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, управление

Студент гр. 0381 _____ Прохоров Б.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить основы объектно-ориентированного программирования на языке C++.

Задание.

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и т.д.). Затем определить класс игры, которое параметризируется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требования:

- Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
- Создан шаблонный класс игры, который параметризируется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и т.д.

Выполнение работы.

Чтобы выполнить задание, нужно было реализовать 3 шаблонных класса: *Rule_1*, *Rule_2*, *Rule_3* и немного модифицировать классы *Presenter*, *Model* и *Player*.

Класс *Presenter*, представляющий собой шаблонный класс игры, является прослойкой между бизнес-логикой и командами управления, т.е. непосредственное изменение состояния игрой должно проходить через этот класс. *Presenter* параметризируется конкретными правилами. Класс игры проводит управление врагами, передачей хода и передаёт информацию куда переместить игрока. Теперь *Presenter* при инициализации принимает параметры правил *Rule_1*, *Rule_2* и *Rule_3*, параметр шаблона правил определяет конкретные значения в правилах.

Rule_1 – шаблонный класс правил, который задаёт правило о максимальном количестве столкновений игрока и стен. При инициализации можно задать количество столкновений, а можно и не задавать, тогда будет выставлено значение по умолчанию, равное пяти.

Rules_2 – шаблонный класс правил, который задаёт правило о минимальном количестве врагов, которых надо убить. При инициализации можно задать это количество, а можно и не задавать, тогда будет выставлено значение по умолчанию, равное нулю. Хоть количество врагов, которых надо убить и задаётся в параметрах, во избежание ошибок, были реализованы геттер и сеттер по количеству.

Rules_3 – шаблонный класс правил, который задаёт правило о минимальном количестве вещей, которые надо подобрать. При инициализации можно задать это количество, а можно и не задавать, тогда будет выставлено значение по умолчанию, равное нулю. Хоть количество вещей, которые надо подобрать и задаётся в параметрах, во избежание ошибок, были реализованы геттер и сеттер по количеству.

У классов правил есть общий метод *check*, который выполняет проверку, выполнено ли правило, игра не завершится, пока не будут соблюдены все правила (либо нажата клавиша “ESC” – экстренный выход из игры).

В классе *Player* необходимо было добавить поле *collisions* для того, чтобы было удобно считать количество столкновений игрока со стеной.

В классе *Model*, представляющим собой модель, необходимо было изменить метод *game_over*, геттеры количества вещей и врагов, как использованных / убитых, так и сколько их было в начале для корректной работы классов правил.

Реализован паттерн MVP – см. классы Model, View, Presenter и приложение А.

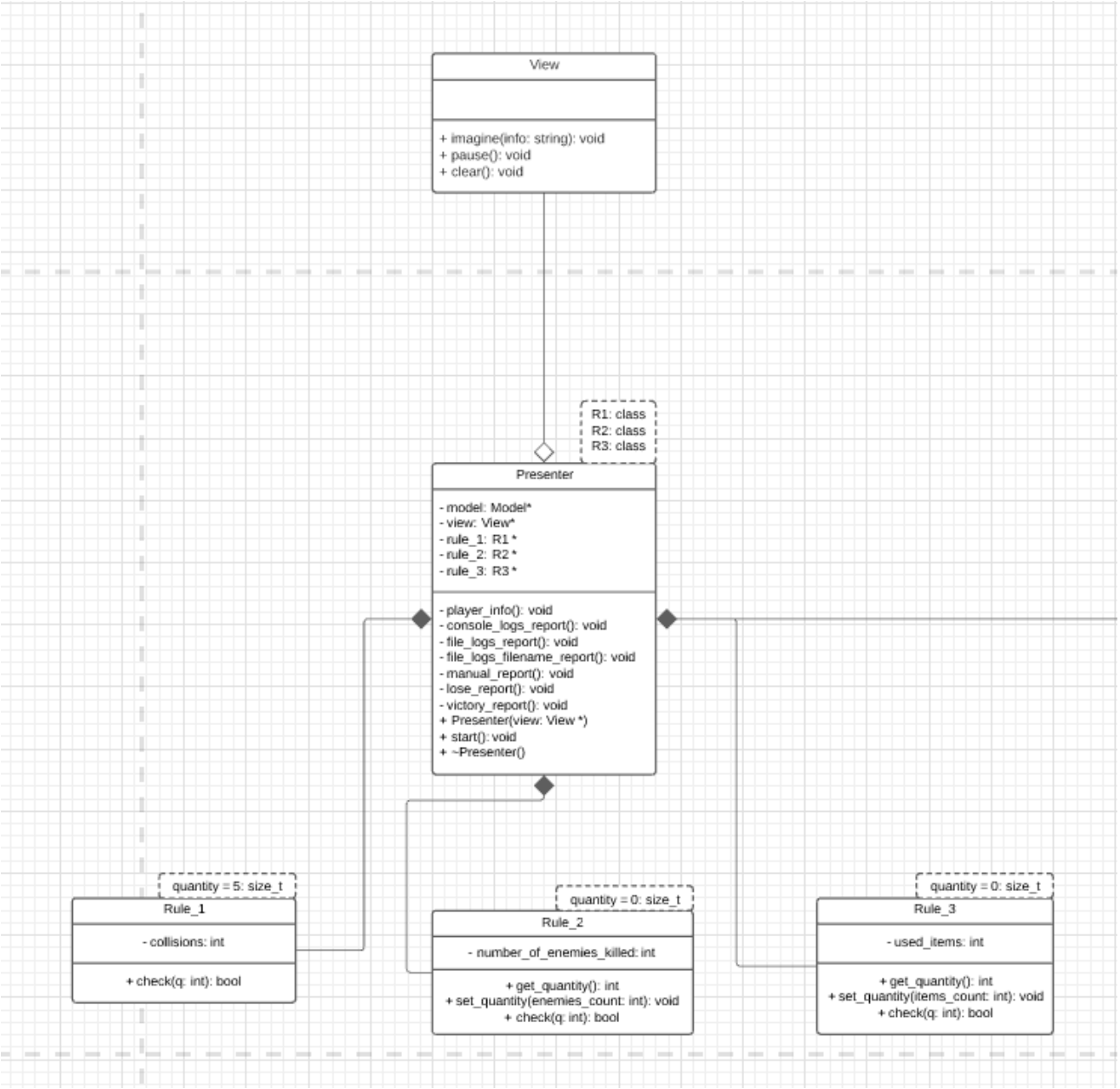
Реализован паттерн Singleton – см. класс Logger и приложение А.

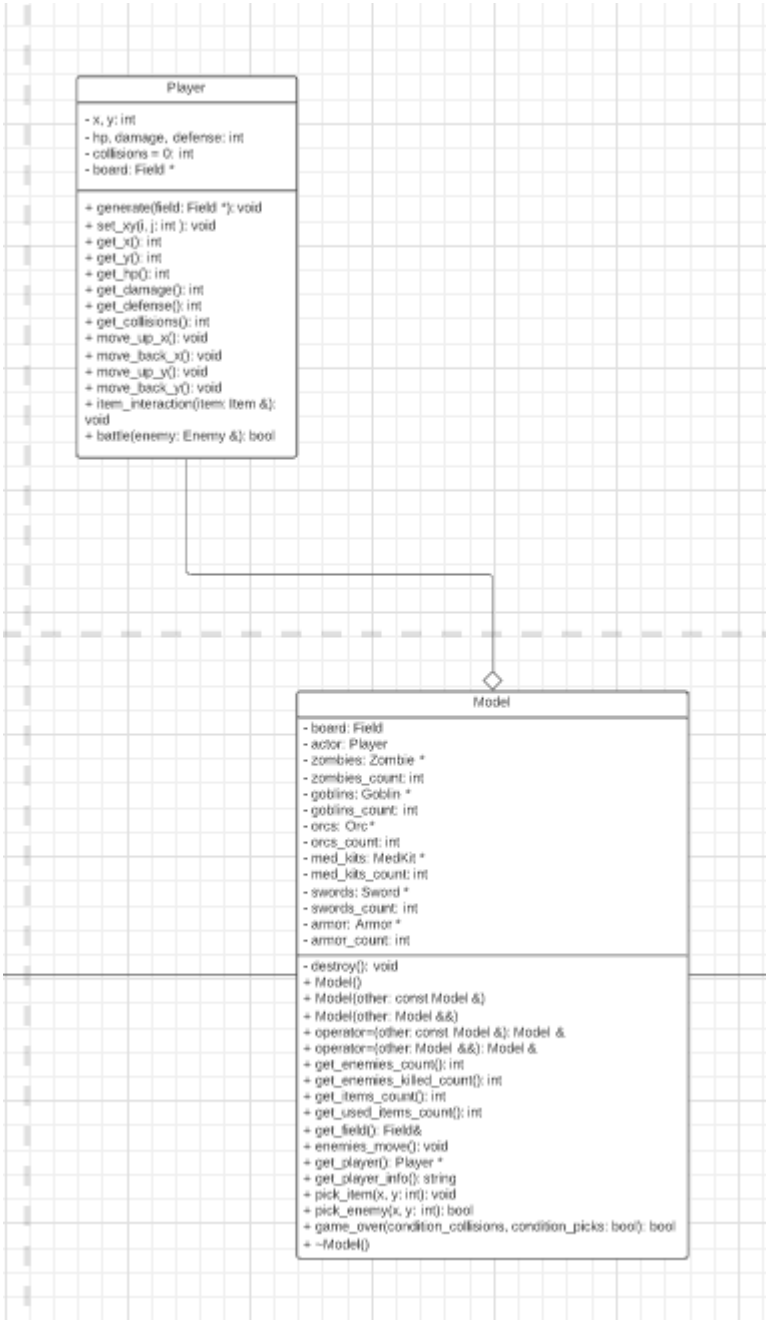
Реализован паттерн Observer – см. класс Logger и приложение А.

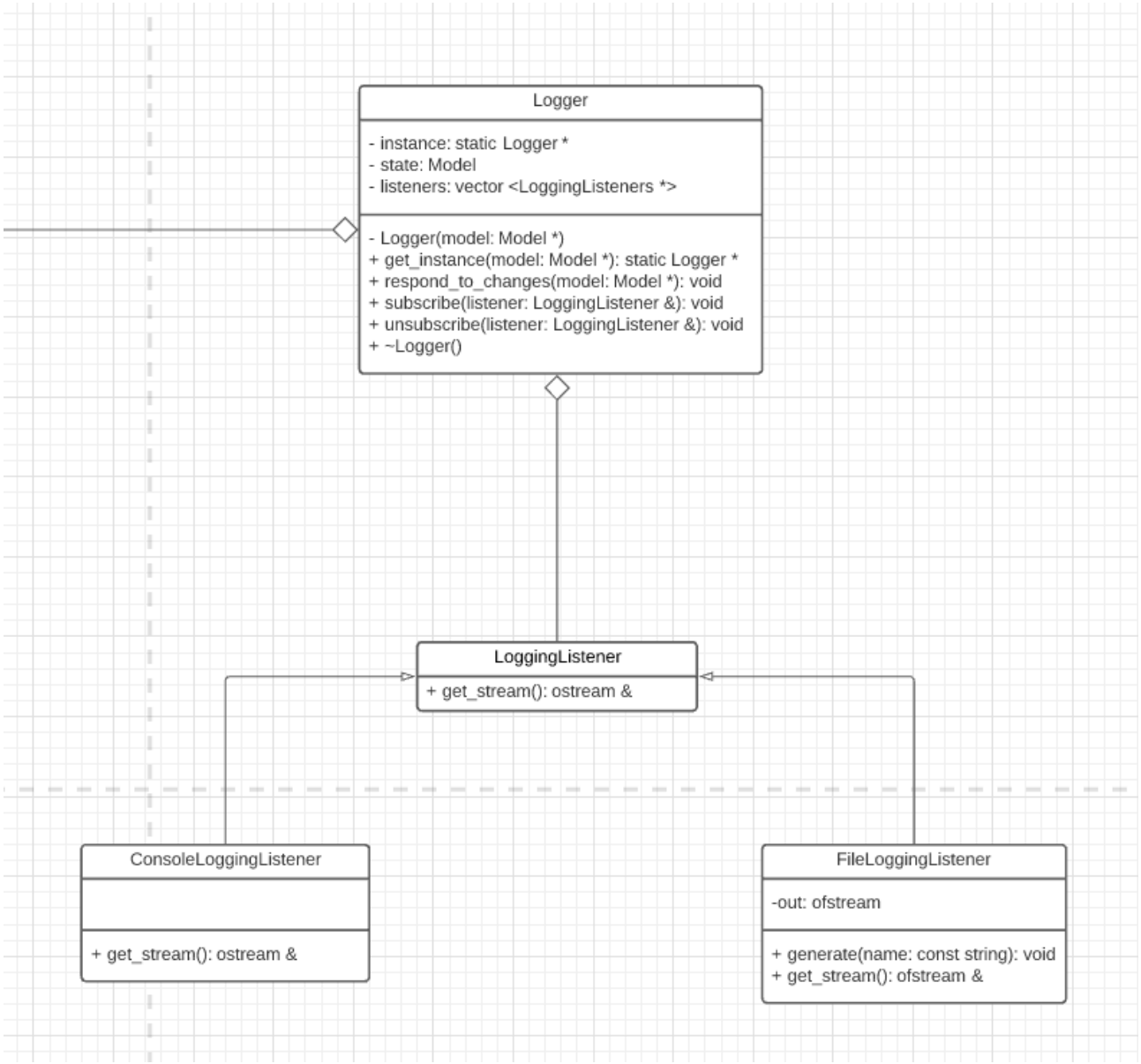
Выводы.

Были изучены основы объектно-ориентированного программирования на языке C++.

ПРИЛОЖЕНИЕ А
UML ДИАГРАММА




Player
- x, y: int
- hp, damage, defense: int
- collisions = 0: int
- board: Field *
+ generate(field: Field *): void
+ set_xy(i, j: int): void
+ get_x(): int
+ get_y(): int
+ get_hp(): int
+ get_damage(): int
+ get_defense(): int
+ get_collisions(): int
+ move_up_x(): void
+ move_back_x(): void
+ move_up_y(): void
+ move_back_y(): void
+ item_interaction(item: Item &): void
+ battle(enemy: Enemy &): bool
Model
- board: Field
- actor: Player
- zombies: Zombie *
- zombies_count: int
- goblins: Goblin *
- goblins_count: int
- orcs: Orc*
- orcs_count: int
- med_kits: MedKit *
- med_kits_count: int
- swords: Sword *
- swords_count: int
- armor: Armor *
- armor_count: int
- destroy(): void
+ Model()
+ Model(other: const Model &)
+ Model(other: Model &&)
+ operator=(other: const Model &): Model &
+ operator=(other: Model &&): Model &
+ get_enemies_count(): int
+ get_enemies_killed_count(): int
+ get_items_count(): int
+ get_used_items_count(): int
+ get_field(): Field&
+ enemies_move(): void
+ get_player(): Player *
+ get_player_info(): string
+ pick_item(x, y: int): void
+ pick_enemy(x, y: int): bool
+ game_over(condition_collisions, condition_picks: bool): bool
+ ~Model()



Ссылка: http://surl.li/audlr