Архітектура та реалізація ігрового проєкту «Sea-Battle-2(Мобільний корабель)»

(Завдання №32)

Виконала:

Студентка 2 курсу

Групи «Комп’ютерна математика»

Ярослава Креневич

2025

Table of Contents

Вступ

Актуальність теми та постановка задачі

Створення комп'ютерних ігор є наочним прикладом застосування об'єктно-орієнтованого програмування. Класична гра «Морський бій» була модифікована і перетворена більше у гру «Тепло-Холодно» на полі 10x10. Задача загалом була модифікована для ускладнення задачі: замість статичного флоту використовується один однопалубний корабель, який може рухатися після пострілу супротивника. Це вимагає від гравців та штучного інтелекту застосування геометричних методів пошуку на основі Манхеттенської відстані.

Мета та завдання проєкту

Мета: Розробити багаторежимну консольну гру мовою C++, що демонструє принципи ООП, поліморфізм та реалізацію пошукового алгоритму для ШІ

Завдання:

1) Реалізувати ігрове поле та логіку розрахунку Манхеттенської відстані
2) Використати псевдографіку для візуального відображення гри
3) Створити ієрархію гравців для підтримки режимів Human vs. Human та Human vs. Bot
4) Розробити алгоритм SmartBot на основі перетину множин можливих позицій
5) Забезпечити коректну обробку фаз гри: постріл, отримання зворотного зв'язку та рух цілі

Основна частина. Архітектура та дизайн проєкту

Обґрунтування Архітектурного Підходу

Проєкт побудовано з використанням архітектурного патерну Model-View-Controller (MVC) для забезпечення розділення відповідальності та гнучкості.

- Model (Модель/Логіка): Класи Board, Point. Містить усі правила гри, стан поля, розрахунок відстані та валідацію руху.
- Controller (Контролер/Управління): Клас Game. Керує життєвим циклом гри, ініціалізацією, перемиканням ходів та обробкою подій.
- View (Вигляд/Відображення): Реалізовано через консольне виведення (Board::printBoard) з використанням псевдографіки та механізмів очищення екрана для забезпечення конфіденційності (режим Hotseat)

Інкапсуляція та клас Board

Клас Board інкапсулює логіку ігрового поля 10x10

Метод	Призначення	Ключові Операції
receiveShot(Point p)	Обробка пострілу.	Перевірка p == shipPosition. Повертає −1 (влучання) або L1 (дистанцію) через функцію manhattanDistance.
moveShip(Point newPos)	Валідація та переміщення корабля.	Перевіряє isNeighbor() (логіка N8) та isCellShot() (заборона руху на обстріляну клітинку).

isShipSunk() const	Перевірка перемоги.	Перевіряє, чи містить grid[shipPosition] мітку CELL_SHOT.
isValidCoordinate()	Перевірка меж поля.	Використовується для валідації вводу гравців.

Поліморфізм та ієрархія гравців

Для підтримки різних режимів гри використовується поліморфізм на основі успадкування від абстрактного базового класу Player.

Абстрактний клас Player

Визначає виключно віртуальні методи для всіх гравців.

- virtual Point makeShot() = 0; (Куди стріляти?)
- virtual Point chooseMove() = 0; (Куди рухатися?)
- updateEnemyView(Point, distance); (Спільна логіка запису результатів)

Реалізації нащадків

- HumanPlayer: Методи реалізовані через консольний ввід (std::cin) з надійною валідацією та лімітом спроб (MAX_ATTEMPTS).
- RandomBot: Методи використовують rand() для генерації валідних, необстріляних координат (для пострілу) або сусідніх клітинок (для руху).
- SmartBot: Методи реалізовані на основі алгоритмічного пошуку

Алгоритм Розумного ШІ

Логіка SmartBot є основою проєкту і ґрунтується на геометрії та постійній корекції даних.

Структура даних

Бот підтримує приватне поле std::vector<Point> candidatePositions, що містить усі клітинки, де теоретично може бути корабель цілі.

Алгоритм пошуку

1) Початкова Фільтрація (filterCandidates): Викликається після промаху. Бот використовує отриману відстань D та координати пострілу (x, y) для видалення всіх точок P зі списку, які не задовольняють умові L1(P, (x, y)) = D.
2) Облік Руху (expandCandidates): Викликається після того, як ціль здійснила рух. Це враховує, що корабель міг перемістится до будь-якої із 8 сусідніх (N8) клітинок.
    - Алгоритм замінює кожну точку P у списку на множину {P об’єднати {Neighbors}(P)}, використовуючи std::set для забезпечення унікальності нових кандидатів.
    - Критично важлива умова: виключаються обстріляні клітинки (!enemyView.isCellShot()).
3) Прийняття рішення (makeshot). : Бот обирає наступну ціль з поточного списку candidatePositions (у простій версії — першу або випадкову валідну).

Висновок:

Проєкт "Морський Бій (Мобільний Корабель)" успішно реалізує поставлені завдання, демонструючи глибоке розуміння архітектурних патернів та об'єктно-орієнтованих принципів C++.

Використання поліморфізму (ієрархія Player) дозволило легко додати різні режими гри, тоді як dynamic_cast забезпечив безпечну інтеграцію специфічної логіки SmartBot.

Розроблений алгоритм пошуку, що базується на Манхеттенській геометрії та динамічній корекції через рух цілі, забезпечує SmartBot ефективну стратегію "полювання" проти людини.

Додавання механізму ліміту спроб та автоматичного розміщення/пропуску ходу підвищило стійкість гри до некоректного вводу користувача.

Цей проєкт може бути розширений шляхом додавання графічного інтерфейсу та клієнт-серверної взаємодії, що підтверджує його гнучкість та масштабованість.