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: Интерфейсы, полиморфизм

Студент гр.0382 _____ Литягин С.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

Изучить применение интерфейсов, полиморфизм.

Задание.

Могут быть три типа элементов располагающихся на клетках:

1. Игрок - объект, которым непосредственно происходит управление. На поле может быть только один игрок. Игрок может взаимодействовать с врагом (сражение) и вещами (подобрать).
2. Враг - объект, который самостоятельно перемещается по полю. На поле врагов может быть больше одного. Враг может взаимодействовать с игроком (сражение).
3. Вещь - объект, который просто располагается на поле и не перемещается. Вещей на поле может быть больше одной.

Требования:

- Реализовать класс игрока. Игрок должен обладать собственными характеристиками, которые могут изменяться в ходе игры. У игрока должна быть прописана логика сражения и подбора вещей. Должно быть реализовано взаимодействие с клеткой выхода.
- Реализовать три разных типа врагов. Враги должны обладать собственными характеристиками (например, количество жизней, значение атаки и защиты, и т. д.; желательно, чтобы у врагов были разные наборы характеристик). Реализовать логику перемещения для каждого типа врага. В случае смерти врага он должен исчезнуть с поля. Все враги должны быть объединены своим собственным интерфейсом.
- Реализовать три разных типа вещей. Каждая вещь должна обладать собственным взаимодействием на ход игры при подборе *(например, лечение игрока).* При подборе, вещь должна исчезнуть с поля. Все вещи должны быть объединены своим собственным интерфейсом.
- Должен соблюдаться принцип полиморфизма

Потенциальные паттерны проектирования, которые можно использовать:

- *Шаблонный метод (Template Method) - определение шаблона поведения врагов*
- *Стратегия (Strategy) - динамическое изменение поведения врагов*
- *Легковес (Flyweight) - вынесение общих характеристик врагов и/или для оптимизации*
- *Абстрактная Фабрика/Фабричный Метод (Abstract Factory/Factory Method) - создание врагов/вещей разного типа в runtime*
- *Прототип (Prototype) - создание врагов/вещей на основе "заготовок"*

Выполнение работы.

Для реализации графического интерфейса используется SFML библиотека.

Создан класс Object – интерфейс всех объектов (т.е. для врагов, вещей и игрока), что могут располагаться на поле. Помимо геттеров и сеттеров (часть которых относится к вещам, а другая - к герою и врагам), содержит виртуальный метод Interaction(Object* unit), который реализуется в классах-наследниках для взаимодействия объектов (более подробно о методе смотрите в описании классов-наследников интерфейса Unit). Общий интерфейс объектов нужен, чтобы мы могли себе позволить не хранить в клетке указатель на каждый из возможных типов объектов (т.е. указатель на вещь, указатель на врага, указатель на игрока), а хранить лишь один указатель на объект. По этой же причине многие методы определены именно в данном интерфейсе, поскольку в будущем к объектам мы будем обращаться в основном через метод GetObject клетки. Соответственно, стоит отметить, что в класс клетки Cell мы добавили поле Object* object, а также метод для получения указателя на объект, метод для получения типа объекта и метод для установки объекта на клетку; методы нужны для работы с объектами на клетках в программе после того, как они будут расставлены по игровому полю.

От интерфейса Object наследуется интерфейс вещей Thing. Он нужен, чтобы мы могли, как минимум, хранить все вещи в массиве типа Thing, а не создавать для каждого типа свой массив. В данном интерфейсе определяется лишь один виртуальный метод IsAvailable(). Он нужен, чтобы мы могли знать, доступна ли вещь или же ее уже подобрали. Соответственно, его реализация в классах-наследниках будет возвращать нам true, если вещь все еще доступна; false – в противном случае.

От интерфейса Thing наследуются классы вещей всех типов. Это классы Axe, Candy и Coin. Имеют одинаковые поля ObjectType type, bool is_available и int coord[2]. Соответственно, поле type нужно для того, чтобы мы знали какой тип имеет данный объект; поле is_available нужно для хранения информации, доступна ли вещь (именно это поле возвращает переопределенный метод IsAvailable(), который был упомянут ранее в интерфейсе вещей); поле coord нужно для хранения координаты (coord[0] – координата по x, coord[1] – координата по y), на которой находится данная вещь. Также каждый из классов имеет поле со значением (далее параметр), которое понадобиться при взаимодействии игрока с вещью. Для Axe это поле int damage (когда игрок подбирает топор, то именно на это значение увеличивается его сила удара); для Candy это поле int hp (когда игрок подбирает конфетку, то именно на это значение увеличивается его здоровье); для Coin это поле int value (когда игрок подбирает монетку, то именно на это значение увеличивается его общее количество монет). Чтобы получить этот параметр, в интерфейса Object был описан виртуальный метод GetData(), который переопределяется в данных классах и возвращает значение параметра; также, если он был вызван, то полю is_available устанавливается значение false, поскольку данный метод вызывается лишь при взаимодействии с данной вещью. Метод описан именно в интерфейсе Object потому, что при взаимодействии мы работаем с полем Object* object клетки, а значит, опиши мы его в интерфейсе Thing, обратиться к нему не получилось бы.

Также от интерфейса Object наследуется интерфейс Unit. Это общий интерфейс для движущихся объектов, т.е. для игрока и врагов. Он также нужен, чтобы мы могли, как минимум, хранить все движимые объекты в массиве типа Unit, а не создавать для каждого типа свой массив. Помимо сеттеров, имеет две важные виртуальные функции: IsAlive() и Move(Cell** cells, int x, int y). Первая нужна, чтобы мы могли узнать, жив ли юнит. Соответственно, его реализация в классах-наследниках будет возвращать нам true, если юнит все еще жив; false – в противном случае. Вторая же нужна для движения юнитов (метод реализован в классах-наследниках).

От интерфейса Unit наследуется класс игрока Hero. Он имеет поля int health (здоровье), int coins (количество монет), int force (наносимый урон), bool is_alive (жив или убит), int maxHealth (максимальное здоровье), ObjectType type (тип объекта), int coord[2] (координаты; coord[0] – координата по x, coord[1] – координата по y). Помимо геттеров и сеттеров, имеет несколько важных переопределенных методов. Во-первых, метод IsAlive(), который был ранее упомянут в интерфейсе Unit; он возвращает поле is_alive. Метод Move(Cell** cells, int x, int y), аргументы которого есть массив клеток игрового поля и координата (x, y), на которую хотим передвинуться, отвечает за передвижение игрока на указанную координату (x, y) в аргументах метода; если координата (x, y) не выходит за границу поля, а также на клетку cells[x][y] можно сдвинуться (проверяется методом IsMovable() клетки) или клетка cells[x][y] имеет тип exit (тип клетки получаем методом GetType()). Дальше проверяется: если на данной клетке нет объекта (т.е. метод клетки GetObjectType() вернет empty), то герой передвигается на клетку cells[x][y] (клетке по координате игрока методом SetObject(Object* object) устанавливается нулевой указатель; игроку методом SetCoord(int x, int y) устанавливается новая координата (x, y); клетке cell[x][y] методом SetObject(Object* object) устанавливается указатель на игрока); если же на данной клетке будет какой-то объект, то вызывается метод игрока Interaction(Object* object), сам игрок при этом не сдвигается. Метод

Interaction(Object* object) , аргумент которого есть указатель на объект, с которым будет взаимодействие: если игрок взаимодействует с объектом типа eye, ent или spider (тип получаем вызовом методе GetType() у object), то данному объекту устанавливаем здоровье методом SetHealth(int health), равное разности здоровья объекта (получаем методом getHealth()) и урону, который наносит игрок (т.е. значения поля force игрока), в методе SetHealth() также присутствует проверка, что если новое здоровье будет меньше нуля, то юниту устанавливается в поле is_alive значение falce; если же объект имеет тип axe, то игроку устанавливается новый урон, который он может наносить, методом SetForce(int damage) (новый урон есть сумма старого и значения, получаемого методом GetData() объекта); если же объект имеет тип coin, то игроку устанавливается новое количество монет методом SetCoins(int value) (новое количество монет есть сумма старого количества и значения, получаемого методом GetData() объекта; также в методе SetCoins() выполняется проверка: если монет больше 10, то игроку методов SetMaxHealth(int maxHealth) устанавливает максимальное здоровье, увеличенное на 5 единиц от старого); если же объект имеет тип candy, то игроку устанавливается новое здоровье методом SetHealth(int health) (новое здоровье есть сумма старого и значения, получаемого методом GetData() объекта; также в методе SetHealth() есть проверка на случай, если новое здоровье превысит максимальное значение; в этом случае в поле health будет установлено значение поля maxHealth).

От интерфейса Unit также наследуются все враги, это классы Eye, Ent и Spider. Все они имеют одинаковые поля, а именно: int health (здоровье), int force (наносимый урон), bool is_alive (жив или убит, ObjectType type (тип объекта), int coord[2] (координаты; coord[0] – координата по x, coord[1] – координата по y). Также имеет геттеры и сеттеры, но тут приведу реализацию нескольких переопределенных методов. Во-первых, метод IsAlive(), реализован также, как и в классе Hero. Метод Move(Cell** cells, int x, int y), аргументы которого есть массив клеток игрового поля и координата (x, y) , на которую хотим

передвинуться, отвечает за передвижение врага на указанную координату (x, y) в аргументах метода; если координата (x, y) не выходит за границу поля, а также на клетку cells[x][y] можно сдвинуться (проверяется методом IsMovable() клетки; в реализации движения Spider не проверяется; это его особенность передвижения), то дальше проверяется: если на данной клетке нет объекта (т.е. метод клетки GetObjectType() вернет empty), то враг передвигается на клетку cells[x][y] (клетке по координате врага методом SetObject(Object* object) устанавливается нулевой указатель; врагу методом SetCoord(int x, int y) устанавливается новая координата (x, y); клетке cell[x][y] методом SetObject(Object* object) устанавливается указатель на врага); если же на данной клетке будет игрок (т.е. объект с типом hero), то вызывается метод игрока Interaction(Object* object), сам враг при этом не сдвигается. Метод Interaction(Object* object), аргумент которого есть указатель на объект, с которым будет взаимодействие, у враг довольно прост: объекту просто устанавливается методом SetHealth(int health) здоровье, равное разности значения старого здоровья объекта(полученное методом GetHealth()) и урона, который наносится данным врагом (т.е. значение в поле force).

Для создания врагов был использован паттерн Абстрактная фабрика. Для его реализации нам нужен интерфейс фабрики и классы конкретных фабрик. Поэтому был создан интерфейс ObjectFactory, имеющий виртуальный метод CreateUnit() для создания конкретных объектов (реализация в классах-наследниках). Собственно, классы-наследники: SpiderFactory, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Spider; EntFactory, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Ent; EyeFactory, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Eye. Данные фабрики используются в классе игры Game при генерации врагов.

Для генерации врагов и вещей на карте в классе игры Game были написаны два метода: CreateThing(Cell** cell, Thing** thing) и CreateEvil(Cell**

cell, Unit** evil), оба метода вызываются в методе класса Game StartGame(). Метод CreateEvil(Cell** cell, Unit** evil), в аргументы которому передают массив клеток игрового поля и массив, куда будут записаны враги, генерирует врагов на поле. В методе задаются 3 новых объекта фабрик: EyeFactory, SpiderFactory и EntFactory, локальная переменна int direction, по которой будет выбираться тип создаваемого врага и локальные переменные int x и int y, что будут отвечать за координату на поле cell (имеют изначально значения 0 и 0 соответственно). Как происходит генерация: direction методом rand() получает значение от 0 до 9; если оно меньше 3, то evil[i] получает указатель на новый объект Eye (его вернет метод CreateUnit() фабрики EyeFactory); если оно меньше 6, то evil[i] получает указатель на новый объект Ent (его вернет метод CreateUnit() фабрики EntFactory); если оно меньше 9, то evil[i] получает указатель на новый объект spider (его вернет метод CreateUnit() фабрики SpiderFactory). Затем, пока cell[x][y] является клеткой, по которой нельзя двигаться или метод GetObjectType данной клетки возвращает тип не empty, то координаты x и y меняют значение, определяемое в диапазоне от 1 до Size – 2, где Size – размер поля. Когда такая координата найдена evil[i] устанавливается на нее, т.е. вызывается метод SetCoord(x, y) объекту, а также cell[x][y] устанавливается этот объект методом SetObject(). Метод CreateThing(Cell** cell, Thing** thing) реализован схожим образом.

Все новые классы, все измененные классы и их методы представлены ниже:

1. В классе Cell было добавлено поле Object* object, а также добавлены следующие методы:
   - метод GetObjectType() возвращает тип объекта, находящего на ней
   - метод GetObject() возвращает указатель на объект, который хранится в поле object клетки
   - метод SetObject(Object* object) устанавливает в поле object клетки указатель на объект

2. Класс интерфейса элемента клетки Object был изменен. Объявлены следующие методы:

- виртуальный метод SetCoord(int x, int y) для присваивания координат объекту (метод реализуется в классах-наследниках)
- виртуальный метод GetCoord() возвращает указатель на массив с координатами объекта (метод реализуется в классах-наследниках)
- виртуальный метод SetType(ObjectType value) устанавливает тип объекту (метод реализуется в классах-наследниках)
- виртуальный метод SetHealth(int health) устанавливает здоровье объекту (метод реализуется в классах-наследниках)
- виртуальный метод GetHealth() возвращает значение здоровья объекта (метод реализуется в классах-наследниках)
- виртуальный метод Interaction(Object* object), аргумент которого есть объект, с которым происходит взаимодействие, проводит взаимодействие с объектом (метод реализуется в классах-наследниках)
- виртуальный метод GetType() возвращает тип объекта (метод реализуется в классах-наследниках)
- виртуальный метод GetForce() возвращает значение урона, наносимого объектом (метод реализуется в классах-наследниках)
- виртуальный метод GetData() возвращает значение поля данных объекта (метод реализуется в классах-наследниках)

3. Класс интерфейса юнитов Unit наследуется от интерфейса Object. Объявлены следующие методы:

- виртуальный метод SetForce(int damage) для установки урона, наносимого юнитом (метод реализуется в классах-наследниках)
- виртуальный метод GetForce() для получения урона, наносимого юнитом (метод реализуется в классах-наследниках)

- виртуальный метод SetMaxHealth(int maxHealth) для установки максимального здоровья (метод реализуется в классах-наследниках)
- виртуальный метод IsAlive() для проверки, активен ли объект
- виртуальный метод Move(Cell** cell, int x, int y) для перемещения юнита (метод реализуется в классах-наследниках)

4. Классы юнитов Ent, Eye, Spider наследуются от класса интерфейса Unit. Описание классов в файлах Ent.cpp, Eye.cpp, Spider.cpp, определение в Ent.h, Eye.h, Spider.h. Имеют поля int health, int force, bool is_alive, ObjectType type, int coord[2]. Содержат следующие методы:

- переопределенный метод SetCoord(int x, int y); устанавливает переданные координаты в coord[0] и coord[1] соответственно (т.е. устанавливает объекту координаты, на которых он находится)
- переопределенный метод GetCoord() возвращает указатель на целочисленный массив coord (поле объекта класса), содержащий координаты объекта
- переопределенный метод SetType(ObjectType value) устанавливает в поле type переданное значение value (т.е. устанавливает тип)
- переопределенный метод SetHealth(int health) устанавливает полю health переданное значение health (т.е. устанавливает количество здоровья); также проверяет, если установленное значение <= 0, то устанавливает полю is_alive значение false
- переопределенный метод GetHealth() возвращает значение из поля health (т.е. возвращает количество здоровья)
- переопределенный метод Interaction(Object* object), аргумент которого есть объект, с которым происходит взаимодействие, проводит взаимодействие с объектом; вызывается метод SetHealth() объекта unit для установки здоровья, равного разности его здоровья до взаимодействия и урона, наносимого объектом, метод взаимодействия которого был вызван к данному объекту; метод

вызывается при определенном условии, которое проверяется при вызове метода Move () (т.е. метода передвижения объекта)

- переопределенный метод GetType() возвращает значение поля type (т.е. возвращает тип объекта)
- переопределенный метод SetForce(int damage) устанавливает в поле force значение damage (т.е. устанавливает урон, наносимый юнитом)
- переопределенный метод IsAlive() возвращает значение поля is_alive (т.е. возвращает true, если юнит все еще жив; false, если юнит уже не жив)
- переопределенный метод Move(Cell** cell, int x, int y), аргументы которого есть массив клеток игрового поля и координата (x,y), куда должен переместиться объект, реализует перемещение юнита; выполняется проверка: если значения x и y >0 и <Size-1, а также клетка cell[x][y] доступна для передвижения (метод клетки IsMovable() возвращает true, если тип клетки подходит для передвижения) (данное условие не проверяется при передвижении юнитов с типом spider), то если на клетке cell[x][y] нет никакого объекта (т.е. метод клетки GetObjectType() вернет empty), то для клетки с координатой, на которой находится юнит, в поле object методом SetObject() устанавливается нулевой указатель, затем объекту устанавливается новая координата (x,y) методом SetCoord(), а клетке по этой же координате в поле object методом SetObject() устанавливается указатель на данный объект; если же на клетке по координате (x,y) есть объект с типом hero (т.е. игрок), то вызывается метод юнита Interaction() к объекту, находящемуся на клетке с данной координатой (таким образом, юниты взаимодействуют только с игроком)

5. Юниты создаются при использовании паттерна Абстрактная фабрика. Для этого был реализован класс интерфейса фабрик ObjectFabric.

Описание класса в файле ObjectFabric.cpp, определение в ObjectFabric.h. Объявлен следующий метод:

- виртуальный метод CreateUnit() возвращает указатель на созданный объект (метод реализуется в классах-наследниках)

6. Классы фабрик EntFabric, EyeFabric, SpiderFabric (производят юнитов ent, eye, spider соответственно) наследуются от класса интерфейса ObjectFabric. Описание классов в файлах EntFabric.cpp, EyeFabric.cpp, SpiderFabric.cpp, определение в EntFabric.h, EyeFabric.h, SpiderFabric.h. Содержат следующий метод:

- переопределенный метод CreateUnit() возвращает указатель на созданный объект

7. Класс Game. Добавлено два метода:

- метод CreateEvil(Field* field, int EVIL) для генерации противников на поле; генерируется EVIL штук противников, с помощью функции rand() выбирает число до 9: если оно менее 3, то создается противник типа eye; если оно менее 6, то создается противник типа ent; если оно менее 9, то создается противник типа spider; созданный противник устанавливается на координату (x,y), где x и y меньше Size-2 и больше 1, а также при условии, что на клетке с такой координатой нет объекта (если объект на клетке уже есть, то выбираются другие x и y с помощью функции rand(), пока условие не выполнится)
- метод CreateThing(Field* field, int THING) для генерации вещей на поле; генерируется THING штук вещей, с помощью функции rand() выбирает число до 9: если оно менее 3, то создается вещь типа candy; если оно менее 6, то создается вещь типа axe; если оно менее 9, то создается вещь типа coin; созданная вещь устанавливается на координату (x,y), где x и y меньше Size-2 и больше 1, а также при условии, что на клетке с такой координатой нет объекта (если

объект на клетке уже есть, то выбираются другие x и y с помощью функции rand(), пока условие не выполнится)

8. Класс интерфейса вещей Thing. Описание класса в файле Thing.cpp, определение в Thing.h. Объявлен следующий метод:

- виртуальный метод IsAvailable() для проверки, доступен ли объект (метод реализуется в классах-наследниках)

9. Классы вещей, наследуемые от интерфейса Thing, Candy, Coin, Axe. Класс Candy имеет поле int hp, класс Axe имеет поле int damage, класс Coin имеет поле int value; классы Candy, Coin, Axe также содержат поля int coord[2], ObjectType type, bool is_available. Содержат следующие методы:

- переопределенный метод SetCoord(int x, int y); устанавливает переданные координаты в coord[0] и coord[1] соответственно (т.е. устанавливает объекту координаты, на которых он находится)
- переопределенный метод GetCoord() возвращает указатель на целочисленный массив coord (поле объекта класса), содержащий координаты объекта
- переопределенный метод GetData() возвращает значение поля данных (hp, damage, value соответственно для классов Candy, Coin, Axe) и устанавливает в поле is_ available значение false (поскольку метод вызывается, когда произошло взаимодействие с этим предметом, то мы и делаем его недоступным в дальнейшем)
- переопределенный метод IsAvailable() возвращает значение поля is_ available (т.е. возвращает true, если предмет все еще доступен; false, если предмет уже не доступен)
- переопределенный метод GetType() возвращает значение поля type (т.е. возвращает тип объекта)

10. Класс игрока Hero. Наследуется от класса интерфейса Unit. Имеет поля int health, int force, int coins, bool is_alive, int MaxHealth, ObjectType type, int coord[2]. Содержат следующие методы:

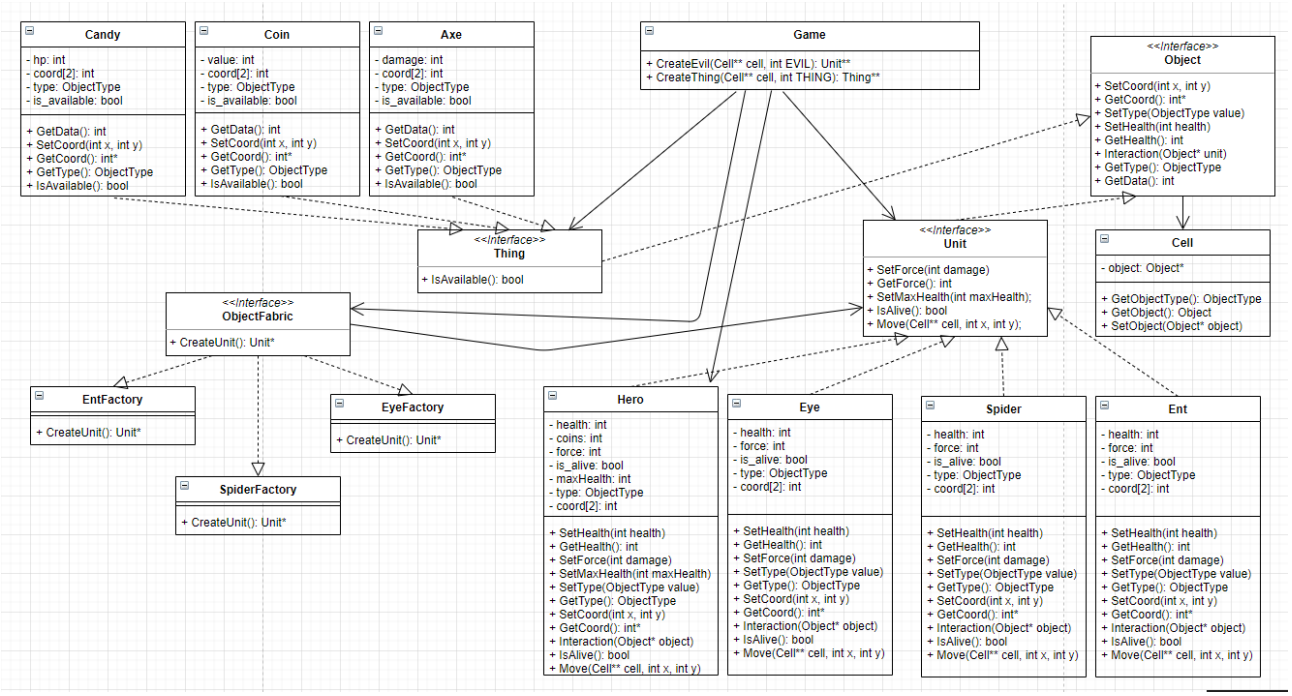
- переопределенный метод SetMaxHealth(int maxHealth) для установки максимального здоровья
- переопределенный метод SetCoord(int x, int y); устанавливает переданные координаты в coord[0] и coord[1] соответственно (т.е. устанавливает объекту координаты, на которых он находится)
- переопределенный метод GetCoord() возвращает указатель на целочисленный массив coord (поле объекта класса), содержащий координаты объекта
- переопределенный метод SetType(ObjectType value) устанавливает в поле type переданное значение value (т.е. устанавливает тип)
- переопределенный метод SetHealth(int health) устанавливает полю health переданное значение health (т.е. устанавливает количество здоровья); также проверяет, если установленное значение <= 0, то устанавливает полю is_alive значение false
- переопределенный метод GetHealth() возвращает значение из поля health (т.е. возвращает количество здоровья)
- переопределенный метод Interaction(Object* unit), аргумент которого есть объект, с которым происходит взаимодействие, проводит взаимодействие с объектом; если тип объекта unit соответствует либо ent, либо eye, либо spider (т.е. противники), то объекту unit устанавливается здоровье, равное разности его здоровья до взаимодействия и урона, наносимого игроком; если тип бъекта unit соответствует axe (т.е. предмет), то игроку устанавливает урон, равный сумме его урона до взаимодействия с предметом и значения поля данных damage предмета; если тип бъекта unit соответствует candy (т.е. предмет), то игроку устанавливает здоровье, равное сумме его здоровья до взаимодействия с предметом и значения поля данных hp предмета; если тип бъекта unit соответствует coin (т.е. предмет), то игроку устанавливает

количество монет, равное сумме его монет до взаимодействия с предметом и значения поля данных value предмета

- переопределенный метод GetType() возвращает значение поля type (т.е. возвращает тип объекта)
- переопределенный метод SetForce(int damage) устанавливает в поле force значение damage (т.е. устанавливает урон, наносимый игроком)
- переопределенный метод IsAlive() возвращает значение поля is_alive (т.е. возвращает true, если игрок все еще жив; false, если игрок уже не жив)
- переопределенный метод Move(Field* field, int x, int y), аргументы которого есть массив клеток игрового поля и координата (x,y), реализует перемещение юнита; выполняется проверка: если значения x и y >0 и <Size-1, а также клетка cell[x][y] доступна для передвижения (метод клетки IsMovable() возвращает true, если тип клетки подходит для передвижения) (данное условие не проверяется при передвижении юнитов с типом spider), то если на cell[x][y] нет никакого объекта (т.е. метод клетки GetObjectType() вернет empty), то для клетки с координатой, на которой находится юнит, в поле object методом SetObject() устанавливается нулевой указатель, затем объекту устанавливается новая координата (x,y) методом SetCoord(), а клетке по этой же координате в поле object методом SetObject() устанавливается указатель на данный объект; если же на клетке по координате (x,y) есть объект, тип которого отличен от empty (т.е. какой-то объект), то вызывается метод юнита Interaction() к объекту, находящемуся на клетке с данной координатой (таким образом, игрок взаимодействует с объектами)
- метод SetCoins(int value) добавляет значение value к полю количества монет игрока coins; если их 10 и более, то увеличивает максимальное здоровье на 5, а значение поля coins становится 0

- метод GetCoins() возвращает значение из поля coins

Рисунок 1 – UML-диаграмма классов.



Выводы.

В ходе лабораторной работы было изучено применение интерфейсов, полиморфизм. Также были разработаны классы юнитов и предметов, класс игрока. Взаимодействие юнита и игрока. Взаимодействие игрока и объектов. Генерация объектов на игровом поле. Был изучен паттерн Абстрактная фабрика.