = &potion2;

selected\_potion = potion2.strength;

}

else {

std::cout << "Invalid choice. Marble not added to any potion." << std::endl;

return;

}

}

else {

std::cout << "Invalid player number. Marble not added to any potion." << std::endl;

return;

}

if (\*selected\_index < sizeof(selected\_potion) / sizeof(selected\_potion[0])) {

// Check if the potion is not full

if (\*selected\_index < sizeof(selected\_potion) / sizeof(selected\_potion[0])) {

// Add the marble to the selected potion

selected\_potion[\*selected\_index] = p[0];

// Increment the index for the selected potion

++(\*selected\_index);

// Remove the marble from the player's container

for (int i = 0; i < player\_count - 1; ++i) {

p[i] = p[i + 1];

}

// Decrement the count to reflect the removal

--player\_count;

std::cout << "Marble added to the selected potion successfully!" << std::endl;

// Display the updated potion

std::cout << "Updated " << ((player\_number == 1) ? "Healing" : "Invisibility") << " potion: ";

current\_potion->disp\_potion(selected\_potion, \*selected\_index);

std::cout << std::endl;

}

// Check if the potion is full

if (\*selected\_index == sizeof(selected\_potion) / sizeof(selected\_potion[0])) {

std::cout << "Potion is full!" << std::endl;

// Call calculate\_score to update player's score

current\_potion->calculate\_score((choice == 1) ? "Healing" : "Strength", player\_number);

// Reset the completed potion

reset\_potion(selected\_potion, selected\_index);

// Return when the potion is full

return;

}

}

else {

std::cout << "Selected potion is full. Marble not added." << std::endl;

}

}

void reset\_potion(char arr[], int\* index) {

std::memset(arr, ' ', \*index);

\*index = 0;

}

};

int random\_no() {

std::random\_device rd;

std::mt19937 eng(rd());

std::uniform\_int\_distribution<> distr(0, 4);

return distr(eng);

}

struct Marble {

char color[5] = { 'R', 'Y', 'B', 'G', 'W' };

};

class Dispenser {

char grid[5][5] = { " " };

Marble m;

char helper = ' ';

mutex dispenserMutex; // Mutex to synchronize dispenser operations

public:

void init\_grid() {

for (int i = 0; i < 5; ++i) {

for (int j = 0; j < 5; ++j) {

grid[i][j] = m.color[random\_no()];

}

}

}

void pick\_marble(char p[], int\* current\_player\_count, int current\_players) {

int row, col;

char pick\_color;

lock\_guard<mutex> lock(dispenserMutex); // Lock the dispenser operations

cout << "Enter row (-1 to exit): ";

cin >> row;

if (row == -1) {

exit(0);

}

cout << "Enter col: ";

cin >> col;

pick\_color = grid[row][col];

helper = pick\_color;

grid[row][col] = 'x';

pick\_consecutive\_marbles(row, col, pick\_color, p, current\_player\_count, current\_players);

disp\_grid();

}

char\* save\_to\_container(char p[], int i) {

p[i] = helper;

cout << "Checking from save func: " << endl;

return p;

}

void pick\_consecutive\_marbles(int row, int col, char color, char\* current\_player\_container, int\* current\_player\_count, int current\_player) {

char pickedColor = color;

// Check below the picked marble

for (int i = row + 1; i < 5 && grid[i][col] == pickedColor; ++i) {

grid[i][col] = 'x';

current\_player\_container[\*current\_player\_count] = pickedColor;

++(\*current\_player\_count);

}

// Check above the picked marble

for (int i = row - 1; i >= 0 && grid[i][col] == pickedColor; --i) {

grid[i][col] = 'x';

current\_player\_container[\*current\_player\_count] = pickedColor;

++(\*current\_player\_count);

}

// Check if both above and below marbles have been picked

int aboveRow = row - 1;

int belowRow = row + 1;

while (aboveRow >= 0 && grid[aboveRow][col] == pickedColor) {

cout << "Picked marble with color: " << pickedColor << " at position (" << aboveRow << ", " << col << ")" << endl;

grid[aboveRow][col] = 'x';

current\_player\_container[\*current\_player\_count] = pickedColor;

++(\*current\_player\_count);

--aboveRow;

}

while (belowRow < 5 && grid[belowRow][col] == pickedColor) {

cout << "Picked marble with color: " << pickedColor << " at position (" << belowRow << ", " << col << ")" << endl;

grid[belowRow][col] = 'x';

current\_player\_container[\*current\_player\_count] = pickedColor;

++(\*current\_player\_count);

++belowRow;

}

// After placing 'x', check the grid to slide the 'x' upwards

check\_grid(col);

// Print the picked marbles in the container of the current player

cout << "\nIn player " << current\_player << " container: " << endl;

player\_disp(current\_player\_container);

}

void check\_grid(int j) {

for (int i = 0; i < 5; ++i) {

if (grid[i][j] == 'x') {

slide\_marble(i, j);

}

}

}

void slide\_marble(int i, int j) {

while (i > 0 && grid[i][j] == 'x') {

swap(grid[i - 1][j], grid[i][j]);

--i;

}

}

void disp\_grid() {

cout << " ";

for (int j = 0; j < 5; ++j) {

cout << j << " ";

}

cout << endl;

for (int i = 0; i < 5; ++i) {

cout << i << " ";

for (int j = 0; j < 5; ++j) {

cout << grid[i][j] << " ";

}

cout << endl;

}

}

void return\_to\_dispenser(char p[], int& count) {

mutex dispenserMutex; // Mutex to synchronize dispenser operations

cout << "Returning marbles to dispenser..." << endl;

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

char returnedMarble = p[i];

int row = 0, col = 0;

// Find a position marked with 'x' in the dispenser

do {

row = random\_no();

col = random\_no();

} while (grid[row][col] != 'x');

// Place the returned marble in the dispenser

grid[row][col] = returnedMarble;

}

// Reset the player's container

count = 0;

}

char pick\_from\_container(char p[], int& count) {

lock\_guard<mutex> lock(dispenserMutex); // Lock the dispenser operations

if (count == 0) {

cout << "No marbles to pick. Container is empty." << endl;

return '\0';

}

int index;

cout << "Enter the index to pick a marble (0 to " << count - 1 << "): ";

cin >> index;

if (index < 0 || index >= count) {

cout << "Invalid index. Please enter a valid index." << endl;

return '\0';

}

char pickedMarble = p[index];

cout << "Picked marble from index " << index << ": " << pickedMarble << endl;

// Remove the picked marble from the container

for (int i = index; i < count - 1; ++i) {

p[i] = p[i + 1];

}

// Decrement the count to reflect the removal

--count;

return pickedMarble;

}

char pick\_from\_index(char p[], int& count, int index) {

if (count == 0) {

cout << "No marbles to pick. Container is empty." << endl;

return '\0';

}

if (index < 0 || index >= count) {

cout << "Invalid index. Please enter a valid index." << endl;

return '\0';

}

char pickedMarble = p[index];

cout << "Picked marble from index " << index << ": " << pickedMarble << endl;

// Remove the picked marble from the container

for (int i = index; i < count - 1; ++i) {

p[i] = p[i + 1];

}

// Decrement the count to reflect the removal

--count;

return pickedMarble;

}

};

void player1(Dispenser\* obj, Gameboard& gameboard) {

std::lock\_guard<std::mutex> lock(mtx);

cout << "Player 1 turn: " << endl;

obj->pick\_marble(p1, &p1\_c, 1); // Pass the address of p1\_c

obj->save\_to\_container(p1, p1\_c);

++p1\_c;

}

void player2(Dispenser\* obj, Gameboard& gameboard) {

std::lock\_guard<std::mutex> lock(mtx);

cout << "Player 2 turn: " << endl;

obj->pick\_marble(p2, &p2\_c, 2); // Pass the address of p2\_c

obj->save\_to\_container(p2, p2\_c);

++p2\_c;

}

void player\_disp(char p[]) {

for (int i = 0; p[i] != '\0'; ++i) {

cout << p[i] << ' ';

}

cout << endl;

}

void display\_player\_potions(Gameboard& gameboard) {

cout << "Player 1 has potions: " << name1 << ": " << gameboard.potion1.Healing[gameboard.player1\_potion\_index1]

<< ", " << name2 << ": " << gameboard.potion1.Courage[gameboard.player1\_potion\_index2] << endl;

cout << "Player 2 has potions: " << name1 << ": " << gameboard.potion2.Healing[gameboard.player2\_potion\_index1]

<< ", " << name2 << ": " << gameboard.potion2.Courage[gameboard.player2\_potion\_index2] << endl;

}

void player\_turn(Dispenser\* obj, Gameboard& gameboard) {

int ex = 0;

int current\_player = 1; // Start with Player 1

while (ex != -1) {

cout << "Press -1 to exit, 0 to continue, 1 to return marbles, 2 to pick from container, 3 pick from a specific index in the player's container (p1 or p2): ";

cin >> ex;

if (ex == -1) {

exit(0);

}

else if (ex == 1) {

cout << "Choose player (1 or 2) to return marbles: ";

int player;

cin >> player;

if (player == 1) {

obj->return\_to\_dispenser(p1, p1\_c);

}

else if (player == 2) {

obj->return\_to\_dispenser(p2, p2\_c);

}

}

else if (ex == 2) {

cout << "Choose player (1 or 2) to pick from container: ";

int player = 0;

cin >> player;

char marble = ' ';

if (player == 1) {

marble = obj->pick\_from\_container(p1, p1\_c);

}

else if (player == 2) {

marble = obj->pick\_from\_container(p2, p2\_c);

}

// Add the picked marble to the potion

gameboard.add\_marble\_to\_potion(&marble, (player == 1) ? p1\_c : p2\_c, player);

}

else if (ex == 3) {

cout << "Choose player (1 or 2) and enter index to pick from container: ";

int player, index;

cin >> player >> index;

char marble = ' ';

if (player == 1) {

marble = obj->pick\_from\_index(p1, p1\_c, index);

}

else if (player == 2) {

marble = obj->pick\_from\_index(p2, p2\_c, index);

}

// Add the picked marble to the potion

gameboard.add\_marble\_to\_potion(&marble, (player == 1) ? p1\_c : p2\_c, player);

}

obj->disp\_grid();

if (current\_player == 1 && gameboard.potion1.isFull()) {

cout << "Player 1 wins! Potion is full." << endl;

exit(0); // Terminate the game

}

else if (current\_player == 2 && gameboard.potion2.isFull()) {

cout << "Player 2 wins! Potion is full." << endl;

exit(0); // Terminate the game

}

if (current\_player == 1) {

std::thread t1(player1, obj, std::ref(gameboard));

t1.join();

cout << "\nIn player 1 grid: " << endl;

player\_disp(p1);

current\_player = 2; // Switch to Player 2

}

else {

std::thread t2(player2, obj, std::ref(gameboard));

t2.join();

cout << "\nIn player 2 grid: " << endl;

player\_disp(p2);

current\_player = 1; // Switch to Player 1

}

}

}

int main() {

Dispenser obj;

obj.init\_grid();

Gameboard gameboard;

display\_player\_potions(gameboard);

player\_turn(&obj, gameboard);

cout << "\nplayer 1 grid: " << endl;

player\_disp(p1);

cout << "\nplayer 2 grid: " << endl;

player\_disp(p2);

return 0;

}