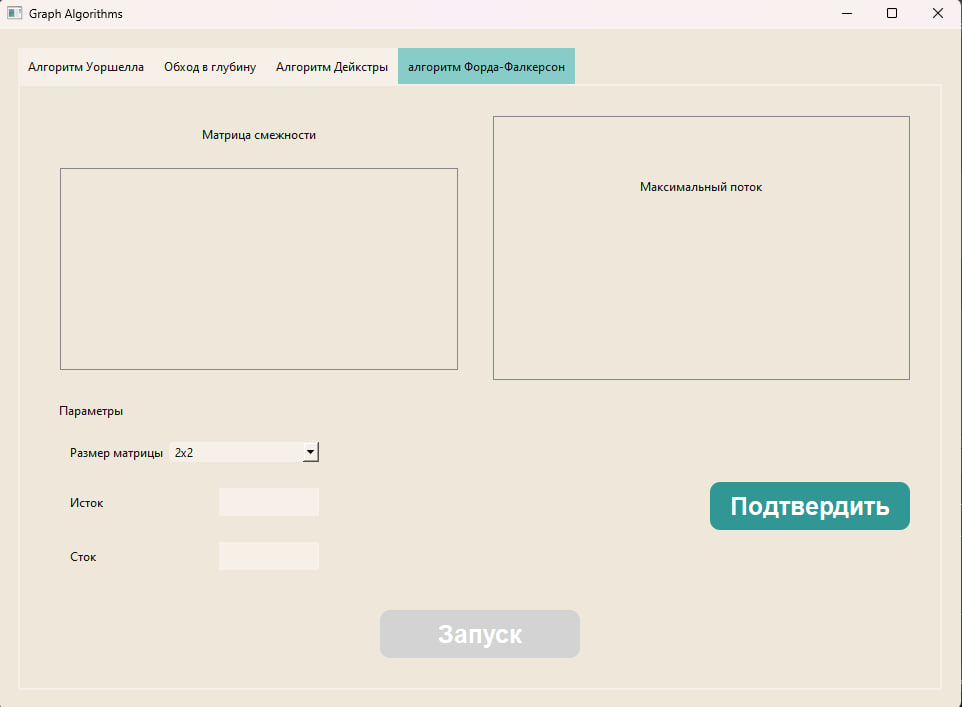
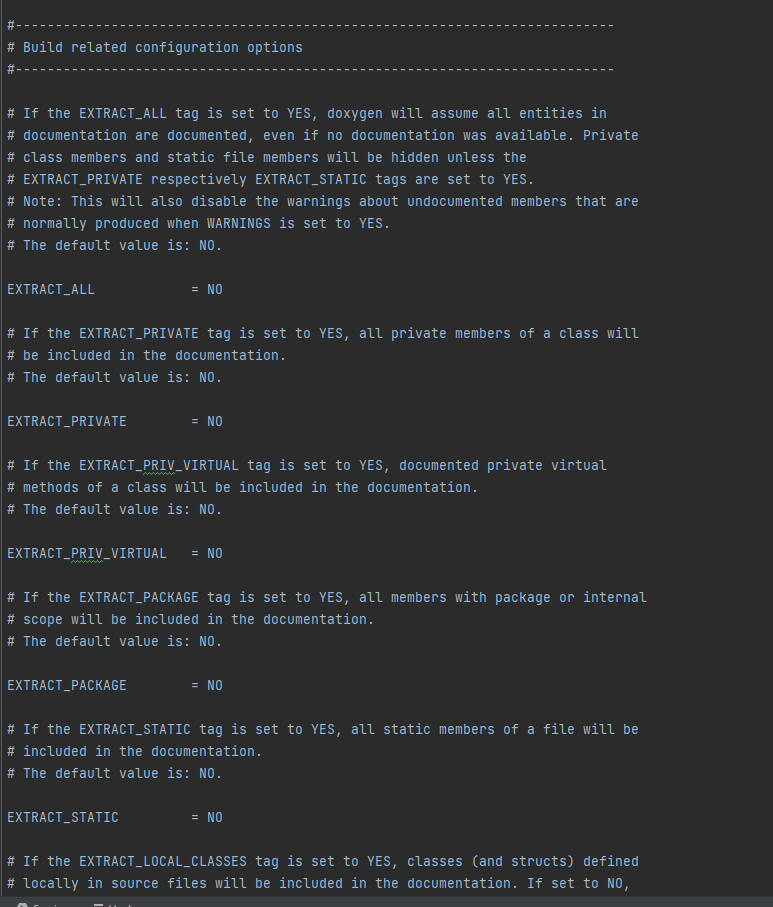
Результат роботи лабораторної 2   




1. Патерн "Модель-Вид-Контролер" (Model-View-Controller, MVC):
   * Опис: Розділяє додаток на три основні компоненти: модель, що зберігає дані і логіку, вид, що відображає інформацію користувачу, та контролер, що обробляє події та керує взаємодією між моделлю та видом.
   * Реалізація: Модель містить логіку гри та обробку даних, вид відображає графічний інтерфейс з ігровим полем та кораблями, а контролер оброблює події від користувача та спілкується з моделлю та видом.
2. Патерн "Спостерігач" (Observer):
   * Опис: Визначає залежність "один-багато" між об'єктами, так що зміна стану одного об'єкта призводить до автоматичного оновлення інших об'єктів, які на нього підписані.
   * Реалізація: Класи моделі, такі як ігрове поле та кораблі, можуть мати підписчиків (наприклад, класи, що представляють графічний інтерфейс). Коли стан моделі змінюється, вона повідомляє своїх підписчиків про це, щоб вони оновили свій вигляд.
3. Патерн "Фабричний метод" (Factory Method):
   * Опис: Надає загальний інтерфейс для створення об'єктів, але дозволяє підкласам вибирати конкретний тип об'єкту, який треба створити.
   * Реалізація: Можна використовувати фабричний метод для створення кораблів. Маємо базовий клас "Корабель", а підкласи можуть бути "Потоплений корабель", "Непотоплений корабель" тощо. Фабричний метод дозволяє підкласам визначати, який саме тип корабля потрібно створити.
4. Патерн "Одиночка" (Singleton):
   * Опис: Гарантує, що клас має тільки один екземпляр, та надає глобальну точку доступу до цього екземпляра.
   * Реалізація: Можна використати патерн Одиночка для створення гри, яка може бути доступна лише в одному екземплярі. Це дозволить забезпечити єдиний стан гри та доступ до неї з будь-якого місця програми.
5. Патерн "Стан" (State):
   * Опис: Дозволяє об'єкту змінювати свою поведінку в залежності від свого внутрішнього стану.
   * Реалізація: Можна використовувати патерн Стан для представлення різних станів гри, таких як "Стан розміщення кораблів", "Стан пошуку кораблів" та "Стан завершеної гри". Кожен стан визначає свою власну поведінку, наприклад, обмеження дій гравця в певних станах гри.
6. Патерн "Команда" (Command):
   * Опис: Упаковує запити або операції в окремі об'єкти, що дозволяє параметризувати клієнта з різними запитами, чергувати або реєструвати запити, а також підтримує скасування операцій.
   * Реалізація: Патерн Команда може використовуватися для реалізації дій користувача в грі. Кожна команда може мати відповідні методи, що виконують певні дії, такі як вибір клітинки на полі або атака противника. Команди можуть бути об'єктами, які зберігаються в черзі та виконуються в потрібний момент.

Патерн "Модель-Вид-Контролер" (Model-View-Controller, MVC):

// Модель

class Model {

private:

std::vector<std::vector<int>> board;

public:

Model(int size) : board(size, std::vector<int>(size, 0)) {}

void updateBoard(int x, int y, int value) {

board[x][y] = value;

}

const std::vector<std::vector<int>>& getBoard() const {

return board;

}

};

// Вид

class View {

private:

const Model& model;

public:

View(const Model& model) : model(model) {}

void displayBoard() const {

const std::vector<std::vector<int>>& board = model.getBoard();

// Відображення ігрового поля

}

};

// Контролер

class Controller {

private:

Model& model;

View& view;

public:

Controller(Model& model, View& view) : model(model), view(view) {}

void updateBoard(int x, int y, int value) {

model.updateBoard(x, y, value);

view.displayBoard();

}

void startGame() {

// Логіка гри

}

};

Патерн "Спостерігач" (Observer):

// Суб'єкт

class Subject {

private:

std::vector<Observer\*> observers;

public:

void attach(Observer\* observer) {

observers.push\_back(observer);

}

void detach(Observer\* observer) {

observers.erase(std::remove(observers.begin(), observers.end(), observer), observers.end());

}

void notify() {

for (Observer\* observer : observers) {

observer->update();

}

}

};

// Конкретний спостерігач

class ConcreteObserver : public Observer {

private:

Subject& subject;

public:

ConcreteObserver(Subject& subject) : subject(subject) {

subject.attach(this);

}

void update() override {

// Оновлення спостерігача

}

~ConcreteObserver() {

subject.detach(this);

}

};

Патерн "Фабричний метод" (Factory Method):

// Базовий клас Корабель

class Ship {

public:

virtual void print() const = 0;

};

// Підкласи Корабель

class Destroyer : public Ship {

public:

void print() const override {

// Друк інформації про Destroyer

}

};

class Cruiser : public Ship {

public:

void print() const override {

// Друк інформації про Cruiser

}

};

// Фабричний метод

class ShipFactory {

public:

virtual Ship\* createShip() const = 0;

};

// Конкретний фабричний метод

class DestroyerFactory : public ShipFactory {

public:

Ship\* createShip() const override {

return new Destroyer();

}

};

class CruiserFactory : public ShipFactory {

public:

Ship\* createShip() const override {

return new Cruiser();

}

};

Патерн "Одиночка" (Singleton):

// Одиночка

class Singleton {

private:

Singleton() {

// Приватний конструктор

}

public:

Singleton(const Singleton&) = delete;

void operator=(const Singleton&) = delete;

static Singleton& getInstance() {

static Singleton instance;

return instance;

}

// Додаткові методи та дані класу

};

Патерн "Стан" (State):

// Клас Стану

class State {

public:

virtual void handle() const = 0;

};

// Конкретні стани

class PlayingState : public State {

public:

void handle() const override {

// Обробка стану гри "Playing"

}

};

class FinishedState : public State {

public:

void handle() const override {

// Обробка стану гри "Finished"

}

};

// Контекст

class Game {

private:

State\* currentState;

public:

Game() : currentState(nullptr) {}

void setState(State\* state) {

currentState = state;

}

void process() const {

currentState->handle();

}

};

Патерн "Команда" (Command):

// Команда

class Command {

public:

virtual void execute() const = 0;

};

// Конкретні команди

class AttackCommand : public Command {

public:

void execute() const override {

// Виконання команди "Атака"

}

};

class MoveCommand : public Command {

public:

void execute() const override {

// Виконання команди "Переміщення"

}

};

// Виконавець

class Executor {

public:

void executeCommand(const Command& command) const {

command.execute();

}

};