

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Связывание классов

Студент гр. 3388

Баатыров Э.М.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Разработать объектно-ориентированную модель игры с сохранением/загрузкой состояния, обеспечивающую гибкость и расширяемость для дальнейшей реализации пользовательского интерфейса. Это является важным шагом в реализации проекта первой игры на языке программирования C++.

Задание

- а) Создать класс игры, который реализует следующий игровой цикл:
- i) Начало игры
 - ii) Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
 - iii) В случае проигрыша пользователь начинает новую игру
 - iv) В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

б) Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

- Класс игры может знать о игровых сущностях, но не наоборот
- Игровые сущности не должны сами порождать объекты состояния
- Для управления самой игрой можно использовать обертки над командами

- При работе с файлом используйте идиому RAII

Выполнение работы

Класс Game

Класс Game нужен для управления логикой морского боя. Он отвечает за инициализацию игры, обработку ходов игроков (пользователя и компьютера), определение победителя и сохранение/загрузку состояния игры. Он оркеструет взаимодействие между различными компонентами игры, такими как игровые поля, менеджеры кораблей и менеджеры способностей (хотя последние пока не используются явно в коде). Поля класса *cell*:

Поля класса:

- `game_state`: Объект класса `GameState`, хранящий текущее состояние игры (расположение кораблей, чья очередь хода, и т.д.). Это центральное хранилище информации о состоянии игры.

- `player_field` (`shared_ptr<PlayingField>`): Указатель на игровое поле игрока. Использование `shared_ptr` предотвращает утечки памяти, позволяя нескольким объектам совместно использовать управление этим ресурсом.

- `enemy_field` (`shared_ptr<PlayingField>`): Указатель на игровое поле компьютера.

- `player_ships` (предположительно `shared_ptr<ShipsManager>`): Указатель на менеджер кораблей игрока, который следит за состоянием кораблей игрока (повреждения, потопленные корабли).

- `enemy_ships` (`shared_ptr<ShipsManager>`): Указатель на менеджер кораблей компьютера.

Методы класса:

- `Game(shared_ptr<PlayingField> player_field, shared_ptr<PlayingField> enemy_field, shared_ptr<ShipsManager> player_ships, shared_ptr<ShipsManager> enemy_ships)`: Конструктор, инициализирующий игру заданными игровыми полями и менеджерами кораблей.

- `start()`: Инициализирует начало игры, возможно, устанавливая начальные значения в `game_state`.
- `playerTurn(size_t x, size_t y, bool use_skill=false, size_t skill_x=0, size_t skill_y=0)`: Обработывает ход игрока. Принимает координаты выстрела (x, y) и опционально информацию о применении способности (use_skill, skill_x, skill_y). Этот метод должен обновлять `game_state`.
- `enemyTurn()`: Обработывает ход компьютера, используя некоторую стратегию (не реализовано в представленном коде). Также должен обновлять `game_state`.
- `isPlayerWin()`: Проверяет, победил ли игрок.
- `isEnemyWin()`: Проверяет, победил ли противник.
- `check_game_status(bool reverse=false)`: Внутренний метод, вероятно, используемый `playerTurn` и `enemyTurn` для проверки состояния игры после каждого хода. `reverse` может указывать на проверку с точки зрения противника.
- `reload_enemy(shared_ptr<PlayingField> playing_field, shared_ptr<ShipsManager> ships_manager)`: Метод для перезагрузки состояния врага. Позволяет изменять состояние врага без полного перезапуска игры.
- `reload_game(shared_ptr<PlayingField> player_field, shared_ptr<PlayingField> enemy_field, shared_ptr<ShipsManager> player_ships, shared_ptr<ShipsManager> enemy_ships)`: Метод для полной перезагрузки игры с новыми полями и менеджерами кораблей.
- `getIsPlayerTurn()`: Возвращает `true`, если ход игрока, `false` - иначе.
- `getIsGameStarted()`: Возвращает `true`, если игра начата, `false` - иначе.
- `save(string filename)`: Сохраняет текущее состояние игры (`game_state`) в файл.
- `load(string filename)`: Загружает состояние игры из файла.

Класс GameState

Класс GameState предназначен для хранения всего состояния игры в любой момент времени. Он скрывает внутренние детали реализации от внешнего мира, предоставляя доступ к состоянию только через методы класса Game (благодаря ключевому слову friend). Это важно для поддержания целостности данных и предотвращения некорректного изменения состояния игры извне.

Поля класса:

- player_field (shared_ptr<PlayingField>): Указатель на игровое поле игрока.
- enemy_field (shared_ptr<PlayingField>): Указатель на игровое поле противника.
- player_ships (shared_ptr<ShipsManager>): Указатель на менеджер кораблей игрока.
- enemy_ships (shared_ptr<ShipsManager>): Указатель на менеджер кораблей противника.
- info_holder: Объект для хранения дополнительной информации об игре (его назначение не ясно из представленного кода).
- skills_manager: Менеджер способностей (на данный момент не используется).
- is_player_turn (bool): Флаг, указывающий, чей сейчас ход (игрока или компьютера).
- is_game_started (bool): Флаг, указывающий, начата ли игра.

Методы класса:

- GameState(): Конструктор, инициализирующий состояние игры.
- operator<<(ostream& os, const GameState& game_state): Перегрузка оператора вывода в поток. Необходима для сериализации состояния игры в строку для сохранения в файл.
- operator>>(istream& is, GameState& game_state): Перегрузка оператора ввода из потока. Необходима для десериализации состояния игры из файла

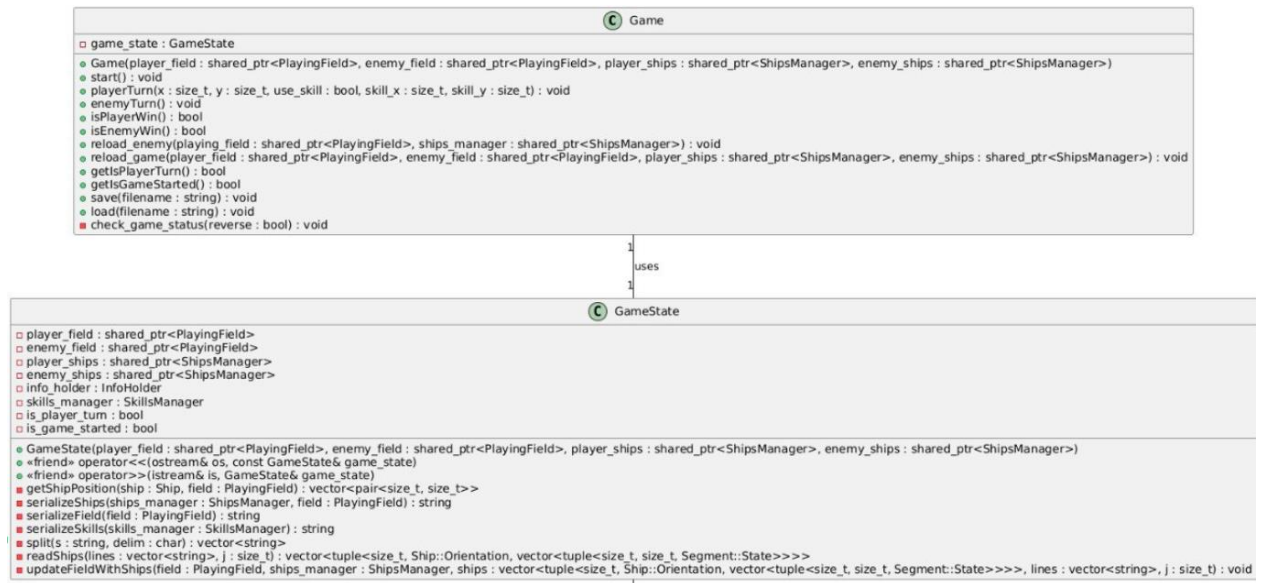
- `getShipPosition(Ship& ship, PlayingField& field) const:`
Вспомогательный метод для получения позиций корабля на поле.
- `serializeShips(ShipsManager& ships_manager, PlayingField& field) const:`
Сериализует информацию о кораблях в строку.
- `serializeField(PlayingField& field) const:` Сериализует информацию об игровом поле в строку.
- `serializeSkills(const SkillsManager& skills_manager) const:` Сериализует информацию о способностях в строку.
- `split(const string &s, char delim):` Вспомогательный метод для разбиения строки по разделителю.
- `readShips(vector<string>& lines, size_t& j):` Читает информацию о кораблях из строки (вероятно, при загрузке из файла).
- `updateFieldWithShips(PlayingField& field, ShipsManager& ships_manager, vector<tuple<size_t, Ship::Orientation, vector<tuple<size_t, size_t, Segment::State>>>>& ships, vector<string>& lines, size_t& j):` Обновляет игровое поле на основе информации о кораблях (вероятно, при загрузке из файла).

main()

Функция `main` в данном коде служит для тестирования разработанных классов (`Game`, `GameState`, `ShipsManager`, `PlayingField`). Она создает экземпляры этих классов, инициализирует игру, выполняет несколько ходов (как игрока, так и компьютера), сохраняет и загружает состояние игры, а затем проверяет, кто выиграл.

Программный код приведен в приложении А.

UML-диаграмма классов



Выводы

В ходе выполнения лабораторной работы была успешно разработана объектно-ориентированная модель игры «Морской бой», включающая в себя механизмы сохранения и загрузки состояния игры. Модель демонстрирует гибкую и расширяемую архитектуру, основанную на четком разделении ответственности между классами. Работа над проектом способствовала углублению знаний в области объектно-ориентированного программирования и практическому применению принципов ООП на языке C++. Разработанная архитектура готова к дальнейшему расширению и интеграции с другими компонентами игры.