**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. 3384 |  | Кузьмин К. В. |
| Преподаватель |  | Шестопалов Р.П. |

Санкт-Петербург

2024

## Цель работы.

Связать между собой классы, созданные в предыдущих лабораторных работах, в полноценную игру «Морской Бой». Реализовать графичечский интерфейс, ввод с клавиатуры и игровой цикл.

## Задание.

Создать класс игры, который реализует следующий игровой цикл:

Начало игры

Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку. В случае проигрыша пользователь начинает новую игру 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.

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 самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

## 

## Выполнение работы.

Состояние игры хранит класс GameState. Его поля – это std::unique\_Ptr<Player> player и opponent. Умные указатели используются, чтобы можно было инициализировать игроков не при создании объекта GameState, а позже (используя ввод пользователя или файл сохранения).

Для состояния игры были перегружены операторы ввода/вывода из потока. Состояние игры преобразуется в json-формат с помощью библиотеки nlohmann/json. Если какой-либо из игроков не определен, то будет выброшено исключение GameStateUndefinedException.

Для игроков определены конструктор из json-объекта и метод to\_json, преобразующий игрока в json-объект. Аналогичные конструктор и метод определены для игрового поля, менеджера кораблей и менеджера способностей. Менеджер способностей преобразуется в список названий способностей, которыми владеет игрок. Менеджер кораблей преобразуется в список кораблей, а каждый корабль в свою очередь преобразуются в json-объект, содержащий состояние сегментов, кооординаты и ориентацию корабля. Поле преобразуется в двухмерный список клеток, а каждая клетка поля преобразуется в json-объект, содержащий булевое значение, отвечающее за наличие тумана в этой клетке. Таким образом при загрузке игрока корабли и поле загружаются отдельно, поэтому после загрузки поля и менеджера кораблей необходимо поместить каждый корабль в только что загруженное поле, иначе они никак не будут связаны между собой.

Класс Game отвечает за ход игры. Он содержит в себе GameState и отвечает за обработку ввода пользователя.

Метод save сохраняет игру в файл save.json. Для этого открывается поток записи в файл std::ofstream и в него записывается объект GameState.

Метод load загружает игру из save.json. Если при открытии файла для чтения произошла ошибка, то выведется сообщение об ошибке. В противном случае из файла считается объект GameState.

Метод createField отвечает за создание поля. Он вводит из потока ввода размеры поля и создает игроков с полями соответствующего размера.

Метод createShips отвечает за создание кораблей. Он вводит из потока ввода количество кораблей размера 1, 2, 3 и 4. Далее проверяет, чтобы корабли выбранного размера могли быть размещены на поле, иначе повторяет ввод. Далее для обоих игроков создаются менеджеры кораблей, а корабли расставляются случайным образом с помощью метода placeShipRandomly.

Метод doCommand принимает на вход строку – команда, введенная игроком, и выполняет её:

* Cursor – вводит координаты из потока ввода и перемещает в них курсор игрока.
* Attack – выполняет атаку. После атаки сразу же происходит атака соперника с помощью метода doOpponentMove.
* Ability – использует способность и выводит её результат в поток вывода.
* Save – сохраняет игру.
* Load – загружает игру.
* Если команды, которую ввёл пользователь, не существует, выводится соответствующее сообщение.

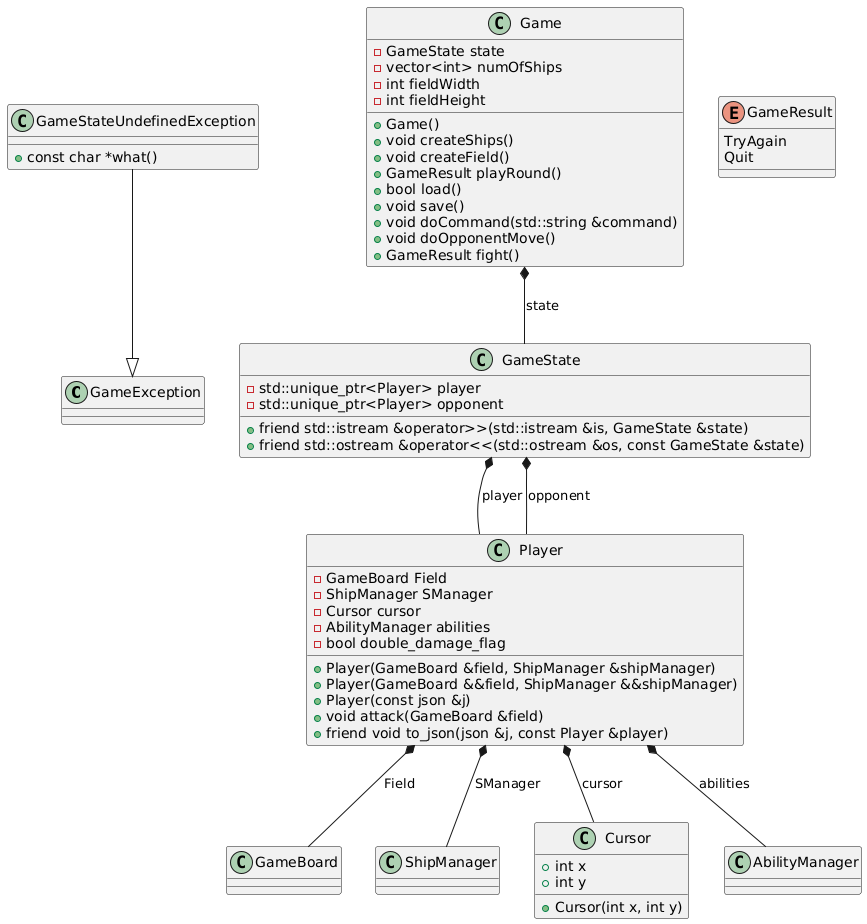
Метод doOpponentMove выполняет атаку соперника. Координаты выбираются случайным образом до тех пор, пока не будут указывать на клетку в тумане или на клетку с поврежденным сегментом корабля. Курсор оппонента перемещается по этим координатам и выполняется атака.

Метод playRound реализует игровой цикл одного раунда. Раунд заканчивается тогда, когда у какого-либо из игроков заканчиваются неуничтоженные корабли. В цикле выводится поле пользователя, поле соперника и вводится команда. Если команда является “quit”, то цикл завершается и метод возвращает GameResult::Quit. Иначе команда выполняется с помощью doCommand. Здесь же происходит обработка игровых исключений. Как только цикл завершается естественным образом, метод возвращает GameResult::TryAgain.

Метод fight отвечает за игру, состоящую из нескольких раундов. В цикле играется раунд. Если он завершился с результатом GameResult::Quit, то этот метод завершается с таким же результатам. В противном случае определяется кто выиграл. Если выиграл пользователь, то для оппонента создаются новое поле и корабли. Иначе метод завершается с результатом GameResult::TryAgain.

В функции main находится еще один цикл. В нём создается объект Game. Игроку дается выбор: загрузить игру или начать новую. Для загрузки игры используется метод Game::load. Для создания новой игры вызываются методы Game::createField и Game::createShips.

Далее с помощью метода Game::fight играется одна игра. Если её результат – GameResult::Quit – то цикл завршается. Иначе всё начинается сначала.

**Диаграмма классов.**

## Тестирование.

Сначала игроку предлагается загрузить сохранение:



## Если файла с сохранением нет, игра начинается сначала.

Пользователь вводит размеры поля и количество кораблей разного размера.



## Перемещение курсора



## 

## Атака клетки (5, 5).





## 



## Использование способностей

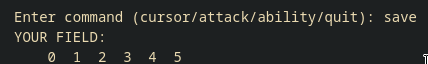
Корабль ранен.



## Корабль уничтожен, получили новую способность.



Эту способность можно использовать.

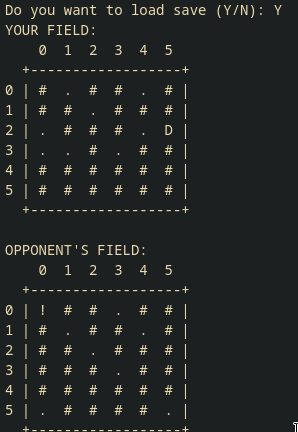


## Сохранили игру.

## Файл с сохранением:

{"opponent":{"abilities":["Scanner","Double damage","Rocket"],"cursor":{"x":1,"y":0},"double\_damage\_flag":false,"field":{"board":[[{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false}]],"height":6,"width":6},"smanager":{"ships":[{"isHorizontal":true,"length":1,"segments":["destroyed"],"x":0,"y":0},{"isHorizontal":false,"length":2,"segments":["damaged","intact"],"x":0,"y":3}]}},"player":{"abilities":[],"cursor":{"x":1,"y":4},"double\_damage\_flag":false,"field":{"board":[[{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":false},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":false},{"has\_fog":false}],[{"has\_fog":false},{"has\_fog":false},{"has\_fog":true},{"has\_fog":false},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true}],[{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true},{"has\_fog":true}]],"height":6,"width":6},"smanager":{"ships":[{"isHorizontal":true,"length":1,"segments":["intact"],"x":1,"y":4},{"isHorizontal":false,"length":2,"segments":["intact","damaged"],"x":5,"y":1}]}}}

## Загрузка этого сохранения:



## Выводы.

Были связаны между собой классы, созданные в предыдущих лабораторных работах, в полноценную игру «Морской Бой». Были реализованы графичечский интерфейс, ввод с клавиатуры и игровой цикл.