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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Объектно-ориентированное программирование»
Тема: Полиморфизм

Студент гр. 3384 _____ Поздеев В.Д

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург
2024

Цель работы.

Изучить концепцию полиморфизма классов. Получить полученные знания на практики путем создания способностей для игры из предыдущей лабораторной работы.

Задание.

Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

Попытка применить способность, когда их нет

Размещение корабля вплотную или на пересечении с другим кораблем

Атака за границы поля

Примечания:

Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс

Не должно быть явных проверок на тип данных.

Выполнение работы.

Были измененый классы Ship, playField, shipManager. Теперь playField и shipManager стали независимыми.

Создан класс vector2d, который хранит в себе класс point2d, а также box2d. Есть пустой конструктор, который инициализирует поля нулями. Первое представляет собой точку на плоскости. В данном классе хранится поля x, y, которые представляют собой целочисленные координаты. Также все поля данного класса являются публичными. Конструктор класса принимает две целочисленные переменные и записывает из поля. Также конструктор перегружен и может принимать тип std::pair. first, second записываются в x, y соответственно. Конструктор копирования записывает поля одного объекта во второй объект. У класса перегруженны оператор присваивания(копирования и перемещения), сложения, вычетания, деления, умножения, +=, -=. Оператор присваивания записывает поля одного объекта в поля другого объекта(или перемещает). Оператор сложения складывает поля x и y у двух объектов. Вычетание вычитает, умножение умножает каждую координаты на число, деление делит координаты на число. += добавляет к координатам координаты объекта, -= вычетает.

В классе box2d все поля публичные. Он хранит координату нижней левой точки point2d min_point, и координату верхней правой точки point2d max_point. Пустой конструктор инициализирует поля нулями. Обычный конструктор принимает две точки и записывает их в поля. Конструктор копирования записывает две точки с одного объекта в другой объект. Конструктор перемещения перемещает эти точки из одного объекта в другой, с помощью std::move. Создан оператор присваивания копирования и перемещения, который записывает в поля одного объекта поля другого объекта(либо перемещает).

Создан метод contains, который принимает точку и выводит true или false, в зависимости если точка лежит между двумя точками(min_point, max_point).

Данный метод contains перегружен и может принимать объект типа box2d. Внутри он проверяет чтобы две точки(min_point, max_point) лежали между двумя точками(min_point, max_point) изначального объекта.

Создан метод intersects, который принимает объект типа box2d. И возвращает true либо false в соответствии, если одна из точек лежит между двумя точками изначального объекта.

Создан класс Segment. Внутри есть поле segmentState state, который является публичным. Внутри класса объявлен enum segmentState, который содержит состояния сегмента(normal, damaged, destroyed).

У класса Segment пустой конструктор, который инициализирует поле state(normal).

У класса Segment создан метод Attack, который изменяет состояние сегмента соответственно normal→damaged→destroyed, с помощью конструкции switch-case.

Класс Ship, теперь содержит в себе вектор(std::vector) указателей(std::shared_ptr) на объекты типа Segment. Также содержит поле bool is_vertical, box2d area(представляет собой координаты первого сегмента и последнего сегмента), int length. Создан default конструктор.

Конструктор принимает длинну корабля точку где он должен быть, а также ориетацию корабля, которая определяется is_vertical. Длинна и положение сразу записываются в поля. Внутри проверяется, чтобы длинна корабля была между 1 и 4, иначе вызывается объект класса invalidShipLength(length). Создается переменная области корабля в первую точку которой записывается координата подаваемая в конструктор, во вторую точку записывается соответственно изначальная координата с прибавкой в соответствии с положением корабля(либо x, либо y). После этого циклом от 0 до length в вектор сегментов добавляются объекты типа std::make_shared().

Создан конструктор копирования и оператор присваивания копирования. Они копируют каждое поле одного объекта в каждое поле изначального объекта. Создан оператор присваивания перемещения, который перемещает поля из одного объекта в изначальный.

Созданы геттеры getLen(), getArea(), IsVertical(), а также getSegments(). Каждый из методов возващает копию на поле, но getSegments, возвращает ссылку на сегменты.

Создан метод isDestroyed(), который проверяет уничтожен ли корабль, проходя циклом по сегментам корабля и проверяя их состояние. Если один не уничтожен значит корабль не уничтожен.

Объект invalidShipLength представляет собой ошибку. В нем хранится поле message, которое хранит в себе текст ошибки, который создается в конструкторе данного класса. С помощью метода what() можно вернуть из этого класса строку ошибки. Данный класс наследуется от std::exception.

В классе shipManager есть два поля вектор указателй(std::shared_ptr) на корабли ships, и количество уничтоженных кораблей int destroyed_ships. Создан default конструктор. Создан конструктор копирования и оператор присваивания копирования, которые копируют каждое поле одного объекта в изначальный.

Созданы геттер getLen(), который возвращает длинну вектора ships.

Создан метод addShip, который принимает указатель(std::shared_ptr) на корабль и добавляет его в вектор указателей.

Создан метод shipInterection, который принмает область корабля box2d ship_area. Внутри проходится по вектору кораблей циклом. В каждой итерации достается область текущего корабля. Минимальная точка сдвигается на 1 по диагонали вниз(x, y уменьшаются на 1) , максимальния на 1 по диагонали вверх. Далее смотрится значение функции area.intersects(ship_area), и если есть пересечение двух областей, то возвращается true. После конца цикла возвращается false.

Создан метод checkDestroyedShips, который внутри считает количество уничтоженных кораблей, прохадясь циклом по вектору. После смотри увеличилось ли количество уничтоженных кораблей и изменяет поле destroyed_ships, если изменилось. Соответственно данный метод возвращает true, false в соответствии если количество кораблей изменилось.

Создан метод allShipsDestroyed, который внутри проверяет количество кораблей и количество уничтоженных кораблей. Если все корабли уничтоженный то возвращается true, иначе false.

Создан класс Cell. Все поля публичные. Внутри создан enum cellState, который хранит состояния клетки поля(unknown, empty, ship). Внутри хранятся поля: указатель(std::shared_ptr) segment(указатель на сегмент корабля) и cellState state, в которой хранится состояние клетки.

Создан конструктор который инициализирует указатель nullptr, a state unknown.

Создан метод Attack(bool change_state), который изменяет состояние сегмента корабля и состояние клетки(если это нужно). Внутри создается условие, что указатель на сегмент является нулевым(nullptr). Если это так, то проверяется, change_state. Если это условие тоже выполняется, то меняется состояние клетки.

Иначе если нужно поменять состояние клетки, то состояние меняется. Вызывается метод у сегмента Attack.

Класс playField, содержит поля box2d area(область игрового поля), а также двойной массив клеток(vector<vector<Cell>>). Создан default конструктор. Создан конструктор, который принимает int size_x, int size_y. Внутри проверяется чтобы size_x и size_y были больше или равны 1. Иначе вызывается ошибка invalidFieldSize(). Далее создается box2d area, первая точка которого инициализируется нулями, а вторая size_x, size_y. Далее это записывается в поле area. После этого вызывается у поля field вызывается метод

field.resize(size_y, std::vector<Cell>(size_x)), который создает двойной массив нужного поля.

Конструктор может принимать point2d size, засчет перегрузки. Внутри происходит тоже самое, написанное выше.

Созданы конструктор копирования, перемещения, которые копируют или соответственно перемещают каждое поле класса playField.

Созданы операторы присваивания копирования и перемещения, которые копируют или соответственно присваивают каждое поле класса.

Созданы методы геттеры getArea, getCell, которые возвращают копии области поля и соответственно конкретной клетки по координате.

Создан метод placeShip, который ставит корабль по нужной координате, нужной длинны, определённой ориентации. Данный метод принимает умный указатель на корабль ship и ссылку на менеджер кораблей ship_manager. Внутри достается область корабля с помощью ship→getArea(). После этого проверяется, что подаваемый корабль содержится в области(area.contains(ship_area)) и не пересикается не с одним из кораблей в ship_manager(ship_manager.shipIntersection(ship_area)). Далее в ship_manager добавляется корабль, с помощью addShip(ship). После этого достается из корабля ссылка на вектор сегментов корабля и в соответствии с положением корабля. Если он вертикальный то циклом по координате y в клетки Cell записываются указатели на сегменты корабля. Иначе по координате y с заданной точки в Cell записываются указатели на сегмент корабля.

В конце если корабль пересикается, либо выходит за пределы вызываются объекты ошибок invalidShipPosition, либо objectOutOfBounds(ship→getArea().min_point) соответственно.

Создан метод Attack, который принимает координаты point2d coordinates и bool not_sneak(скрытая атака). Внутри проверятся, чтобы координата находилась в области поля, в противном случае вызывается ошибка objectOutOfBounds(). Затем по индексу двойного массива у Cell вызывается метод Attack(not_sneak).

Создан интерфейс класс IAbility. У него созданы 2 виртуальныйх метода apply() и setPlayer(Player * player). Также виртуальный деструктор virtual ~IAbility(){}.

Создан класс inputManager, который хранит в себе поле ссылку на std::isteram & is.

Конструктор класса принимает ссылку на std::istream & is и заисывает её в поле.

Создан метод inputCoordinates, который принимает ссылку на объект типа point2d & coordinates и записывает в него через ввод координаты. Внутри создается строка std::string input. Затем вызывается getline(), который принимает поток данных is, и строку в которую данный поток вводится. Далее создаются переменные x, y в которые записываются координаты. Создается условие (!(ss>>x>>y) || ss.fail() || !ss.eof()), которое проверяет корректность ввода и чтобы ввод завершился без ошибок. Если происходит ошибка, то вызывается inputException(). Иначе в поля point2d записываются соответствующие координаты.

Создан метод inputShip, который также как и предыдущий метод в строку записывает ввод пользователя, а затем из строки происходит запись в переменные int orientation = 0, x = 0, y = 0, length = 0. Елси происходи какая-либо из ошибок при вводе, то вызывается inputException(). Если пользователь вводит в orientation что-либо помимо 0 и 1, вызывается inputException(). Данный метод принимает умный указатель на корабль и записывает в его поля нужные значения.

Создан метод inputAction, который принимает char & c и записывает в него символ из ввода.

Клас inputException представляет собой объект ошибки, который наследуется от std::exception. Внутри хранится строка сообщения об ошибке,

она инициализируется в конструкторе класса. Есть метод what, который возвращает строку об ошибке.

Создан класс outputManager, который хранит в себе ссылку на std::ostream & os.

Конструктор класса принимает ссылку на ostream & os и записывает в поле.

Создан метод drawMessage, он принимает строку std::string, и выводит в поток os.

Создан метод drawField, который принимает ссылку на поле, а также bool fog. Внутри создается двойной цикл по области игрового поля и выводится на экран каждое состояние символ для каждого состояния, если fog включен. Иначе если в сегменте nullptr, выводится „~“, если в сегменте nullptr, иначе индекс enum состояния сегмента, если fog включен.

Создан метод update(), который вызывает команду системы clear, для очистки терминала.

Создан класс scannerAbility, который наследуется от класса IAbility(public IAbility). Внутри создано поле Player * player, которое хранит указатель на игрока. Создан конструктор scannerAbility(), который инициализирует указатель nullptr.

Переопределён метод setPlayer(). Внутри записывается указатель на игрока.

Переопределён метод apply(). Внутри из Player достаются ссылки playField & play_field, inputManager & input, outputManager & output из player, т.к player является публичным. Внутри создается point2d coordinates. Создается бесконечный цикл. Вызывается метод drawMessage у outputManager. Где просится ввод координат. С помощью inputManger.inputCoordinates(), записываются координаты. После ввода координат, они проверяются на коорректность ввода (не выходят за область видимого поля). Далее если все

успешно запускается двойной цикл, который считает количество найденных сегментов в определённой области и выводит количество найденных сегментов в игровом поле с помощью outputManager.

Создан класс doubleDamageAbility, который наследуется от класса IAbility(public IAbility). Внутри создано поле Player * player, которое хранит указатель на игрока. Создан конструктор doubleDamageAbility(), который инициализирует указатель nullptr.

Переопределён метод setPlayer(). Внутри записывается указатель на игрока.

Переопределён метод apply(). Внутри значения у поля double_damage объекта player меняется на true. Далее достается outputManager из player и пишется сообщение, с помощью drawMessage, что способность активированна, можно нанести двойной урон.

Создан класс shellingAbility, который наследуется от класса IAbility(public IAbility). Внутри создано поле Player * player, которое хранит указатель на игрока. Создан конструктор shellingAbility(), который инициализирует указатель nullptr.

Переопределён метод setPlayer(). Внутри записывается указатель на игрока.

Переопределён метод apply(). Внутри из игрока достается указатель на игровое поле opponent_play_field, ссылка на менеджер вывода output, указатель на менеджер кораблей opponent_ship_manager. В начеле проверяется, с помощью метода allShipsDestroyed() у opponent_ship_manager, что все корабли уничтоженны. Если условие выполнястя, то выводится на экран, что все корабли уже уничтожены и метод возвращается. Далее выодится в терминал с помощью метода drawMessage у output сообщение, что активированна способность бомбардировки. Далее создается генератор рандомных чисел std::mt19937 gen, который принимает сид из std::random_device{}(). Далее

создаются целочисленные переменные x, y которые равны gen()%(ширина поля) и gen()%(высота поля) соответственно. Тем самым мы получаем рандомные координаты поля. Затем создается цикл, который работает до тех пор пока полученные координаты не указывают на сегмент корабля и данный сегмент является уничтоженным. Если оба условия не выполняются, то цикл заканчивается. Далее у opponent_play_field вызывается метод Attack, где not_sneak = false. Тем самым мы скрытно атакуем найденный сегмент. После этого в терминал выводится сообщение, что один из кораблей был атакован.

Создан класс abilitiesManager, который хранит в себе очередь (std::queue) указателей(std::shared_ptr<IAbility>) на способности abilities.

В конструкторе класса создается вектор указателей на способности, затем он заполняется по очереди способностью сканнера, двойного урона и обстрела. После этого вызывается метод std::shuffle, который перемешивает вектор с помощью рандомного генератора std::mt19937, который принимает сид из std::random_device()(). Далее создается цикл по вектору, в котором каждый указатель добавляется в очередь abilities, с помощью push.

Создан метод createRandomAbility, который создает случайную способность. Внутри вызывается генератор случайных чисел std::mt19937 gen, который принимает std::random_device{}()(генерирует сид). Далее создается int ability_index который равен gen()%(количество способностей). Далее создается switch-case, который принимает данный индекс и в соответствии с индексом создает и добавляет в очередь указатель на способность.

Создан метод applyAbility, который принимает указатель на Player * player. Внутри проверяется, если abilities.size() <= 0, то вызывается ошибка noAbilitiesException(). Далее из очереди достается верхний указатель, с помощью метода front() и записывается в указатель std::shared_ptr<IAbility> last_ability. Далее у данной способности вызывается метод setPlayer, которая принимает указатель на игрока player. Далее вызывается метод apply() и удаляется из очереди способность abilities.pop().

Класс noAbilitiesException представляет собой объект ошибки, который вызывается в случае, когда игрок пытается использовать способность, когда очередь является пустой. Внутри хранится строка сообщения об ошибке. В методе what, возвращается указатель на const char. Класс наследуется от std::exception.

Создан класс игрока Player, который содержит в себе следующие поля: abilitiesManager abilities_manager, playField * opponent_play_field, shipManager * opponent_ship_manager, inputManager & input_manager, outputManager & output_manager, playField, play_field, shipManager ship_manager и bool double_damage.

Конструктор класса принимает ссылки на input_manager, output_manager и записывает их в соответствующие поля.

Создан метод getOpponent, данный метод принимает указатель на игрока Player * player и записывает play_field и ship_manager в соответствующие поля указателей.

Создан метод placeShip. Внутри создана try-catch конструкция, который ловит соответствующие возможные ошибки и выводит их на экран с помощью outputManager.drawMessage(e.what()). Внутри try, с помощью output_manager, в терминал выводится сообщение о том что нужно ввести корабль и формат ввода корабля. Далее создается указатель на корабль std::shared_ptr<Ship> ship в который инициализируется std::make_shared<Ship>(). Далее с помощью inputManager записывается корабль(метод inputShip). Далее корабль добавляется в play_field.placeShip(ship, ship_manager). После этого вызывается метод update у output_manager.

Создан метод useAbility(). Внутри try-catch конструкция. В try вызывается applyAbility(this). То есть посылается указатель на игрока. В catch вызывается output_manager.drawMessage(e.what()). То есть выводится найденная ошибка.

Создан метод Attack. Внутри бесконечный цикл, который завершается, когда пользователь смог правильно ввести атаку. Создана try-catch,

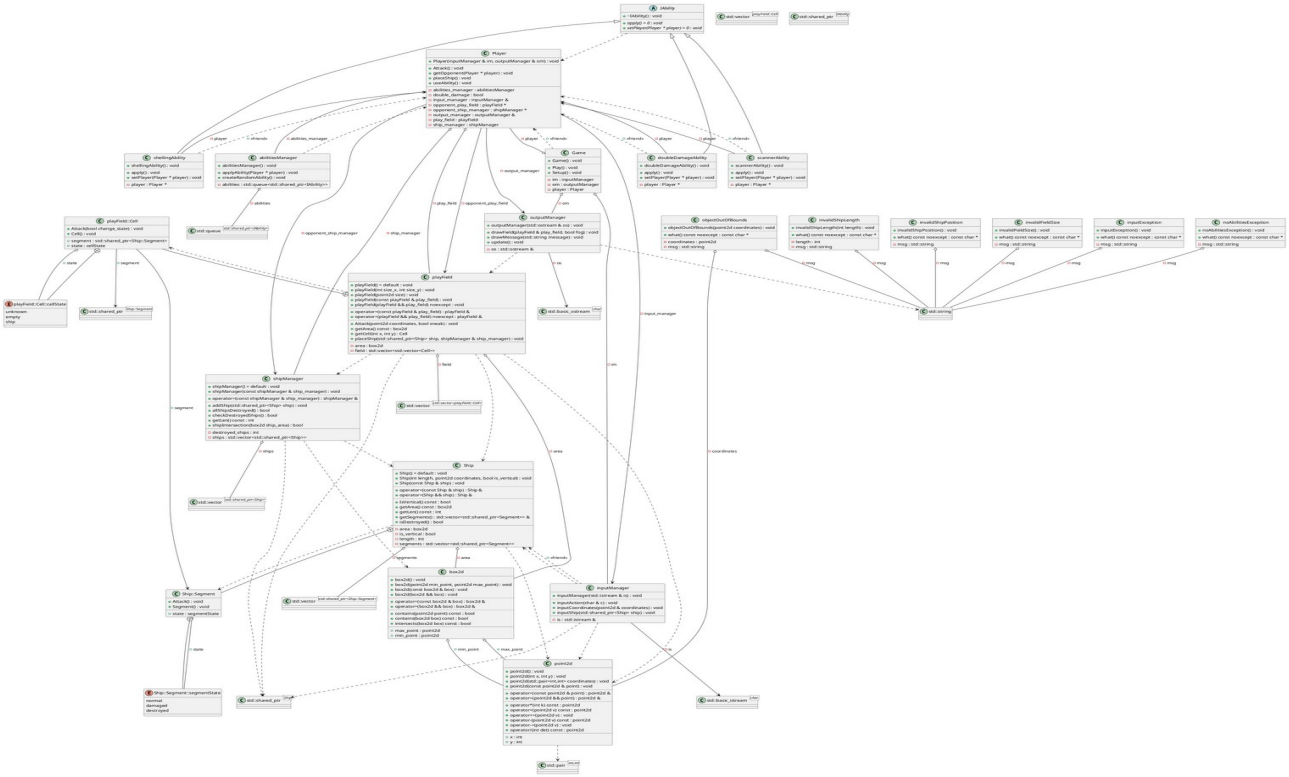
конструкция. Внутри try создается координата point2d coordinates, которая с помощью input_manager.inputCoordinates(coordinates) заполняет координаты. Далее если double_damage true, то атакуетс по заданным координатам поле противника, затем double_damage меняется на false. После этого атакуется вражеское поле по координатам и вызывается break. Если была поймана ошибка, то с помощью outputManager выводится сообщение ошибки. После этого проверяется, если какой-либо корабль уничтожился, с помощью checkDestroyedShips в opponent_ship_manager. Если таковое случилось, то выводится сообщение, что был уничтожен корабль, поэтому игрок получает способность. Далее вызывается метод createRandomAbility() у abilities_manager.

Создан класс Game, который задает логику игры, каким образом она работает. Пока, что данный логика игры такая, что игрок создает игровое поле, заставляет его кораблями.

Затем атакует свое-же поле, уничтожая свои корабли. Когда все корабли будут уничтожены, игра завершится.

Описанные выше два класса будут изменены, к следующей лабораторной работе будет улучшенна и изменена логика игры.

Диаграммы классов.



Тестирование.

Тестирование ввода размера поля:

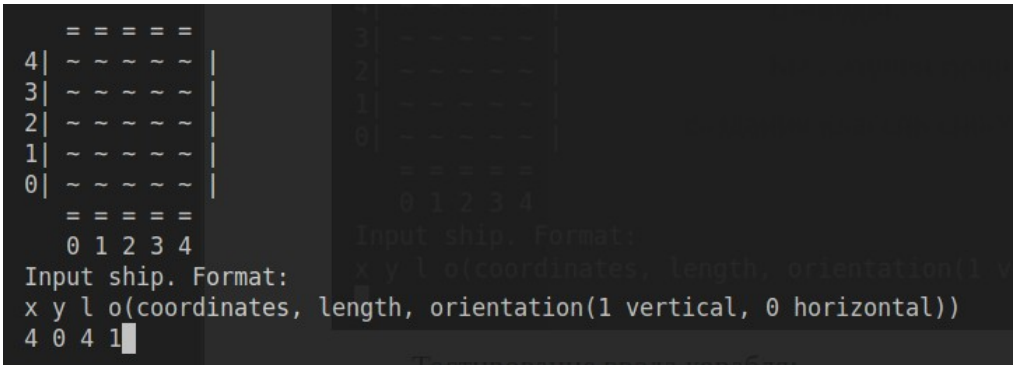
```
Input Field. Format:
w h(width, height)
5 5
```

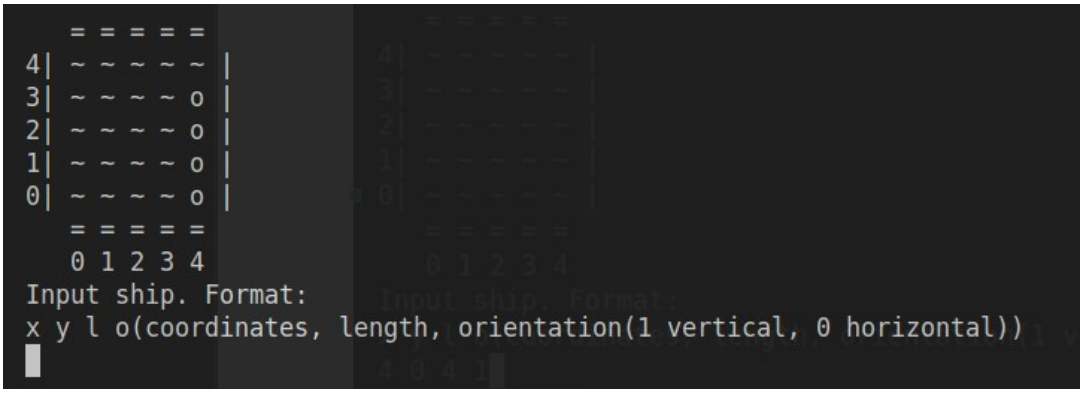
```
   = = = = =
4| ~ ~ ~ ~ ~ |
3| ~ ~ ~ ~ ~ |
2| ~ ~ ~ ~ ~ |
1| ~ ~ ~ ~ ~ |
0| ~ ~ ~ ~ ~ |
   = = = = =
   0 1 2 3 4
Input ship. Format:
x y l o(coordinates, length, orientation(1 vertical, 0 horizontal))
```

Тестирование ввода корабля:

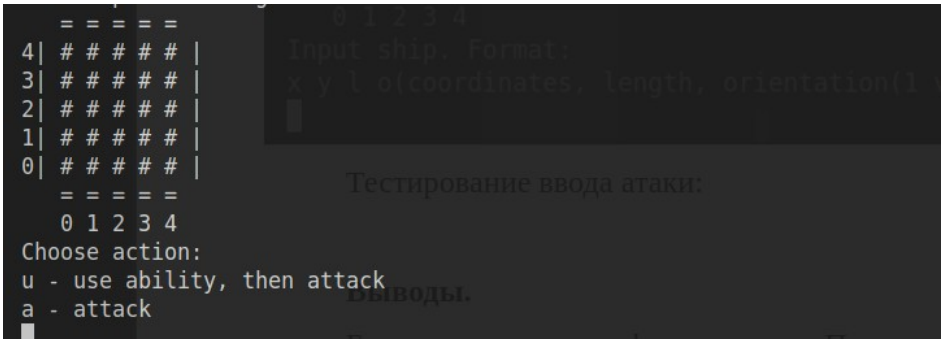
Выводы.

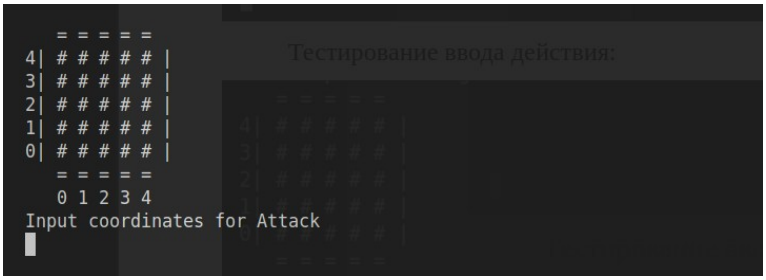
Был изучен полиморфизм классов. Получены
создания классов способностей и класса менеджер

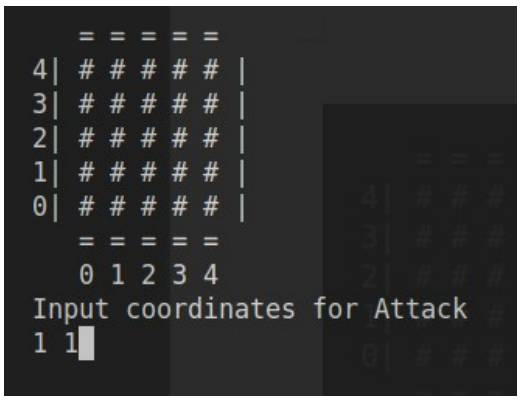




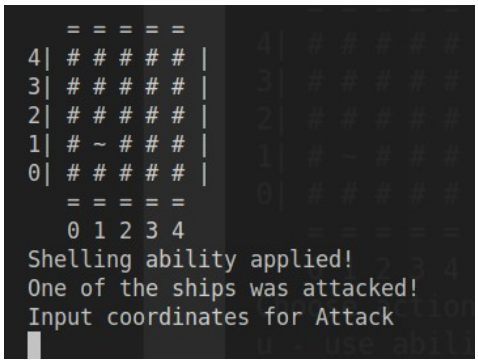
Тестирование ввода действия:

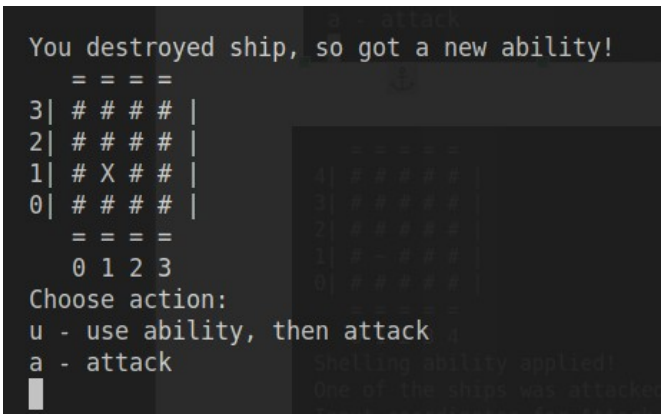


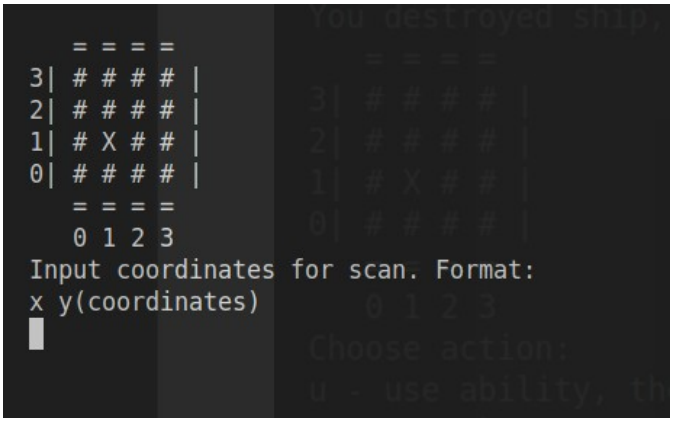


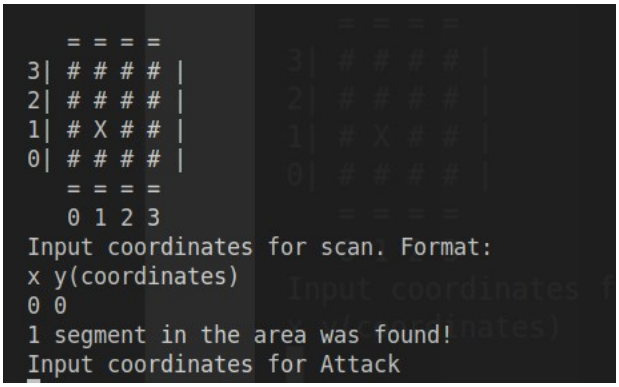

= = = = =
4| # # # # # |
3| # # # # # |
2| # # # # # |
1| # # # # # |
0| # # # # # |
= = = = =
0 1 2 3 4
Input coordinates for Attack
1 1

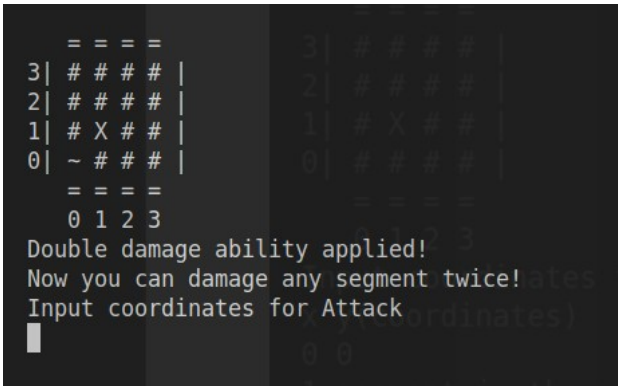

= = = = =
4| # # # # # |
3| # # # # # |
2| # # # # # |
1| # ~ # # # |
0| # # # # # |
= = = = =
0 1 2 3 4
Choose action:
u - use ability, then attack
a - attack

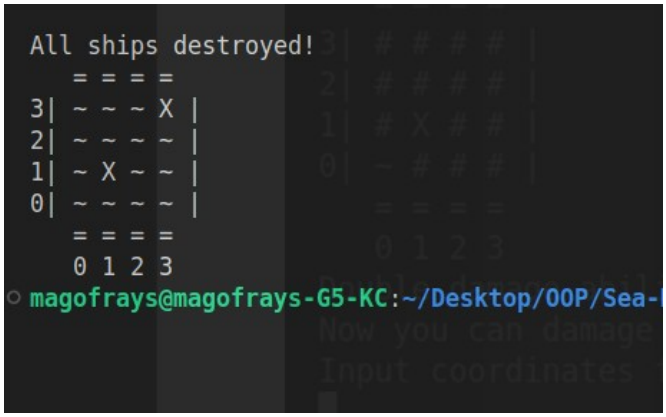

= = = = =
4| # # # # # |
3| # # # # # |
2| # # # # # |
1| # ~ # # # |
0| # # # # # |
= = = = =
0 1 2 3 4
Shelling ability applied!
One of the ships was attacked!
Input coordinates for Attack


You destroyed ship, so got a new ability!
= = = =
3| # # # # |
2| # # # # |
1| # X # # |
0| # # # # |
= = = =
0 1 2 3
Choose action:
u - use ability, then attack
a - attack


= = = =
3| # # # # |
2| # # # # |
1| # X # # |
0| # # # # |
= = = =
0 1 2 3
Input coordinates for scan. Format:
x y(coordinates)


= = = =
3| # # # # |
2| # # # # |
1| # X # # |
0| # # # # |
= = = =
0 1 2 3
Input coordinates for scan. Format:
x y(coordinates)
0 0
1 segment in the area was found!
Input coordinates for Attack


= = = =
3| # # # # |
2| # # # # |
1| # X # # |
0| ~ # # # |
= = = =
0 1 2 3
Double damage ability applied!
Now you can damage any segment twice!
Input coordinates for Attack



Выводы.

Был изучен полиморфизм классов. Получены практические навыки путем создания классов способностей и класса менеджера способностей.