ПОЯСНЕНИЕ КОДА

**Мини-игра №3: «Крестики-нолики»**

1. **Описание программы**

Эта программа реализует консольную игру «Крестики-нолики» на C++. Игроки поочередно выбирают клетки на поле 3×3, вводя их номера. Побеждает тот, кто первым выстроит три своих символа (X или O) в ряд, столбец или диагональ. Есть два режима:

1. **Два игрока** – обычный режим, где играют два человека.
2. **Игра против компьютера** – компьютер выбирает случайные ходы.

После каждого хода программа проверяет победу или ничью. В случае победы выводится сообщение о выигрыше. После завершения игры можно начать заново.

2. **Разборы кода**

// отображения игрового поля

void printField(const vector<char>& field) {

cout << "\n";

for (int i = 0; i < 9; i += 3) {

// вывод строк игрового поля

cout << " " << field[i] << " | " << field[i + 1] << " | " << field[i + 2] << "\n";

if (i < 6) cout << "---+---+---\n"; // Разделитель строк

}

cout << "\n";

}

Выводит текущее состояние поля в удобном формате 3х3, далее добавляется разделительные линии между строками.

bool checkWin(const vector<char>& field, char player) {

// массив с выигрышными комбинациями (по строкам, столбцам и диагоналям)

const int winCombos[8][3] = {

{0, 1, 2}, {3, 4, 5}, {6, 7, 8}, // горизонтальные линии

{0, 3, 6}, {1, 4, 7}, {2, 5, 8}, // вертикальные линии

{0, 4, 8}, {2, 4, 6} // диагональные линии

};

// проверка есть ли у игрока три одинаковых символа в одной из комбинаций

for (auto& combo : winCombos) {

if (field[combo[0]] == player && field[combo[1]] == player && field[combo[2]] == player) {

return true; // Игрок победил

}

}

return false; // Победы нет

}

Проверяет 8 возможных выигрыш комбинаций, если три одинаковые символы Х или О , одной из них возвращает (true).

// получение корректного хода от игрока

int getPlayerMove(vector<char>& field) {

setlocale(LC\_ALL, "RU");

int move;

while (true) {

cout << "Введите номер клетки (1-9): ";

// проверка является ли ввод числом и попадает ли в диапазон от 1 до 9

if (!(cin >> move) || move < 1 || move > 9 || field[move - 1] == 'X' || field[move - 1] == 'O') {

cout << "Некорректный ввод. Попробуйте снова.\n";

cin.clear(); // сброс ошибку ввода

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // очистка буфер ввода

}

else {

return move - 1; // возвращать индекс клетки (от 0 до 8)

}

}

}

Пользователь выбирает 1 до 9 числа и проверяет свободна ли выбранная клетка, если ввод некорректен ( буква и тд) предлагает повторить вводить

// случайный ход компьютера

int getComputerMove(vector<char>& field) {

vector<int> availableMoves;

for (int i = 0; i < 9; i++) {

if (field[i] != 'X' && field[i] != 'O') {

availableMoves.push\_back(i); // добавление свободные клетки в список

}

}

return availableMoves[rand() % availableMoves.size()]; // возвращать случайный ход

}

Программа собирает список доступных клеток и выбирает случайную клетку от индекса.

int main() {

setlocale(LC\_ALL, "RU");

srand(time(0)); // инициализиация генератора случайных чисел

char playAgain; // переменная для хранения ответа пользователя на повтор игры

char gameMode; // переменная для выбора режима игры

do {

// выбор режима игры

cout << "Выберите режим игры: 1 - два игрока, 2 - игра против компьютера: ";

cin >> gameMode;

// инициализация игрового поля номерами клеток (1-9)

vector<char> field = { '1','2','3','4','5','6','7','8','9' };

char currentPlayer = 'X'; // начало игрок X

int moves = 0; // количество сделанных ходов

bool gameWon = false; // флаг, указывающий, выиграна ли игра

while (moves < 9 && !gameWon) { // пока не будет 9 ходов или победителя

printField(field); // вывод игрового поля

int move;

if (gameMode == '2' && currentPlayer == 'O') {

move = getComputerMove(field); // ход компьютера

cout << "Компьютер выбрал клетку " << move + 1 << "\n";

}

else {

move = getPlayerMove(field); // получение хода от игрока

}

field[move] = currentPlayer; // запись символа игрока в выбранную клетку

moves++; // увеличиваем счетчик ходов

//првоерка, победил ли текущий игрок

if (checkWin(field, currentPlayer)) {

printField(field); // вывод итогового игрового поля

cout << "Игрок " << currentPlayer << " победил!\n";

gameWon = true; // устанавливаем флаг победы

break; // завершаем цикл игры

}

// менять текущего игрока (переключаем между X и O)

currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

}

if (!gameWon) { // если игра завершилась, но победителя нет - ничья

printField(field);

cout << "Ничья!\n";

}

// спросить, хочет ли игрок сыграть еще раз

cout << "Хотите сыграть еще раз? (y/n): ";

cin >> playAgain;

} while (playAgain == 'y' || playAgain == 'Y'); // повтор игру, если ответ 'y' или 'Y'

return 0;

}

Предлагает выбрать режим игры игрок vs игрок или против компьютера.

Далее строит 3х3 клетки

Режим Игрок vs Игрок

Запрашивает ход игрока

Переключение между игроками

Проверяет победу

Проверяет ничья если все 9 клетки заняты

Режим компьютера

Компьютер случайно выбирает клетку

После окончания игры спрашивает хочет ли еще раз сыграть

**Мини-игра №4: «Битва с монстром»**

**Описание программы**

Эта программа представляет собой простую текстовую RPG-игру, в которой герой сражается с монстром. У каждого персонажа есть определённые характеристики, такие как здоровье, атака, защита, мана и количество лечений. В каждом раунде игрок может выбрать одно из нескольких действий, а затем монстр выполняет своё случайное действие. Игра продолжается до тех пор, пока здоровье одного из участников не упадёт до нуля.

**Разбор кода**

// структура, представляющая персонажа

struct Character {

std::string name; // имя персонажа

int health; // здоровье

int attack; // атака

int defense; // защита

int mana; // мана

int heals; // количество лечений

Структура Character представляет персонаж и содержит следующие параметры

Name – имя персонажа

Health – здоровье

Attack – атака

Defense – защита

Mana – мана

Heals – количество лечений

// метод атаки другого персонажа

void attackTarget(Character& target) {

setlocale(LC\_ALL, "RU");

int damage = attack - target.defense; // учитывать защиту цели

if (damage < 0) damage = 0; // урон не может быть отрицательным

std::cout << name << " атакует " << target.name << " и наносит " << damage << " урона.\n";

target.health -= damage; // уменьшение здоровье цели

}

Персонаж наносит урон монстру, вычитая его защиту из своей атаки, урон не может быть отрицательным, после атаки здоровье монстра уменьшается

// Метод для использования заклинания

void castSpell(Character& target) {

setlocale(LC\_ALL, "RU");

if (mana >= 10) { // проверка хватает ли маны

int spellDamage = attack \* 2; // заклинание наносит удвоенный урон

std::cout << name << " использует заклинание и наносит " << spellDamage << " урона!\n";

target.health -= spellDamage;

mana -= 10; // тратим ману

}

else {

std::cout << "Недостаточно маны для заклинания!\n";

}

}

Если у персонажа достаточно мана, он наносит двойный урон

После использования заклинания тратится 10 мана

Если маны не хватает выводит сообщение об недостаточной мане

// Метод для защиты

void defend() {

setlocale(LC\_ALL, "RU");

std::cout << name << " принимает защитную стойку! Урон понижен на 50% в следующем раунде.\n";

defense \*= 2; // временно увеличиваем защиту в 2 раза

}

Герой удваивает свою защиту, снижая получаемый урон в след раунде

// Метод для лечения

void heal() {

setlocale(LC\_ALL, "RU");

if (heals > 0) { // проверка есть ли ещё лечения

int healing = 20;

health += healing; // восстановление здоровье

--heals;

std::cout << name << " лечится на " << healing << " HP. Осталось лечений: " << heals << "\n";

}

else {

std::cout << "Лечения закончились!\n";

}

}

Если у героя есть лечение на 20Hp , при использование уменьшается на 1

// функция для отображения текущего состояния боя

void printStatus(const Character& hero, const Character& monster) {

setlocale(LC\_ALL, "RU");

std::cout << "\n--- Статус сражения ---\n";

std::cout << hero.name << " (Здоровье: " << hero.health << ", Защита: " << hero.defense << ", Мана: " << hero.mana << ", Лечения: " << hero.heals << ")\n";

std::cout << monster.name << " (Здоровье: " << monster.health << ")\n";

std::cout << "-------------------------\n\n";

}

Показывает текущие характеристики героя и монстра в каждом раунде

// ход игрока

void heroTurn(Character& hero, Character& monster) {

setlocale(LC\_ALL, "RU");

int choice;

std::cout << "Выберите действие:\n";

std::cout << "1. Атаковать\n2. Заклинание (10 маны)\n3. Защититься\n4. Лечение\n5. Уклонение (50% шанс)\n";

std::cin >> choice;

switch (choice) {

case 1:

hero.attackTarget(monster);

break;

case 2:

hero.castSpell(monster);

break;

case 3:

hero.defend();

break;

case 4:

hero.heal();

break;

case 5:

if (rand() % 2 == 0) { // 50% шанс успешного уклонения

std::cout << hero.name << " уклоняется от атаки!\n";

}

else {

std::cout << hero.name << " не смог уклониться!\n";

}

break;

default:

std::cout << "Некорректный ввод! Пропуск хода.\n";

}

}

Герой выбирает 1 из доступных действих, шанс уклонение 50%

// ход монстра

void monsterTurn(Character& monster, Character& hero) {

setlocale(LC\_ALL, "RU");

if (monster.health > 0) {

int action = rand() % 3; // случайное действие монстра

if (action == 0) {

monster.attackTarget(hero);

}

else if (action == 1) {

std::cout << monster.name << " использует сильную атаку!\n";

hero.health -= monster.attack \* 1.5; // усиленная атака

}

else {

std::cout << monster.name << " пытается оглушить " << hero.name << "!\n";

if (rand() % 2 == 0) {

std::cout << hero.name << " оглушён и пропускает ход!\n";

}

}

}

}

Монстр случайно выбирает свое действие

Обычная атака

Сильная атака (умножение в 1.5)

Оглушение с шансом 50% пропусстить ход

// главная функция

int main() {

setlocale(LC\_ALL, "RU");

srand(time(0)); // инициализация генератора случайных чисел

// создание персонажей

Character hero = { "Герой", 100, 15, 5, 30, 3 };

Character monster = { "Монстр", 80, 12, 3, 0, 0 };

// основной игровой цикл

while (hero.health > 0 && monster.health > 0) {

printStatus(hero, monster);

heroTurn(hero, monster);

if (monster.health > 0) {

monsterTurn(monster, hero);

}

}

// определение победителя

if (hero.health > 0) {

std::cout << "Вы победили!\n";

}

else {

std::cout << "Вы проиграли...\n";

}

return 0;

}

Создаются герой и монстр, бой будет продолжаться пока у обоих есть здоровье