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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: Шаблонные классы

Студент гр. 3342 _____ Корниенко А.Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Создать классы для управления игрой и считывания ввода пользователем, для отрисовки поля игры.

Задание

a. Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.

b. Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.

c. Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.

d. Реализовать класс, отвечающий за отрисовку поля.

Примечание:

- Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
- После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
- Для представления команды можно разработать системы классов или использовать перечисление enum.
- Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
- При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

Выполнение работы

Класс Commands хранит команды, доступные пользователю.

Методы:

std::map<std::string, std::string> get_command_map() – геттер для поля

std::map<std::string, std::string> command_map

std::vector<std::string> get_long_commands() – геттер для поля

std::vector<std::string> long_commands

Поля:

std::map<std::string, std::string> command_map – сопоставляет короткую версию команды с их полной версией

std::vector<std::string> long_commands – хранит комманды.

Класс ConsoleInputer обрабатывает ввод пользователя.

Методы:

std::string getInput() – обрабатывает ввод пользователя

std::string getCommand(const std::string& input) – проверяет на корректность команде, введённую строку

std::string getString() – методы для ввода строки

Поля:

Input input_metod – класс, определяющий метод ввода.

Commands& commands – команды.

Класс ConsolePrinterMessage определяет вывод данных (в консоль)

Методы:

void print(std::string string) – определяет вывод.

Класс ConsoleShow_Table определяет вывод поля в консоль

Методы:

void showTable(Table& table, bool flag) – выводит поле

void showShips(ManagerShips& ship_manager) – выводит состояния корабли

Класс Controller преобразует ввод пользователя в команды и выполняет при помощи их. Наследуется от IController.

Методы:

void initialize_commands() – инициализует команды

virtual int setMode() – используется для того, чтобы задать начало игры (новая или загрузка игры), и конца игры (новая игра или закончить)

std::string setFilename() – задаёт имя файла для сохранения

std::vector<int> setPlaceship(int len_ship) – задаёт расположение корабля

Coord setCoord() – задаёт координату для атаки или для способности (при необходимости)

void Endgame() – конец игры

void run() – начало игры

void print_message(std::string string) – вывод сообщения через класс, который вывод информацию в консоль

Поля:

Commands list_commands - команды

Game &game - игра

Inputer inputer – класс, который обрабатывает ввод пользователя

View<Printer_Table, Printer_Message> view – классы вывода информации об игре

std::map<std::string, std::function<void()>> commands – сопоставляет команды с функциями, которые должны выполняться при их вводе

bool isGameRunning – хранит true, когда игра находится в цикле, false при конце игры

Класс Input определяет ввод пользователя

Методы:

std::string input() – определяет ввод пользователя

Класс Printer_Message отвечает за вывод сообщений

Методы:

void attackMessage(Game &game) – сообщение об итоге атаки

void useSkill(Game &game) – сообщение о применение способности

void nextSkill(Game &game) – сообщение об информации о следующей доступной способности

void help() – сообщение об информации о доступных командах

void print_message(std::string message) – вывод сообщения

Поля:

ConsolePrinterMessage printer – определяет вывод информации

Commands& commands – команды

Класс Printer_Table выводит информацию о полях

Методы:

void drawGameTable(Game &game) – рисует поле игрока и бота

void showShips(Game &game, bool flag) – выводит состояния кораблей

void showTableShips(Game &game, bool flag) – выводит поле и состояния кораблей

void showTable(Game &game, bool flag) – выводит поле

Поля:

ConsoleShow_Table show – выводит поля

Класс View принимает команды от контроллера и выводит информацию об игре.

Методы:

void solve(std::string command) – принимает команду и выводит, что необходимо

void print_message(std::string string) – выводит сообщение

Поля:

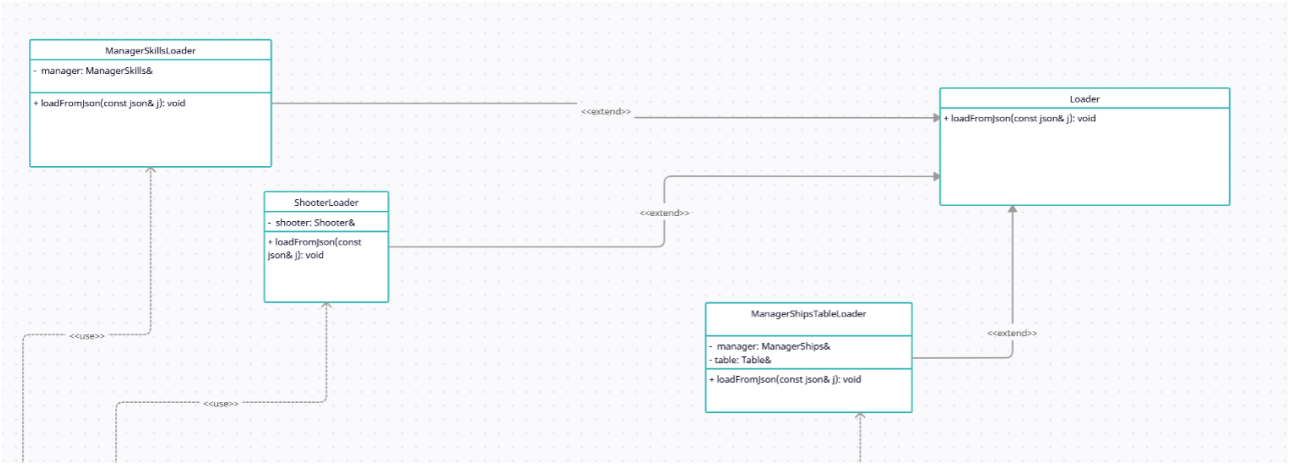
Commands& list_commands - команды

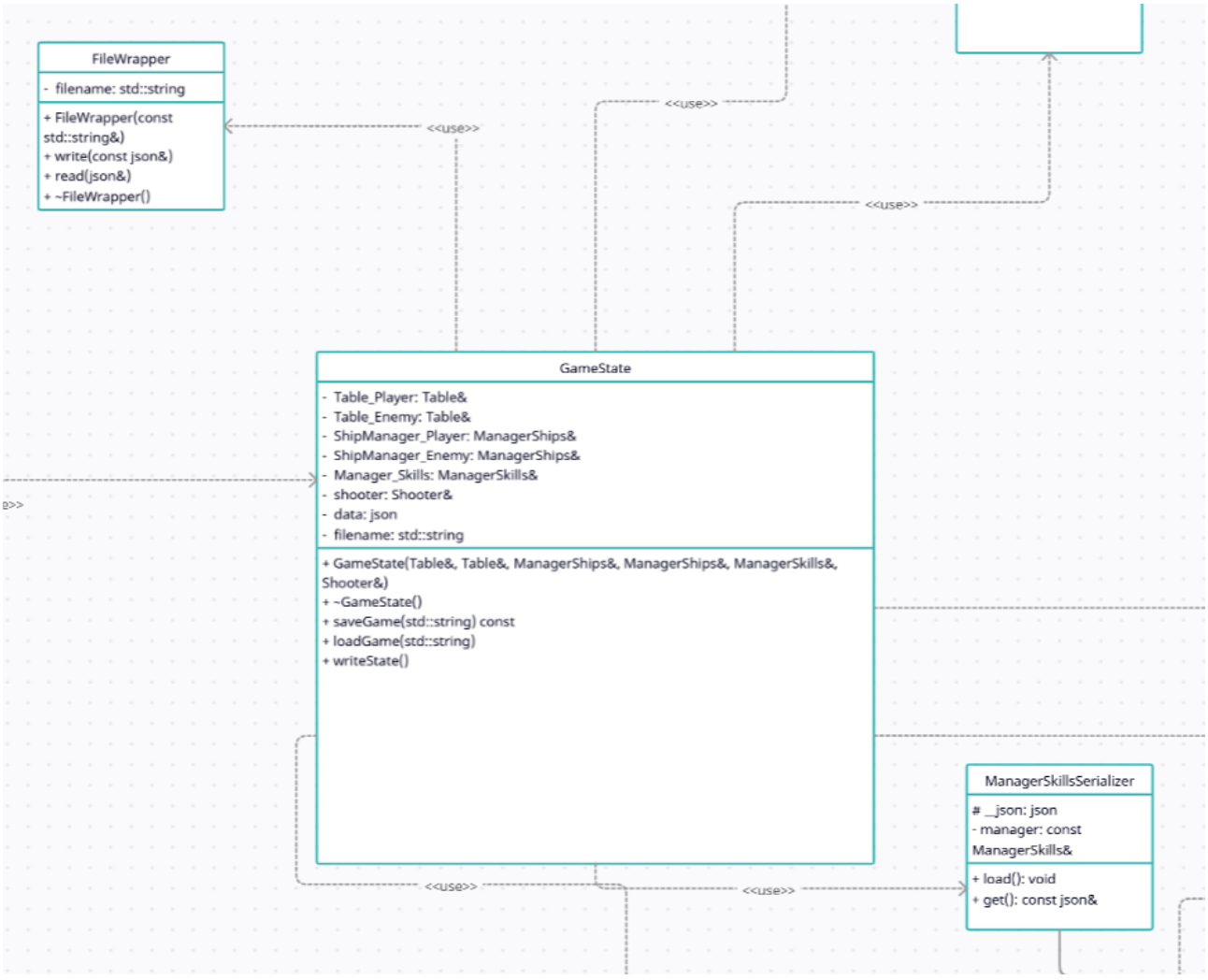
Printer_Table printer_table – выводит информацию о поле

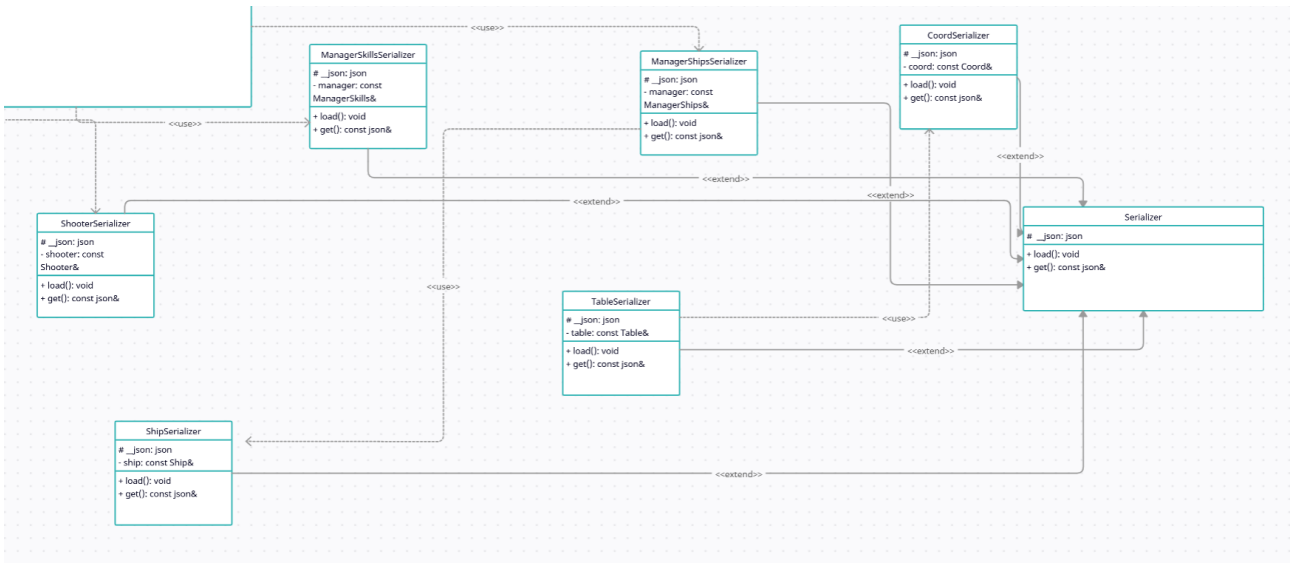
Printer_Message printer_message – выводит сообщения

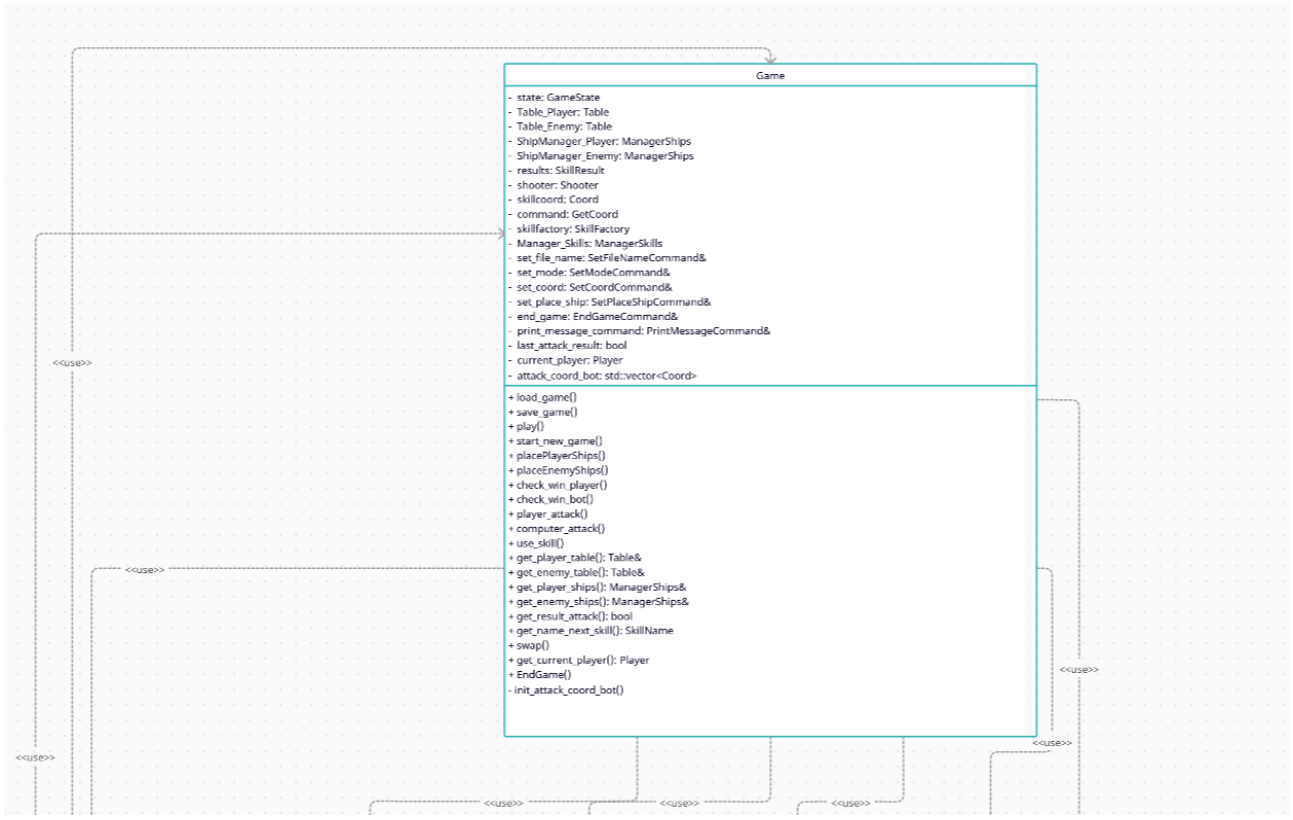
Game& game - игра

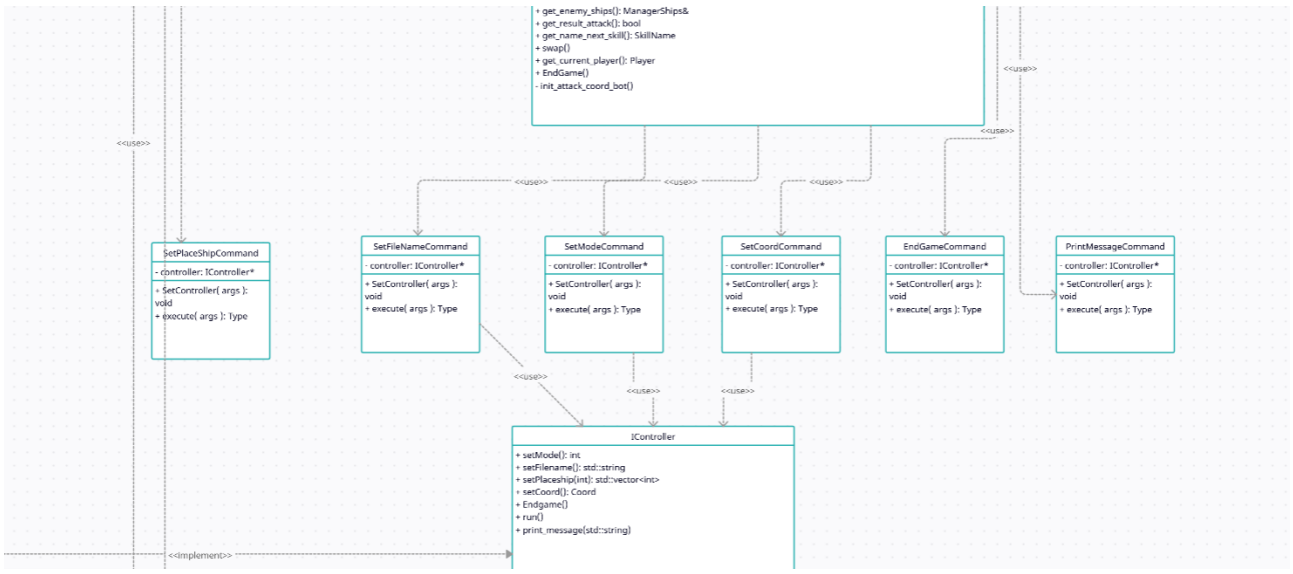
UML-диаграммы классов:

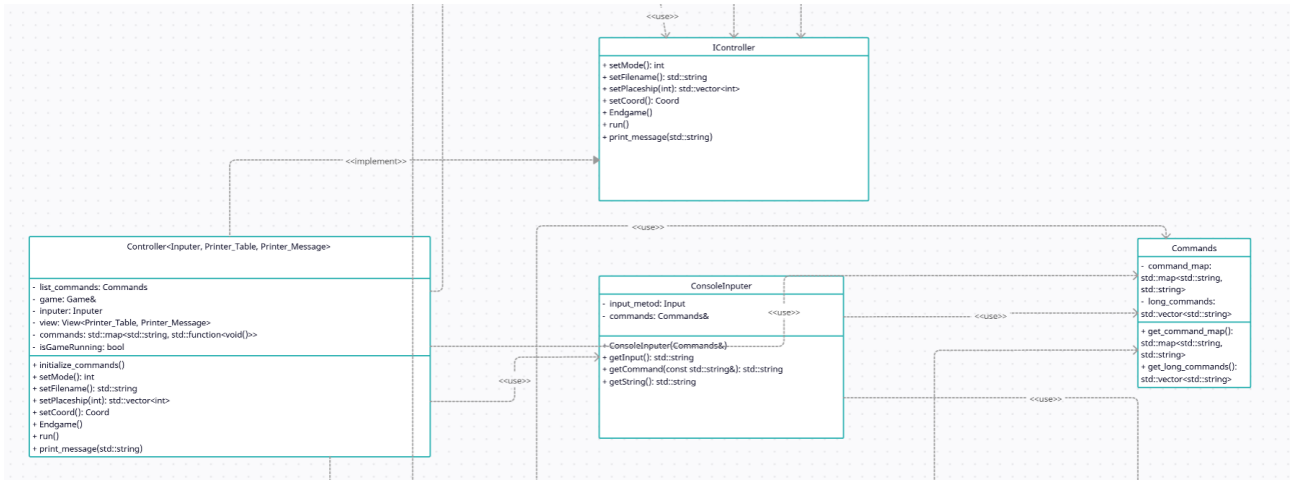


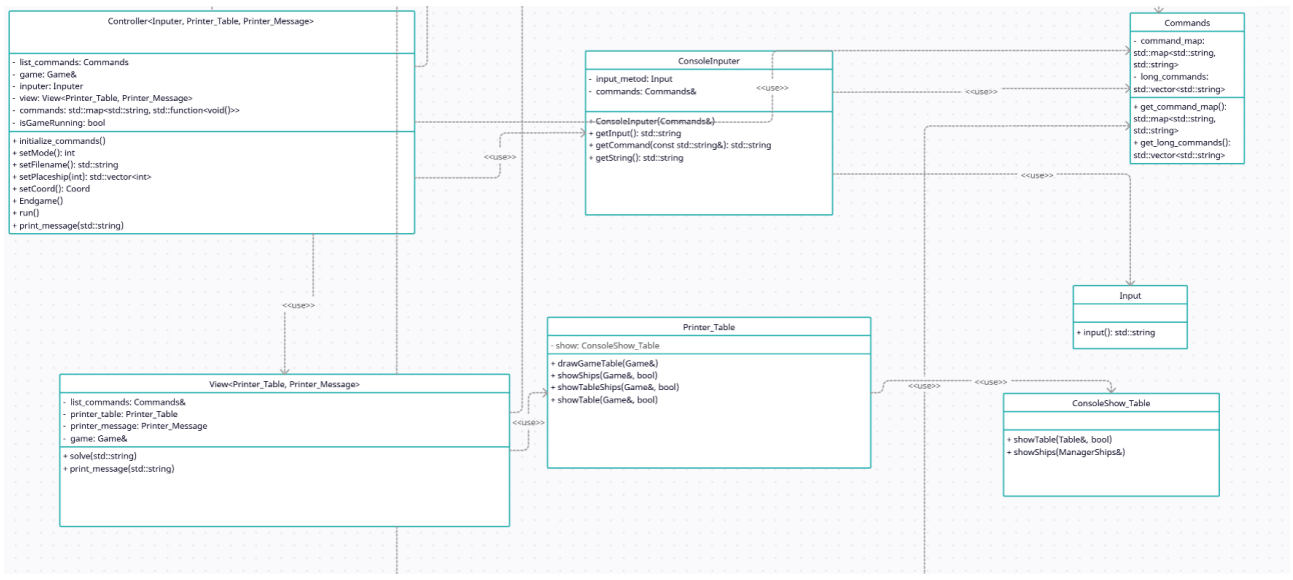


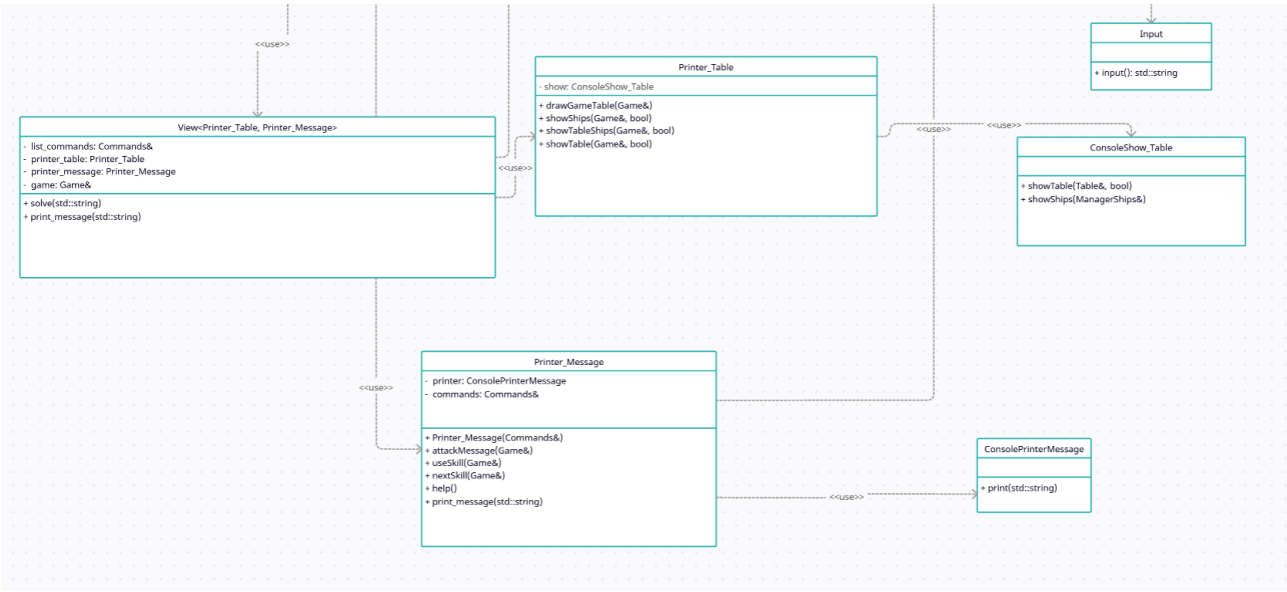

ManagerSkillsSerializer
__json: json
- manager: const ManagerSkills&
+ load(): void
+ get(): const json&
ManagerShipsSerializer
__json: json
- manager: const ManagerShips&
+ load(): void
+ get(): const json&
CoordSerializer
__json: json
- coord: const Coord&
+ load(): void
+ get(): const json&
ShooterSerializer
__json: json
- shooter: const Shooter&
+ load(): void
+ get(): const json&
Serializer
__json: json
+ load(): void
+ get(): const json&
TableSerializer
__json: json
- table: const Table&
+ load(): void
+ get(): const json&
ShipSerializer
__json: json
- ship: const Ship&
+ load(): void
+ get(): const json&
<<use>>
<<extend>>


Game
- state: GameState
- Table_Player: Table
- Table_Enemy: Table
- ShipManager_Player: ManagerShips
- ShipManager_Enemy: ManagerShips
- results: SkillResult
- shooter: Shooter
- skillcoord: Coord
- command: GetCoord
- skillfactory: SkillFactory
- Manager_Skills: ManagerSkills
- set_file_name: SetFileNameCommand&
- set_mode: SetModeCommand&
- set_coord: SetCoordCommand&
- set_place_ship: SetPlaceShipCommand&
- end_game: EndGameCommand&
- print_message_command: PrintMessageCommand&
- last_attack_result: bool
- current_player: Player
- attack_coord_bot: std::vector<Coord>
+ load_game()
+ save_game()
+ play()
+ start_new_game()
+ placePlayerShips()
+ placeEnemyShips()
+ check_win_player()
+ check_win_bot()
+ player_attack()
+ computer_attack()
+ use_skill()
+ get_player_table(): Table&
+ get_enemy_table(): Table&
+ get_player_ships(): ManagerShips&
+ get_enemy_ships(): ManagerShips&
+ get_result_attack(): bool
+ get_name_next_skill(): SkillName
+ swap()
+ get_current_player(): Player
+ EndGame()
- init_attack_coord_bot()
<<use>>


+ get_enemy_ships(): ManagerShips&
+ get_result_attack(): bool
+ get_name_next_skill(): SkillName
+ swap()
+ get_current_player(): Player
+ EndGame()
- init_attack_coord_bot()
<<use>>
SetPlaceShipCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
SetFileNameCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
SetModeCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
SetCoordCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
EndGameCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
PrintMessageCommand
- controller: IController*
+ SetController(args): void
+ execute(args): Type
IController
+ setMode(): int
+ setFilename(): std::string
+ setPlaceship(int): std::vector<int>
+ setCoord(): Coord
+ Endgame()
+ run()
+ print_message(std::string)
<<implement>>


IController
+ setMode(): int
+ setFilename(): std::string
+ setPlaceship(int): std::vector<int>
+ setCoord(): Coord
+ Endgame()
+ run()
+ print_message(std::string)
<<implement>>
Controller<Inputer, Printer_Table, Printer_Message>
- list_commands: Commands
- game: Game&
- inputer: Inputer
- view: View<Printer_Table, Printer_Message>
- commands: std::map<std::string, std::function<void()>>
- isGameRunning: bool
+ initialize_commands()
+ setMode(): int
+ setFilename(): std::string
+ setPlaceship(int): std::vector<int>
+ setCoord(): Coord
+ Endgame()
+ run()
+ print_message(std::string)
ConsoleInputer
- input_metod: Input
- commands: Commands&
+ ConsoleInputer(Commands&)
+ getInput(): std::string
+ getCommand(const std::string&): std::string
+ getString(): std::string
Commands
- command_map: std::map<std::string, std::string>
- long_commands: std::vector<std::string>
+ get_command_map(): std::map<std::string, std::string>
+ get_long_commands(): std::vector<std::string>
<<use>>


Controller<Inputer, Printer_Table, Printer_Message>
- list_commands: Commands
- game: Game&
- inputer: Inputer
- view: View<Printer_Table, Printer_Message>
- commands: std::map<std::string, std::function<void()>>
- isGameRunning: bool
+ initialize_commands()
+ setMode(): int
+ setFilename(): std::string
+ setPlaceship(int): std::vector<int>
+ setCoord(): Coord
+ Endgame()
+ run()
+ print_message(std::string)
ConsoleInputer
- input_metod: Input
- commands: Commands&
+ ConsoleInputer(Commands&)
+ getInput(): std::string
+ getCommand(const std::string&): std::string
+ getString(): std::string
Commands
- command_map: std::map<std::string, std::string>
- long_commands: std::vector<std::string>
+ get_command_map(): std::map<std::string, std::string>
+ get_long_commands(): std::vector<std::string>
Input
+ input(): std::string
Printer_Table
- show: ConsoleShow_Table
+ drawGameTable(Game&)
+ showShips(Game&, bool)
+ showTableShips(Game&, bool)
+ showTable(Game&, bool)
View<Printer_Table, Printer_Message>
- list_commands: Commands&
- printer_table: Printer_Table
- printer_message: Printer_Message
- game: Game&
+ solve(std::string)
+ print_message(std::string)
ConsoleShow_Table
+ showTable(Table&, bool)
+ showShips(ManagerShips&)
<<use>>


Input
+ input(): std::string
Printer_Table
- show: ConsoleShow_Table
+ drawGameTable(Game&)
+ showShips(Game&, bool)
+ showTableShips(Game&, bool)
+ showTable(Game&, bool)
ConsoleShow_Table
+ showTable(Table&, bool)
+ showShips(ManagerShips&)
View<Printer_Table, Printer_Message>
- list_commands: Commands&
- printer_table: Printer_Table
- printer_message: Printer_Message
- game: Game&
+ solve(std::string)
+ print_message(std::string)
Printer_Message
- printer: ConsolePrinterMessage
- commands: Commands&
+ Printer_Message(Commands&)
+ attackMessage(Game&)
+ useSkill(Game&)
+ nextSkill(Game&)
+ help()
+ print_message(std::string)
ConsolePrinterMessage
+ print(std::string)
<<use>>

Выводы

В ходе выполнения работы были разработаны классы для управления игрой и считывания ввода пользователем, для отрисовки поля игры.