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

Студентка гр. 3385 _____ Мучник М.К.

Преподаватель _____ Первицкий А.Ю.

Санкт-Петербург

2024

Цель работы

Разработать класс, отвечающий за игровой цикл и объединяющий в себе все игровые сущности. Изучить работу с файлами, реализовать сохранение.

Задание

a. 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.

b. 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 можно использовать обертки над командами
- При работе с файлом используйте идиому RAII

Выполнение работы

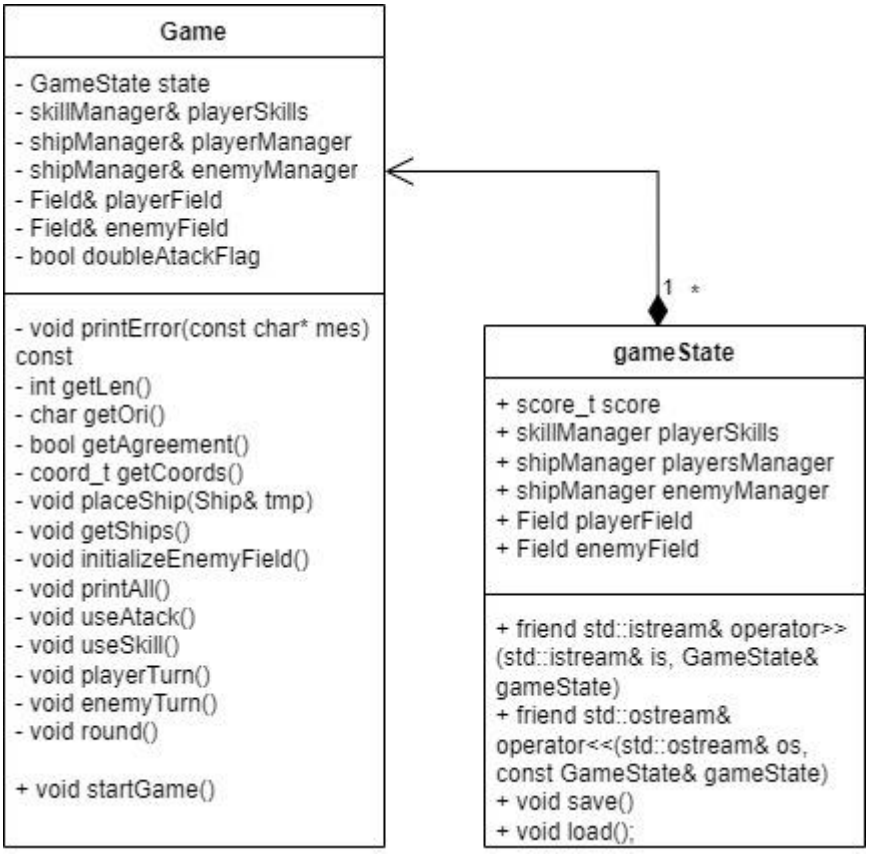

Рисунок 1 – UML-диаграмма реализованных классов

Реализация класса игры.

Для реализации игрового процесса был создан класс **Game**, в котором реализована основная логика игры, разбитая на следующие приватные и один публичный методы:

- методы, отвечающие за считывание информации и инициалищацию getLen(), getOri, getCoords(), getAgreement() – считывают из стандартного потока ввода следующие данные: длина корабля, ориентация корабля, координаты клетки поля, пользовательское соглашение (да/нет) соответственно;
- вспомогательный метод printError(const char* mes) – выводит в стандартный поток ошибок сообщение об ошибке mes, также отчищает поток ввода (на случай, если ошибка вызвана некорректным вводом);

- метод placeShip(Ship& ship) – располагает корабль, полученный в качестве аргумента, на поле.
- методы getShips() и initializeEnemyField отвечают за расстановку кораблей на поле. Пользователю предлагается вручную поставить свои корабли, в то время как корабли противника расставляются случайным образом;
- метод PrintAll() – вспомогательный метод, выводящий все игровые сущности и информацию о них (доступную игроку) в стандартный поток вывода;
- метод useAtack() – ожидает введение координат поля, после чего осуществляет по ним атаку. Если при атаке корабль был уничтожен, игроку выдаётся случайная способность;
- метод useSkill() – применяет первую по очерёдности способность из очереди способностей. Запрашивает ввод координат для способности сканера. При успешном применение способности выводит соответствующее сообщение на экран;
- методы playerTurn() и enemyTurn() – реализуют ход игрока и противника. Игроку предлагается на выбор осуществить атаку или воспользоваться способностью, после чего вызывается соответствующий метод. Противник в свой ход стреляет по случайной клетке;
- метод round() – запускает раунд, в котором чередуются ходы игрока и противника до тех пор, пока корабли первого или второго не будут уничтожены. При завершении раунда пользователю на экран выводится сообщение о победе либо поражение, после чего запускается новая игра с переинициализацией тех классов, которые необходимы;
- публичный метод startGame() – запускает игровой процесс: вызывает методы getShip(), initializeEnemyField(), round(). При

наличие в каталоге файла save.txt предлагает пользователю загрузить предыдущее сохранение игры.

В классе определены приватные поля классов состояния игры, пользовательского и вражеского менеджера кораблей и игрового поля, пользовательского менеджера способностей, а также флага двойной атаки. Все игровые сущности непосредственно хранятся в классе состояния игры, доступ к ним в классе игры реализован через ссылки.

Реализация класса состояния игры.

В качестве класса состояния игры был реализован класс **gameState**, публичными полями которого являются игровые сущности, а также структура score_t, содержащая информацию об игровом счёте.

Для класса состояния также переопределены операторы ввода и вывода, извлекающие или записывающие из потока информацию о текущем состояние игровых сущостей.

Сохранение и загрузку игры осуществляют методы save() и load(), по умолчанию открывающие файл save.txt и извлекающие/записывающие соответствующую информацию при помощи переопределённых операторов.

Выводы

Разработан класс, отвечающий за игровой цикл и объединяющий в себе все игровые сущности. Изучена работа с файлами и обработка исключений, реализовано сохранение и загрузка.