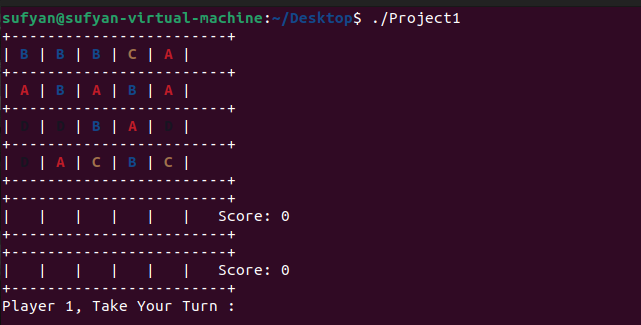
**Interface:**

****

**Code:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <pthread.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/shm.h>

using namespace std;

// Constants

const int BOARD\_ROWS = 4;

const int BOARD\_COLS = 5;

const int NUM\_PLAYERS = 2;

const int PLAYER\_COLS = 5;

// Shared Memory IDs

int shmid\_gameBoard;

int shmid\_playerBoard[NUM\_PLAYERS];

int shmid\_playerScores;

// Mutexes

pthread\_mutex\_t mutex\_gameBoard = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mutex\_playerBoard[NUM\_PLAYERS] = {PTHREAD\_MUTEX\_INITIALIZER, PTHREAD\_MUTEX\_INITIALIZER};

// Define the game state

enum GameState

{

INITIALIZING,

RUNNING,

FINISHED

};

GameState gameStatus = INITIALIZING;

// Define the player colors

const char PLAYER\_COLORS[NUM\_PLAYERS] = {'R', 'B'};

// Define the potion ingredients

const char RED = 'R';

const char YELLOW = 'Y';

const char BLUE = 'B';

const char BLACK = 'K';

// Define the game board structures

char \*gameBoard;

char \*playerBoard[NUM\_PLAYERS];

int \*playerScores;

// Forward declarations

void assessReaction(int player);

int getPlayerMove(int player);

void showGameBoard();

void displayPlayerInfo();

void executeMove(int move, int player);

bool hasGameFinished();

void announceChampion();

void \*gameLoop(void \*arg);

// Function to create shared memory for the game board

template <typename T>

T \*createSharedMemory(int &shmid, int size)

{

shmid = shmget(IPC\_PRIVATE, sizeof(T) \* size, IPC\_CREAT | 0666);

T \*memory = (T \*)shmat(shmid, NULL, 0);

return memory;

}

// Function to initialize shared memory

void initializeSharedMemory()

{

srand(time(NULL));

gameBoard = createSharedMemory<char>(shmid\_gameBoard, BOARD\_ROWS \* BOARD\_COLS);

for (int i = 0; i < NUM\_PLAYERS; ++i)

{

playerBoard[i] = createSharedMemory<char>(shmid\_playerBoard[i], PLAYER\_COLS);

}

playerScores = createSharedMemory<int>(shmid\_playerScores, NUM\_PLAYERS);

for (int i = 0; i < BOARD\_ROWS \* BOARD\_COLS; ++i)

{

gameBoard[i] = rand() % 4 + 'A';

}

for (int i = 0; i < NUM\_PLAYERS; ++i)

{

for (int j = 0; j < PLAYER\_COLS; ++j)

{

playerBoard[i][j] = ' ';

}

}

for (int i = 0; i < NUM\_PLAYERS; ++i)

{

playerScores[i] = 0;

}

}

// Function to display the game board

void showGameBoard()

{

pthread\_mutex\_lock(&mutex\_gameBoard);

// Print top border

cout << "+";

for (int i = 0; i < BOARD\_COLS \* 5 - 1; ++i)

{

cout << "-";

}

cout << "+" << endl;

for (int i = 0; i < BOARD\_ROWS; i++)

{

cout << "| ";

for (int j = 0; j < BOARD\_COLS; j++)

{

char ingredient = gameBoard[i \* BOARD\_COLS + j];

switch (ingredient)

{

case 'A':

cout << "\033[1;31m" << ingredient << "\033[0m | "; // Red

break;

case 'B':

cout << "\033[1;34m" << ingredient << "\033[0m | "; // Blue

break;

case 'C':

cout << "\033[1;33m" << ingredient << "\033[0m | "; // Yellow

break;

case 'D':

cout << "\033[1;30m" << ingredient << "\033[0m | "; // Black

break;

default:

cout << ingredient << " | ";

}

}

cout << endl;

cout << "+";

for (int i = 0; i < BOARD\_COLS \* 5 - 1; ++i)

{

cout << "-";

}

cout << "+" << endl;

}

pthread\_mutex\_unlock(&mutex\_gameBoard);

}

// Function to display player boards and scores

void displayPlayerInfo()

{

for (int player = 0; player < NUM\_PLAYERS; player++)

{

pthread\_mutex\_lock(&mutex\_playerBoard[player]);

// Print top border

cout << "+";

for (int i = 0; i < PLAYER\_COLS \* 5 - 1; ++i)

{

cout << "-";

}

cout << "+" << endl;

cout << "| ";

for (int j = 0; j < PLAYER\_COLS; j++)

{

char ingredient = playerBoard[player][j];

switch (ingredient)

{

case 'A':

cout << "\033[1;31m" << ingredient << "\033[0m | "; // Red

break;

case 'B':

cout << "\033[1;34m" << ingredient << "\033[0m | "; // Blue

break;

case 'C':

cout << "\033[1;33m" << ingredient << "\033[0m | "; // Yellow

break;

case 'D':

cout << "\033[1;30m" << ingredient << "\033[0m | "; // Black

break;

default:

cout << ingredient << " | ";

}

}

cout << " Score: " << playerScores[player] << " ";

cout << endl;

cout << "+";

for (int i = 0; i < PLAYER\_COLS \* 5 - 1; ++i)

{

cout << "-";

}

cout << "+" << endl;

pthread\_mutex\_unlock(&mutex\_playerBoard[player]);

}

}

// Function to get the player's move

int getPlayerMove(int player)

{

int move;

while (true)

{

cout << "Player " << (player + 1) << ", Take Your Turn : ";

cin >> move;

if (move >= 1 && move <= PLAYER\_COLS)

{

return move;

}

else

{

cout << "Please enter a number between 1 and 5." << endl;

}

}

}

// Function to process the player's move

void executeMove(int move, int player)

{

int col = move - 1;

if (gameBoard[col] != ' ')

{

char ingredient = gameBoard[col];

for (int i = 0; i < BOARD\_ROWS - 1; i++)

{

gameBoard[i \* BOARD\_COLS + col] = gameBoard[(i + 1) \* BOARD\_COLS + col];

}

gameBoard[(BOARD\_ROWS - 1) \* BOARD\_COLS + col] = ' ';

for (int i = 0; i < PLAYER\_COLS; i++)

{

if (playerBoard[player][i] == ' ')

{

playerBoard[player][i] = ingredient;

break;

}

}

assessReaction(player);

}

else

{

cout << "Invalid move. Please choose a column with marbles." << endl;

}

}

// Function to check if a move caused a reaction

void assessReaction(int player)

{

int count = 0;

char lastIngredient = ' ';

for (int i = 0; i < PLAYER\_COLS; i++)

{

if (playerBoard[player][i] == ' ')

{

break;

}

if (playerBoard[player][i] == lastIngredient)

{

count++;

}

else

{

count = 1;

lastIngredient = playerBoard[player][i];

}

if (count >= 3)

{

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

{

playerBoard[player][j] = ' ';

}

char ingredient = playerBoard[player][i];

// Update the player's score

if (ingredient == RED)

{

playerScores[player] += 2;

}

else if (ingredient == YELLOW)

{

playerScores[player] += 3;

}

else if (ingredient == BLUE)

{

playerScores[player] += 4;

}

else

{

playerScores[player] += 5;

}

assessReaction(player);

break; // Exit the loop after finding and processing one reaction

}

}

}

// Function to check if the game is over

bool hasGameFinished()

{

for (int j = 0; j < BOARD\_COLS; j++)

{

if (gameBoard[j] != ' ')

{

return false;

}

}

return true;

}

// Function to declare the winner and display scores

void announceChampion()

{

displayPlayerInfo();

if (playerScores[0] > playerScores[1])

{

cout << "Player 1 WON" << endl;

}

else if (playerScores[1] > playerScores[0])

{

cout << "Player 2 WON" << endl;

}

else

{

cout << " TIE " << endl;

}

}

// Function for the game loop in a separate thread

void \*gameLoop(void \*arg)

{

while (!hasGameFinished())

{

showGameBoard();

displayPlayerInfo();

int move1 = getPlayerMove(0);

executeMove(move1, 0);

showGameBoard();

displayPlayerInfo();

if (hasGameFinished())

{

break;

}

int move2 = getPlayerMove(1);

executeMove(move2, 1);

}

announceChampion();

shmdt(gameBoard);

for (int i = 0; i < NUM\_PLAYERS; ++i)

{

shmdt(playerBoard[i]);

}

shmdt(playerScores);

shmctl(shmid\_gameBoard, IPC\_RMID, NULL);

for (int i = 0; i < NUM\_PLAYERS; ++i)

{

shmctl(shmid\_playerBoard[i], IPC\_RMID, NULL);

}

shmctl(shmid\_playerScores, IPC\_RMID, NULL);

pthread\_exit(NULL);

}

int main()

{

initializeSharedMemory();

pthread\_t thread;

pthread\_create(&thread, NULL, gameLoop, NULL);

pthread\_join(thread, NULL);

return 0;

}