**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 |  | Горский К.Д. |
| 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.

За отрисовку игры отвечает абстракный GameRenderer. От него наследуются две реализации: GUIGameRenderer и TUIGameRenderer — графический отрисовщик игры и отрисовщик в окне терминала. У отрисовщика есть много перегрузок метода handle. Перегрузка для Field отрисовывает переданное поле, перегрузка для std::string\_view отображает текст, перегрузка для bbox2 (bounding box) отрисовывает курсор. Перегрузка для Ability отрисовывает способность. Поскольку способности — это объекты виртуального класса, невозможно правильно разрешить перегрузку в компайл-тайме. Для этого в класс способности был добавлен виртуальный метод handleBy(GameRenderer &renderer) { renderer.handle(\*this); }. Таким образом, можно выбрать перегрузку метода handle для конкретной способности.

За игру отвечает класс Game. Он содержит в себе два поля класса Player: игрок и оппонент, а также AIController — объект для управления оппонентом. В конструкторе Game принимает ссылку на GameRenderer (внедрение зависимости). Все запросы на отрисовку будет обрабатывать этот отрисовщик. За это отвечает шаблонный метод template<class T> Game::render(const T& object) { renderer.handle(object); }.

Также Game содержит указатель на текущее игровое состояние (GameState). Для обработки команд ввода у Game есть метод handle(InputMessage message). Если получено сообщение QUIT, то игра завершается, если LOAD, то производится загрузка игры из файла «data/game.sav», если SAVE, то сохранение игры в этот файл, иначе задача обработки делегируется текущему игровому состоянию.

Начальное игровое состояние — SetupFieldState: выбор размера поля.

Существуют также другие состояния, между которыми они друг вв друга переходят согласно определенным правилам:

* SetupShipsState — выбор кораблей, которые далее нужно будет расставить.
* PlaceShipsState — расстановка кораблей.
* SetupAIState — то же самое, только для оппонента, корабли он расставляет сам.
* PlayerActionState — ход игрока. Можно использовать способности. При атаке происходит переход к следующему состоянию.
* AnyKeyState — просит нажать любую кнопку и переносит к следующему состоянию.
* AIActionState — ход оппонента. Просит нажать любую кнопку и переносит к ходу игрока.

Если на AIActionState или PlayerActionState у кого-либо закончились корабли, то выводится соответствующее сообщение и меняется состояние: если выиграл игрок, то игра переходит к состоянию SetupAIState, чтобы задать новое поле для оппонента; если выиграл оппонент, то игра переходит к состоянию SetupFieldState, чтобы начать всё сначала.

Как для класса способностей, так и для класса GameState, был создан реестр (шаблон проектирования Factory). Оба этих реестра реализуют шаблонный Registry, поэтому дублирование кода минимальное. Этот реестр используется при сохранении и загрузке, описанным выше.

Для сохранения и загрузки игры в поток вывода/ввода (std::istream и std::ostream) были перегружены операторы « и ». Внутри них используется соответствующий оператор для всех полей класса (для Player и AIController) эти операторы тоже перегружены. С сохранением/загрузкой GameState всё сложнее: конкретный его тип неизвестен во время компиляции, поэтому тут оказался полезен реестр состояний. В файл сохранения записывается название способности (виртуальный метод getName у GameState) и уже после этого используется виртуальный оператор «, записывающий **конкретное** состояние в поток. При загрузке сохранения считывается название состояния и с помощью реестра находится конкретный класс с таким именем и создается. После этого используется виртуальный оператор », загружающий **конкретное** состояние из потока. Игра сохраняется в текстовом формате.

За ввод команд отвечает класс абстрактный GameInput. От него наследуются два реализации: GUIGameInput и TUIGameInput. В конструкторе принимает ссылку на игру, которой он будет посылать команды ввода. Метод handle считывает команды и передает их игре.

В GUIGameInput можно задавать настройки управления. За это отвечает класс GUIControls. Он представляет собой std::unordered\_map, где ключ — это кнопка клавиатуры, а значение — тип команды, которой эта кнопка соответствует. В конструкторе задается управление по умолчанию. С помощью оператора » можно загрузить настройки из файла. Файл настроек имеет следующий формат:

*команда1 клавиша*

*команда2 клавиша*

*…*

Настройки по умолчанию выглядят так:

*primary Return*

*secondary e*

*tertiary r*

*quit Escape*

*left a*

*right d*

*up w*

*down s*

*save F1*

*load F2*

Primary — это «главное» действие (например, разместить корабль).

Secondary — это действие второстепенной важности (например, повернуть корабль).

Tertiary — это действие третьестепенной важности (например, поставить крестик в определенной клетке поля).

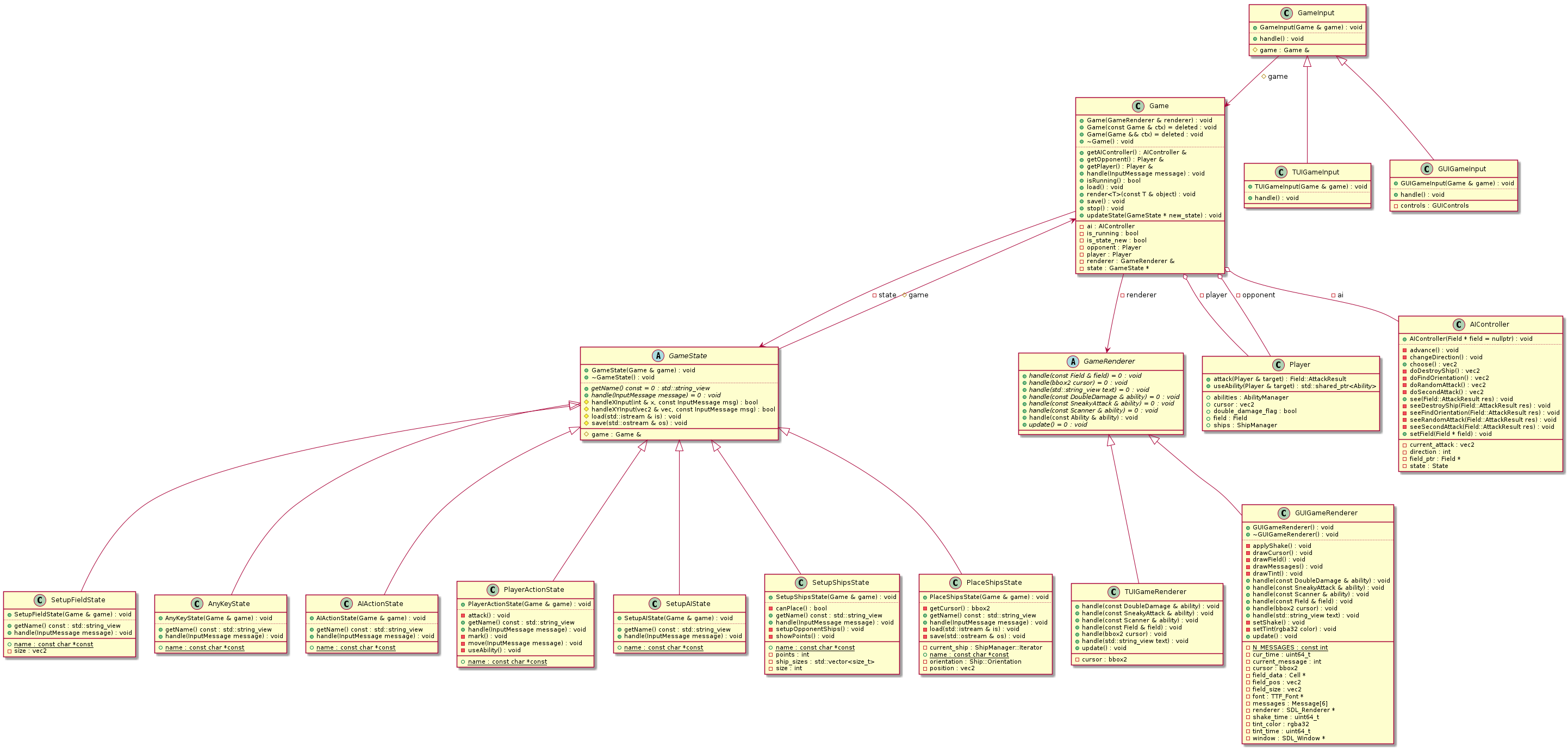
Quit — выйти из игры.

Left, right, up, don — это действия по перемещению курсора.

Save, Load — команды для сохранения/загрузки.

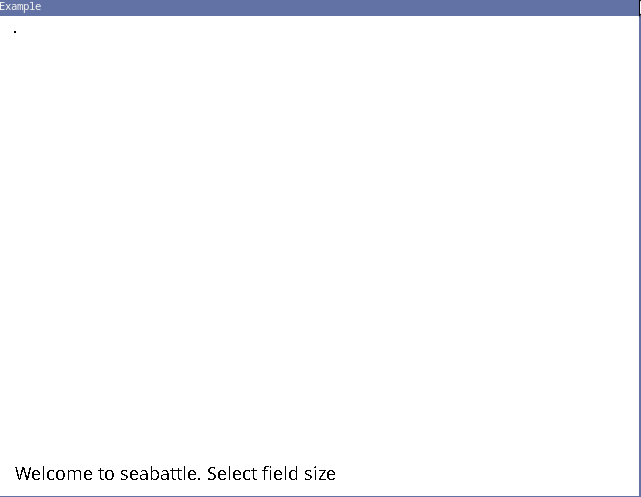
При загрузке настроек проверяется, чтобы одна кнопка отвечала только за одну команду и чтобы для всех команд были установлены кнопки.

**Диаграммы классов.**

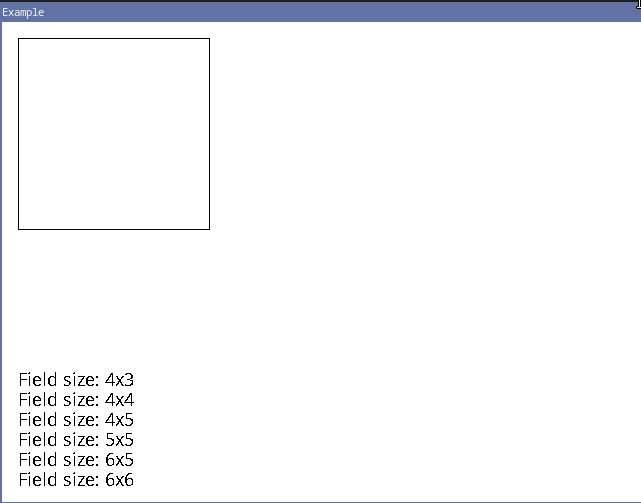


## Тестирование.

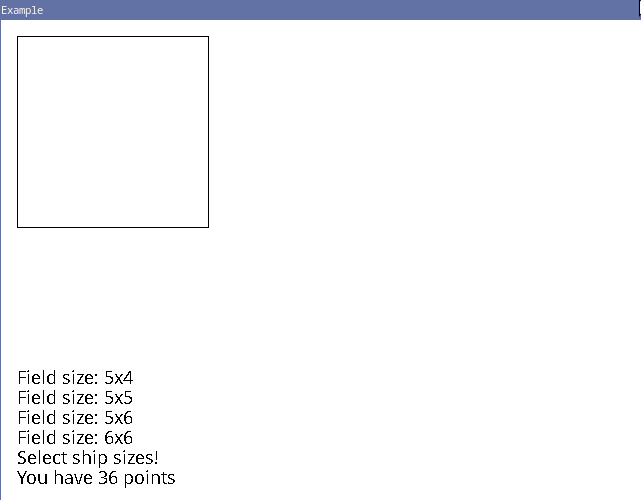
При запуске игра переходит в состояние SetupFieldState и просит задать размеры поля.



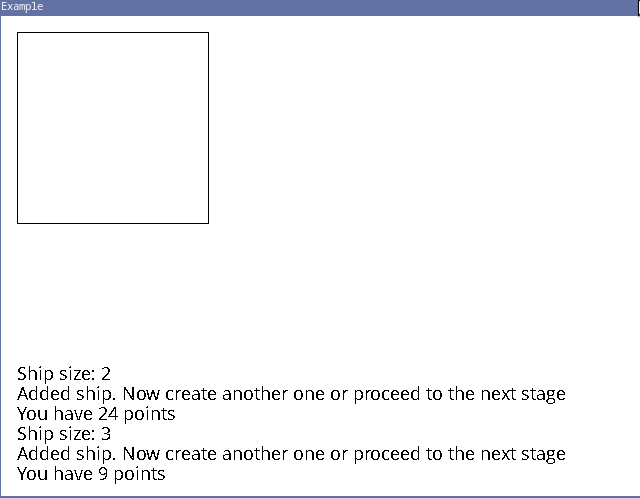
С помощью команд left, right, up, down можно задать размеры поля.



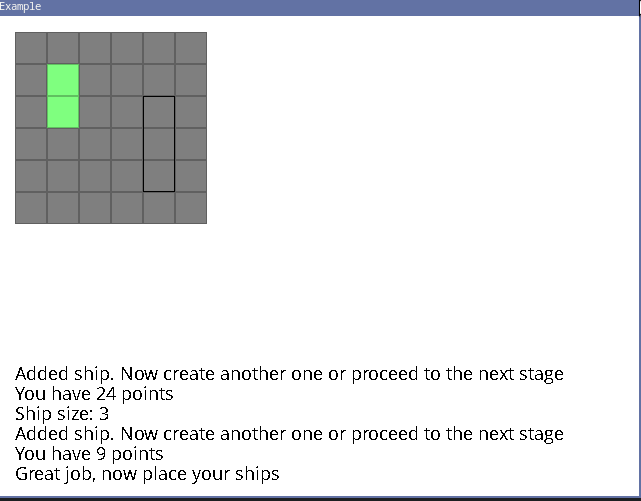
Далее игра переходит в состояние SetupShipsState



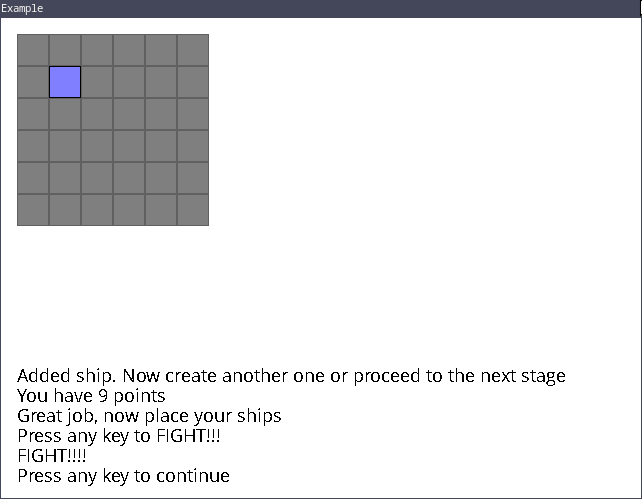
## Игрок выбрал корабль размера 2 и корабль размера 3



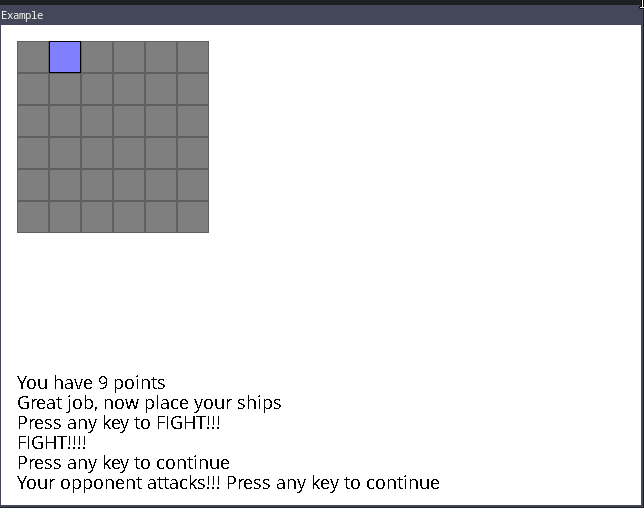
Следующее состояние — PlaceShipsState. Игрок расставляет корабли



Бой. Игрок сделал ход:



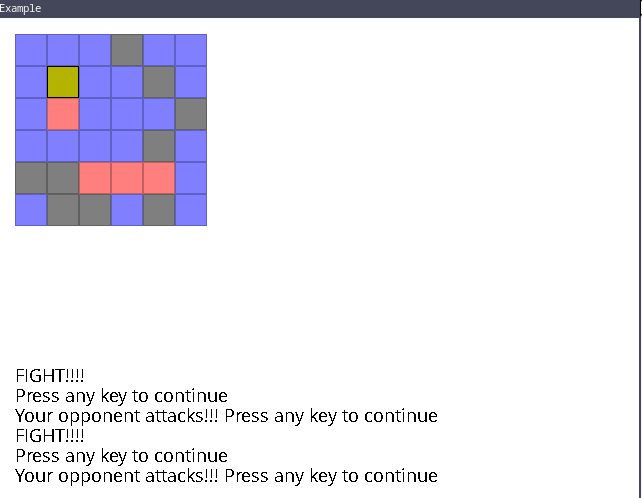
## Оппонент сделал ход:



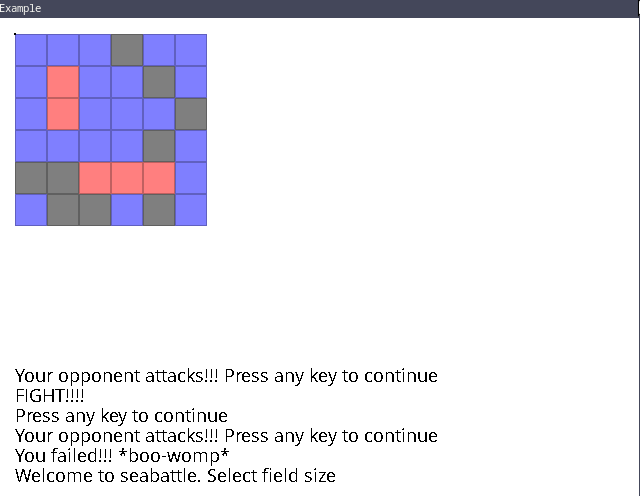
## Победа. У оппонента создается новое поле:



## Следующим ходом оппонент выиграет (поле игрока не изменилось после победы игрока):



## Игрок проиграл, игра начинается сначала:



## 

## Файл с сохранением:

2

22 222

6 6

| | | fog | | fog |

| fog ship 0 0 | | fog | fog | |

| fog ship 0 1 | fog | fog | fog | fog |

fog | fog | fog | fog | fog | fog |

fog | | fog ship 1 0 | fog ship 1 1 | fog ship 1 2 | fog |

fog | fog | fog | | fog | fog |

4 2

0

1

Sneaky Attack

2

00 212

6 6

fog | fog | fog | | fog | fog |

fog | | fog marked | fog marked | | fog |

fog | fog marked | ship 0 0 | fog marked | fog | fog |

fog | fog marked | ship 0 1 | fog marked | | fog |

| fog marked | marked | fog marked | fog | fog |

fog | fog | fog ship 1 0 | fog ship 1 1 | fog ship 1 2 | fog |

5 1

0

3

Double Damage

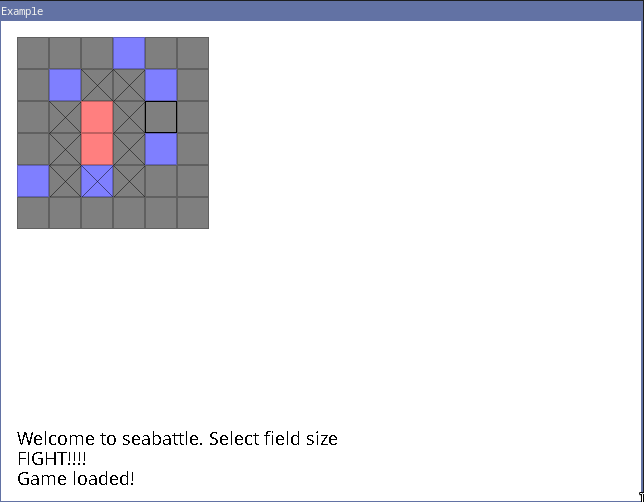
Sneaky Attack

Scanner

0 5 1 21936

PlayerAction

## 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.