«ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

кафедра ПМиК

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»

Тема: «Крестики-нолики» в графическом режиме

Выполнил: студент группы ИС-241

Сотников П. А.

Проверил к.т.н., доцент кафедры прикладной математики и кибернетики

Ситняковская Е.И.

Оценка _____

Новосибирск – 2024г

Содержание

GitHub: https://github.com/playsees/Courcework_OOP

Постановка задачи

Курсовая работа должна быть написана с использованием объектно-ориентированных технологий. Описания объектов и методов необходимо оформить в отдельном модуле. Иерархия классов должна включать минимум 3 класса, один из которых – абстрактный. Реализовать игру «Крестики нолики в графическом режиме».

Теория

Полиморфизм

Полиморфизм — это принцип объектно-ориентированного программирования, который описывает возможность объектов различных классов использовать одинаковые интерфейсы или методы, но давать различные реализации этих методов. Полиморфизм позволяет обрабатывать объекты производных классов как объекты базового класса, что делает код более гибким и удобным для использования. Принцип полиморфизма включает в себя переопределение методов (полиморфизм через наследование) и полиморфизм через интерфейсы. Полиморфизм через наследование позволяет производным классам переопределить методы базового класса с собственной реализацией, при этом код, который использует базовый класс, может вызывать эти методы, не зависимо от фактического типа объекта. Это позволяет программисту работать с объектами различных классов, используя общие интерфейсы или абстрактные классы. Полиморфизм через интерфейсы предполагает, что разные классы могут реализовать один и тот же интерфейс, но давать различные реализации. Это позволяет использовать объекты разных классов через общий интерфейс, обеспечивая единые способы взаимодействия с ними.

Инкапсуляция

Инкапсуляция — это принцип объектно-ориентированного программирования, который объединяет данные (поля) и методы, оперирующие этими данными, внутри класса, и скрывает их от прямого доступа извне. Основная идея инкапсуляции заключается в том, что объекты должны иметь четко определенный интерфейс, через который можно выполнять операции с объектами, в то время как внутренняя структура и детали реализации остаются скрытыми. Принцип инкапсуляции помогает обеспечить контролируемый доступ к данным и методам класса, предотвращая модификацию или неправильное использование данных извне. Для этого используются модификаторы доступа, такие как публичный (public), приватный (private), защищенный (protected), которые определяют, какие члены класса могут быть доступны извне, какие могут быть доступны только внутри класса или его подклассов. Важными понятиями, связанными с инкапсуляцией, являются сокрытие информации и согласованность интерфейса.

Пример в коде:

```
class TicTacToe : public Display, public Input { //мн. наследование
private:
    vector<vector<Symbol>> board; //игровая доска
    vector<AbstractPlayer*> players; //игроки
    int freeCells; //сколько свободных клеток осталось
    t state; //игра идет (true) или уже закончилась (false)
```

С помощью модификатора доступа private мы предотвращаем модификацию или неправильное использование данных извне.

Наследование

Наследование — это принцип объектно-ориентированного программирования, который позволяет создавать новые классы на основе существующих классов. В рамках наследования, существующий класс, называемый родительским или базовым классом, передает свои свойства (поля и методы) новому классу, который называется дочерним или производным классом. Дочерний класс наследует все свойства родительского класса и может расширять их, добавлять новые свойства и методы, или переопределять унаследованные методы для своих нужд. Это позволяет создавать иерархию классов, группируя их по общим характеристикам и поведению. Принцип наследования позволяет повторно использовать код, минимизировать дублирование и упростить проектирование иерархии классов. Он также способствует созданию более абстрактной и гибкой архитектуры программы, позволяя работать с объектами различных классов, производных от одного базового класса, с помощью общего интерфейса.

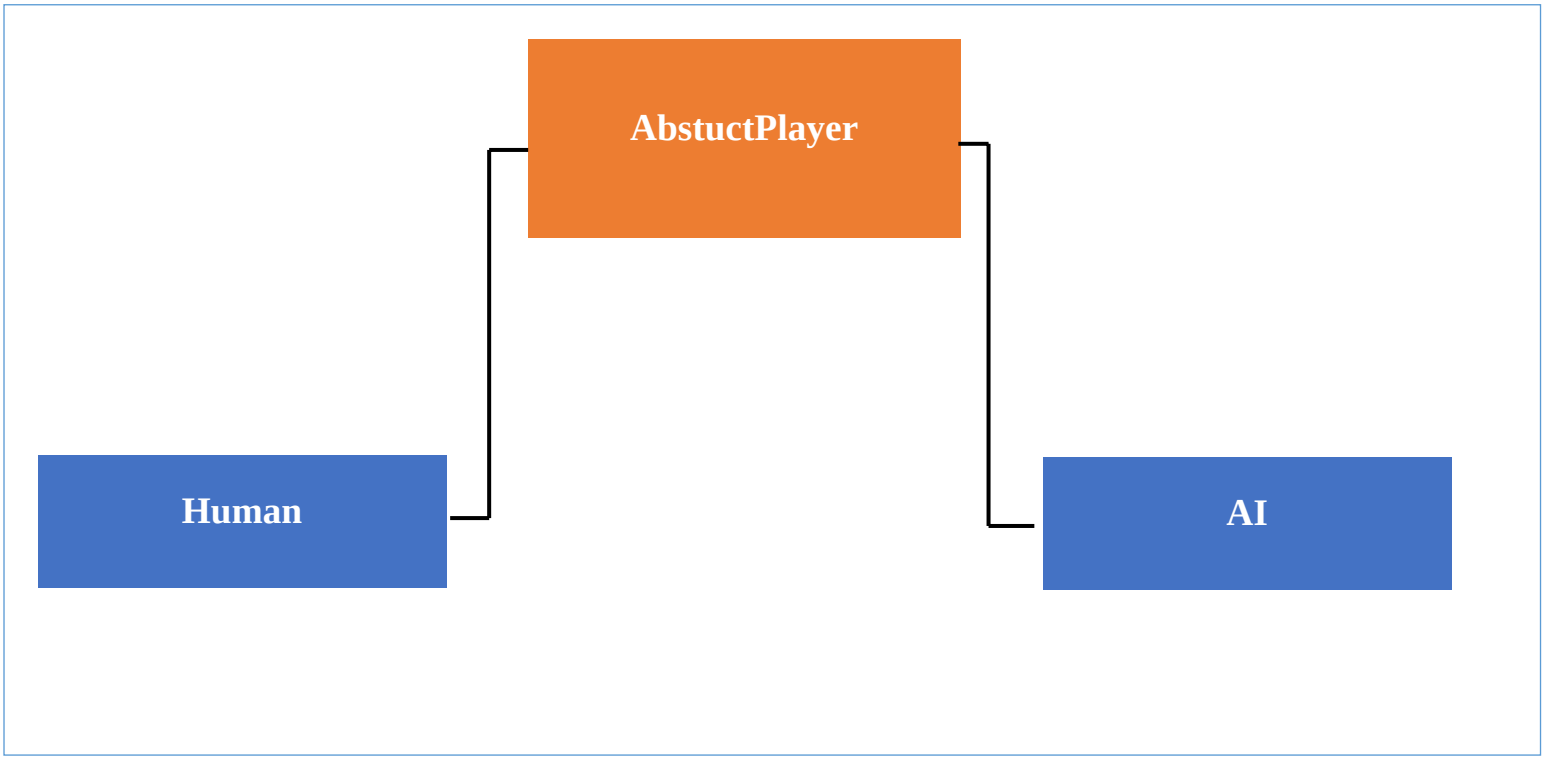
Пример в коде:

```
class Ai : public AbstractPlayer

class Human : public AbstractPlayer
```

Классы Ai и Human наследуются от базового класса AbstractPlayer. Они наследуют его свойства и методы, а также добавляют свои собственные.

Иерархия Объектов



От Базового класса AbstuctPlayer наследуются классы Human и AI.

Класс AbstuctPlayer:

- Является базовым классом
- Имеет абстрактный метод doStep
- Имеет дружественную функцию.

Класс Human

- Является унаследованным классов от AbstructPlayer
- Использует метод doStep принимает вектор векторов board, который представляет игровое поле, и устанавливает знак игрока на выбранную клетку.

Класс AI

- Является унаследованным классов от AbstructPlayer
- Использует метод doStep принимает вектор векторов board, который представляет игровое поле, и устанавливает знак игрока на выбранную клетку.

Класс Display – используется для отрисовки содержимого игры.

Класс Input – используется для интерфейса пользовательского ввода.

Класс TicTacToe

- Является производным классом от двух базовых классов Display и Input

- Использует метод handleInput, который отвечает за обработку пользовательского ввода в игре "Крестики-нолики" с использованием библиотеки SFML.
- Использует метод draw, который отвечает за отображение игрового поля "Крестики-нолики" в графическом окне с использованием библиотеки SFML.

Класс Symbol – является перечислением с тремя возможными значениями: X(крестик), O(нолик) и N(пусто).

Сеттеры:

- setRow – установить значение для строк
- setCol – установить значение для столбцов

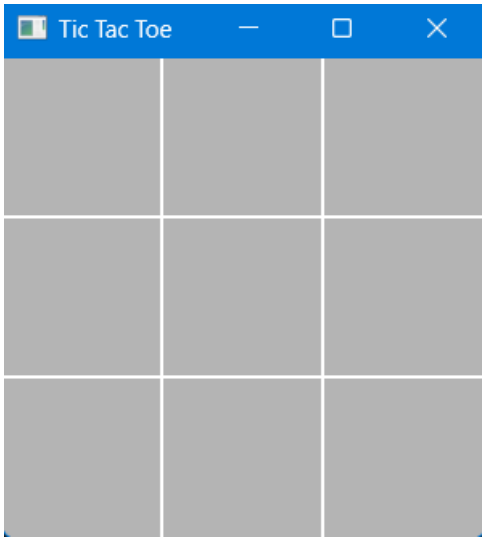
Геттер getSymbol, который возвращает знак текущего игрока.

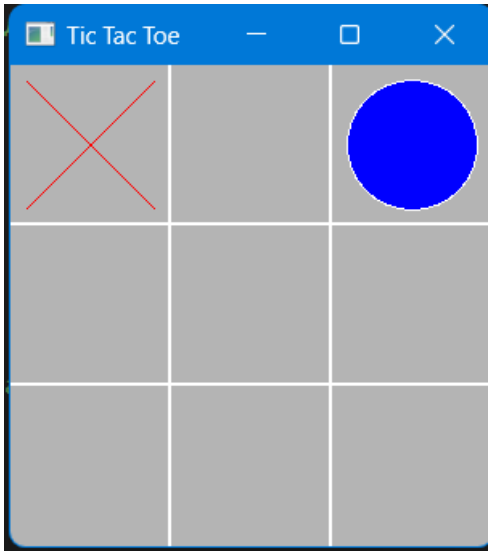
Описание Алгоритма

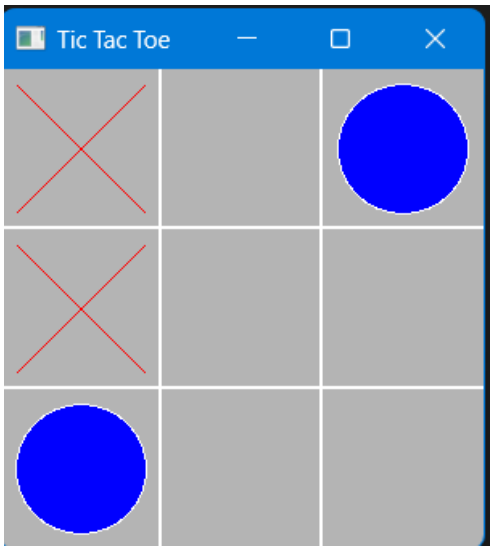
Алгоритм работы программы, следующий:

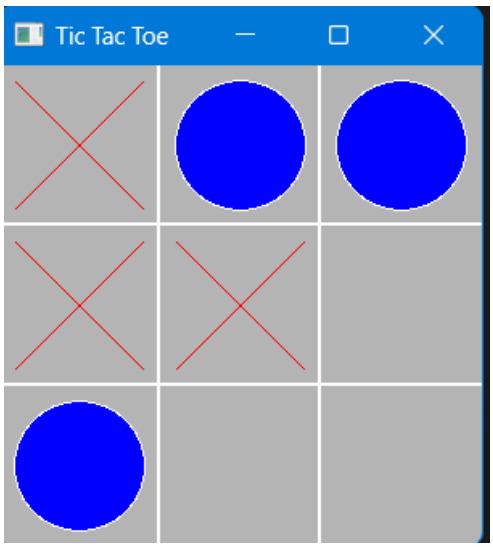
1. Создание игры с двумя игроками: человеком и компьютером.
2. Инициализация игрового поля и начальных значений для игроков.
3. Ожидание пользовательского ввода и обработка событий (нажатие клавиш мыши или клавиши Enter).
4. Если пользователь нажал на клетку игрового поля, то происходит проверка на валидность хода и установка значения на игровом поле.
5. Если пользователь нажал на клавишу Enter, то игра сбрасывается в начальное состояние.
6. После каждого хода происходит проверка на выигрыш одного из игроков или на ничью.
7. Если игра закончилась, то выводится соответствующее сообщение.
8. Если игра продолжается, то ход переходит к следующему игроку.

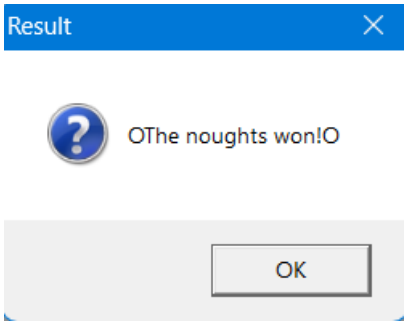
Результат работы программы

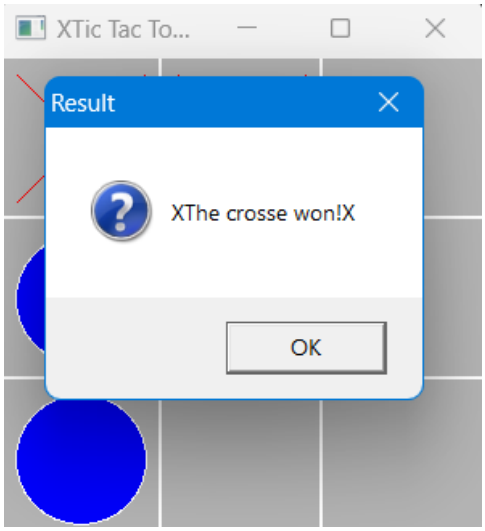


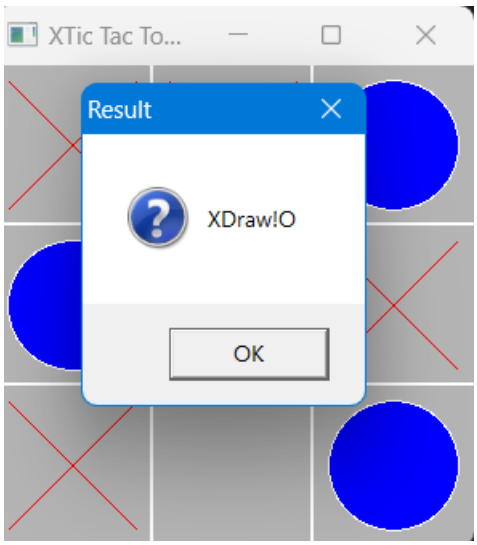





Tic Tac Toe


Result
OThe noughts won!O
OK


XTic Tac To...
Result
XThe crosse won!X
OK


XTic Tac To...
Result
XDraw!O
OK

Приложение

```
#include <SFML/Graphics.hpp>
#define WIN32_LEAN_AND_MEAN
#include <vector>
#include <iostream>
#include <fstream>
#include <windows.h>

using namespace std;

const static int ROWS = 3; //кол-во строк
const static int COLS = 3; //кол-во столбцов
const static int CELL_SIZE = 100; //размер одной клетки

//возможные знаки
enum class Symbol { X, O, N };

//базовый класс игрока
class AbstractPlayer {
protected: //инкапсуляция полей
Symbol c; //знак текущего игрока
int row, col; //выбранная клетка для хода

public:

//конструктор по умолчанию
AbstractPlayer() : c(Symbol::X), row(0), col(0) {}

//конструктор с параметром (перегрузка)
AbstractPlayer(Symbol c) : row(0), col(0) {
this->c = c;
}

//конструктор копирования
AbstractPlayer(const AbstractPlayer& p) {
}

//геттеры - сеттеры
void setRow(int row) {
this->row = row;
}

void setCol(int col) {
this->col = col;
}

Symbol getSymbol() const {
return c;
}

//текущий игрок делает ход
virtual void doStep(vector<vector<Symbol>>& board) = 0;

//дружественная функция вывода информации об объекте в поток
friend ostream& operator<<(ostream& stream, const AbstractPlayer& p) {
```

```
stream << "{" << p.row << "; " << p.col << "} : " << (int)p.c;
return stream;
}
};

//игрок - человек (наследование)
class Human : public AbstractPlayer {
};

//игрок компьютер (наследование)
class Ai : public AbstractPlayer {
};

//интерфейс для отрисовки содержимого игры
class Display {
public:
virtual void draw(sf::RenderWindow& window) const = 0;
};

//интерфейс пользовательского ввода
class Input {
public:
virtual void handleInput(sf::Event& event, sf::RenderWindow& window) = 0;
};

//основная игра
template<typename t>
class TicTacToe : public Display, public Input { //мн. наследование
private:
vector<vector<Symbol>> board; //игровая доска
vector<AbstractPlayer*> players; //игроки
int freeCells; //сколько свободных клеток осталось
t state; //игра идет (true) или уже закончилась (false)

public:

//конструктор по списку инициализации
TicTacToe(initializer_list<AbstractPlayer*> list) {
players = list;
restart();
}

//сброс игры
void restart() {
state = true;
freeCells = ROWS * COLS;
board = vector<vector<Symbol>>(ROWS, vector<Symbol>(COLS, Symbol::N));
}

//проверка кто выиграл
int winGame(Symbol c) const {

//главная диагональ
if (c == board[0][0] && c == board[1][1] && c == board[2][2])
return true;

//побочная
if (c == board[0][2] && c == board[1][1] && c == board[2][0])
return true;
```

```
//вертикали
for (int j = 0; j < COLS; j++) {
if (c == board[0][j] && c == board[1][j] && c == board[2][j])
return true;
}

//горизонтали
for (int i = 0; i < ROWS; i++) {
if (c == board[i][0] && c == board[i][1] && c == board[i][2])
return true;
}

//нет выигрыша для знака c
return false;
}

//обработка пользовательского ввода
void handleInput(sf::Event& event, sf::RenderWindow& window) override {

//нажали клавишу мыши
if (event.type == sf::Event::MouseButtonPressed && state) {

//получаем позицию мыши на клетке
sf::Vector2i mousePos = sf::Mouse::getPosition(window);
int row = mousePos.y / CELL_SIZE;
int col = mousePos.x / CELL_SIZE;

//если клетка валидная
if (row >= 0 && row < ROWS && col >= 0 && col < COLS && board[row][col] ==
Symbol::N) {

//устанавливаем значение на поле (ход игрока)
players[0]->setRow(row);
players[0]->setCol(col);
players[0]->doStep(board);
cout << *players[0] << endl;
freeCells--;

//игрок выиграл?
if (winGame(players[0]->getSymbol())) {
state = false;
MessageBoxA(NULL, "win krestiki", "result", MB_OK | MB_ICONQUESTION);
}
else {

//ничья
if (state && freeCells == 0) {
state = false;
MessageBoxA(NULL, "draw!", "result", MB_OK | MB_ICONQUESTION);
}
else {

//ход компьютера
players[1]->doStep(board);
cout << *players[1] << endl;
freeCells--;

//компьютер выиграл?
```

```
if (winGame(players[1]->getSymbol())) {
state = false;
MessageBoxA(NULL, "win noloki!", "result", MB_OK | MB_ICONQUESTION);
}
}
}

//ничья
if (state && freeCells == 0) {
state = false;
MessageBoxA(NULL, "draw!", "result", MB_OK | MB_ICONQUESTION);
}
}
}

//нажали на кнопку Enter
if (event.type == sf::Event::KeyReleased) {
if (event.key.code == sf::Keyboard::Enter) {
restart(); //сброс игры
}
}
}

//отрисовка игрового поля
void draw(sf::RenderWindow& window) const override {
window.clear(); //очистка окна

//по всем клеткам игрового поля
for (int i = 0; i < ROWS; ++i) {
for (int j = 0; j < COLS; ++j) {

//рисуем квадрат
sf::RectangleShape cell(sf::Vector2f(CELL_SIZE, CELL_SIZE));
cell.setFillColor({ 180,180,180 });
cell.setPosition(j * CELL_SIZE, i * CELL_SIZE);
cell.setOutlineThickness(2);

//отрисовка клетки
window.draw(cell);

//это крестик
if (board[i][j] == Symbol::X) {
sf::VertexArray cross(sf::Lines, 4);
cross[0].position = sf::Vector2f(j * CELL_SIZE + 10, i * CELL_SIZE + 10);
cross[1].position = sf::Vector2f((j + 1) * CELL_SIZE - 10, (i + 1) *
CELL_SIZE - 10);
cross[2].position = sf::Vector2f(j * CELL_SIZE + 10, (i + 1) * CELL_SIZE -
10);
cross[3].position = sf::Vector2f((j + 1) * CELL_SIZE - 10, i * CELL_SIZE +
10);
for (int k = 0; k < 4; k++) {
cross[k].color = { 255,0,0 };
}
window.draw(cross);
}

//это круг
else if (board[i][j] == Symbol::O) {
sf::CircleShape circle(CELL_SIZE / 2 - 10);
```

```
circle.setOutlineThickness(1);
circle.setPosition(j * CELL_SIZE + 10, i * CELL_SIZE + 10);
circle.setFillColor({ 0,0,255 });
window.draw(circle);
}
}
}

//обновляем экран
window.display();
}

//деструктор
~TicTacToe() {
for (int i = 0; i < players.size(); i++) {
delete players[i];
}
}
};

//главная функция
int main() {
srand(time(0)); //инициализация счетчика сл. чисел

//создаем игру
sf::RenderWindow window(sf::VideoMode(COLS * CELL_SIZE, ROWS * CELL_SIZE),
"Tic Tac Toe");
TicTacToe<bool> game({ new Human(Symbol::X), new Ai(Symbol::O) });

//пока игра активна
while (window.isOpen()) {

//оьбработка событий
sf::Event event;
while (window.pollEvent(event)) {

//закрыли окно с игрой
if (event.type == sf::Event::Closed) {
window.close();
}

//обработка пользовательского ввода
game.handleInput(event, window);
}

//отрисовка игры
game.draw(window);
}

//успешное завершение программы
return 0;
}
```