Игра в шашки

Оглавление

[Цель 2](#_Toc533364766)

[Задачи 2](#_Toc533364767)

[Введение 3](#_Toc533364768)

[Архитектура игры 3](#_Toc533364769)

[Класс cell 3](#_Toc533364770)

[Класс board 4](#_Toc533364771)

[Класс InputOutput 6](#_Toc533364772)

[Класс game 7](#_Toc533364773)

[Анализ выполненной работы 10](#_Toc533364774)

[Инструкция по использованию: 10](#_Toc533364775)

[Использованная литература 11](#_Toc533364776)

# Цель

Разработать приложение, представляющее собой программную реализацию известной логической игры «Шашки».

# Задачи

Координаты и возможные состояния шашек то есть их цвет

Стартовое состояние шашек

Проверка возможен ли ход

Поверка возможна ли битва

Сделать ход

Отображение цвета шашки

Получение данных для будущего хода и их преобразование

Отображение доски

Отображение доски после сделанного хода

Количество выбывших шашек

Проверка на победителя

Смена игрока

Направление будущего хода

# Введение

Игра идет на поле размерами 8 X 8 клеток, шашки занимают первые три ряда с каждой стороны, бить можно произвольное количество шашек в любых направлениях, простая может бить назад, дамка может ходить на любое число полей, цель игры съесть или запереть все шашки противника. Каждая из сторон в начале игры имеет по 12 шашек. Шашки расставляются в трех ближайших к игрокам, горизонталях. Два центральных ряда полей остаются свободными. Здесь на этих полях, происходит сближение и первое соприкосновение противостоящих сил. Противники ходят поочередно, перемещая шашки своего цвета по игровым полям. Первыми начинают белые. Выигрывает та сторона, которой удалось уничтожить или заблокировать движение всех шашек противника.

# Архитектура игры

## Класс cell

В классе cell используется шаблон pair [1] для хранения координаты шашки и класс State для хранения данных о цвете каждой шашки

#pragma once

#include <utility>

using pos = std::pair<size\_t, size\_t>;

enum class **State** { BLACK , WHITE, BLANK};

class **Cell**

{

State mState;

public:

**Cell**(State state);

~**Cell**();

State **GetState**() const { return mState; }

void **SetState**(State state) { mState = state; }

};

## Класс board

#pragma once

#include "Cell.h"

#include <map>

using map = std::map<pos, Cell>;

class board

{

const size\_t mBoardSize = 8;

map mCells;

public:

enum class **MoveResult** { SECCESSFUL\_MOVE, SECCESSFUL\_COMBAT, PROHIBITED };

**board**();

~**board**();

void **ResetMap**();

MoveResult CheckMove(const pos &startPos, const pos &endPos, bool direction);

MoveResult MakeMove(const pos &startPos, const pos &endPos, bool direction);

const map& GetMap() const { return mCells; }

};

В этом классе мы используем шаблон map[2] для распределения шашек по доске как в стартовом состоянии:

void board::**ResetMap**()

{

for (size\_t i = 0; i < mBoardSize; i++)

{

for (size\_t j = 0; j < mBoardSize; j++)

{

State state(State::BLANK);

if (i == 0 && j % 2 != 0) {

state = State::BLACK;

}

if (i == 2 && j % 2 != 0) {

state = State::BLACK;

}

if (i == 1 && j % 2 == 0) {

state = State::BLACK;

}

if (i == 7 && j % 2 == 0) {

state = State::WHITE;

}

if (i == 6 && j % 2 != 0) {

state = State::WHITE;

}

if (i == 5 && j % 2 == 0) {

state = State::WHITE;

}

const pos position = pos(i, j);

Cell cell = Cell(state);

mCells.insert({ move(position), std::move(cell) });

}

}

}

так и в координатах последующих ходов далее мы проверяем возможен ли ход или будет битва

board::MoveResult board::CheckMove(const pos &startPos, const pos &endPos, bool direction)

{

const int dy= endPos.first- startPos.first;

const int dx= endPos.second - startPos.second;

board::MoveResult result = board::MoveResult::PROHIBITED;

if (endPos.first >= 0 && endPos.first < mBoardSize && endPos.second >= 0 && endPos.second < mBoardSize)

{

auto targetCellState = mCells.at(endPos).GetState();

if (targetCellState == State::BLANK)

{

if (abs(dx) == 2 && abs(dy) == 2) {

pos VictimPos((endPos.first + startPos.first) / 2, (endPos.second + startPos.second) / 2);

auto VictimCellState = mCells.at(VictimPos).GetState();

auto StartCellState = mCells.at(startPos).GetState();

result =

targetCellState != VictimCellState &&

StartCellState != VictimCellState ? board::MoveResult::SECCESSFUL\_COMBAT : result;

}

else if ((abs(dx) == 1 && dy == 1 && direction) || (abs(dx) == 1 && dy == -1 && !direction)) {

result = board::MoveResult::SECCESSFUL\_MOVE;

}

}

}

return result;

}

и делаем ход в зависимости от того что будет: ход или битва

board::MoveResult board::MakeMove(const pos &startPos, const pos &endPos, bool direction)

{

auto MoveResult = CheckMove(startPos, endPos, direction);

switch (MoveResult) {

case board::MoveResult::SECCESSFUL\_COMBAT :

mCells.at(endPos).SetState((mCells.at(startPos).GetState()));

mCells.at(startPos).SetState(State::BLANK);

mCells.at(pos((endPos.first + startPos.first) / 2, (endPos.second + startPos.second) / 2)).SetState(State::BLANK);

break;

case board::MoveResult::SECCESSFUL\_MOVE:

mCells.at(endPos).SetState((mCells.at(startPos).GetState()));

mCells.at(startPos).SetState(State::BLANK);

break;

}

return MoveResult;

}

## Класс InputOutput

#pragma once

#include <utility>

#include "Cell.h"

#include <map>

using movePos = std::pair <pos, pos>;

using map = std::map<pos, Cell>;

class **InputOutput**

{

static auto **CastState**(State state);

public:

**InputOutput**();

~**InputOutput**();

static movePos GetMove();

static void DrawBoard(const map& board);

};

В этом классе мы проводим вывод «цвета» шашки

auto InputOutput::**CastState**(State state)

{

std::string result;

switch (state)

{

case State::BLACK:

result = "B";

break;

case State::WHITE:

result = "W";

break;

default:

result = " ";

break;

}

return std::move( result);

}

Здесь мы используем шаблон move [3] для более эффективной передачи результата

Далее мы получаем данные для будущего хода и преобразовываем их

movePos InputOutput::GetMove() {

//22 33

movePos position;

size\_t p1, p2;

std::cin >> p1;

std::cin >> p2;

position.first.first = p1 / 10;

position.first.second = p1 - 10 \* position.first.first;

position.second.first = p2 / 10;

position.second.second = p2 - 10 \* position.second.first;

return move(position);

}

И рисуем доску

void InputOutput::DrawBoard(const map& board)

{

auto boardSize = sqrt(board.size());

for (size\_t i = 0; i < boardSize; i++)

{

for (size\_t j = 0; j < boardSize; j++)

{

std::cout << CastState(board.at(pos(i, j)).GetState());

std::cout << "|";

}

std::cout << std::endl;

}

}

## Класс game

#include "board.h"

#include "InputOutput.h"

enum class **Player** { NONE, BLACK, WHITE };

class game

{

board mBoard;

InputOutput mIO;

size\_t mBlackScore;

size\_t mWhiteScore;

Player mLastplayer;

movePos MakeIO();

void **SwitchPlayer**();

void **UpdateScore**();

bool **GetDirection**()const;

bool bIsSurrounded;

public:

**game**();

~**game**();

Player **GetWinner**();

void **Start**();

};

Это основной класс в нем собираются все вышеперечисленные функции и проводится сама игра а так же результат хода

movePos game::MakeIO()

{

//print board

system("cls");

auto map = mBoard.GetMap();

mIO.DrawBoard(map);

//ask move

auto newmove = mIO.GetMove();

return move(newmove);

}

Подсчет количества выбывших шашек

void game::**UpdateScore**()

{

if (mLastplayer == Player::BLACK) {

mBlackScore++;

}

else if (mLastplayer == Player::WHITE)

{

mWhiteScore++;

}

}

Проверка на победителя

Player game::**GetWinner**()

{

Player winner(Player::NONE);

if (bIsSurrounded) {

if (mLastplayer == Player::WHITE) {

winner = Player::BLACK;

}

else

{

winner = Player::WHITE;

}

}

else {

if (mWhiteScore == 12) {

winner = Player::WHITE;

}

else

if (mBlackScore == 12) {

winner = Player::BLACK;

}

}

return winner;

}

Смена игрока

void game::**SwitchPlayer**()

{

if (mLastplayer == Player::WHITE) {

mLastplayer = Player::BLACK;

}

else

{

mLastplayer = Player::WHITE;

}

}

Направление будущего хода

bool game::**GetDirection**() const

{

bool direction;

switch (mLastplayer) {

case Player::WHITE:

direction = false;

break;

case Player::BLACK:

direction = true;

break;

}

return move(direction);

}

Основная функция игры

void game::**Start**()

{

mBoard.ResetMap();

while (GetWinner() == Player::NONE)

{

board::MoveResult moveResult(board::MoveResult::PROHIBITED);

while (moveResult == board::MoveResult::PROHIBITED) {

// IOStuff

auto newmove = MakeIO();

bool direction = GetDirection();

moveResult = mBoard.MakeMove(newmove.first, newmove.second, direction);

}

if (moveResult==board::MoveResult::SECCESSFUL\_MOVE) {

UpdateScore();

}

SwitchPlayer();

}

}

# Анализ выполненной работы

Что сделано: создано игровое ядро, позволяющее играть на примитивном уровне

Что можно сделать:

1. Добавить графику, то есть нарисовать доску и нарисовать шашки(можно с помощью QT а можно оптимизировать функции лежащие в <https://github.com/Wencelsav/-/blob/master/%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8%20%D0%B4%D0%BB%D1%8F%20%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B8>) или вставить доску и шашки из картинок
2. Добавить «королеву»
3. Изменить способ ввода данных то есть добавить захват мыши
4. Добавить Искусственный интеллект
5. Добавить возможность изменения размеров доски и количества шашек
6. Добавить правила для других типов шашек
7. Добавить таймер
8. Дополнительные цвета доски

# Инструкция по использованию:

запустить в Visual Studio или при компиляции из этого файла добавить запускающую функцию

#include "game.h"

int main() {

game game;

game.Start();

}

# Использованная литература

1. <https://ru.cppreference.com/w/cpp/utility/pair>

2. <https://ru.cppreference.com/w/cpp/container/map>

3. https://ru.cppreference.com/w/cpp/utility/move