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 _____ Тихонов С.В.

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.

Класс Object – интерфейс всех объектов (т.е. для врагов, вещей и игрока), что могут располагаться на поле. Помимо get... и set... (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 друг с другом. Общий интерфейс нужен, чтобы не хранить в клетке указатель на каждый из возможных типов объектов, а хранить лишь один указатель на объект. Поэтому многие методы определены в данном интерфейсе, в будущем к объектам мы будем обращаться через метод GetObject клетки. В клетку был добавлен данный метод, метод для получения указателя на объект, метод, чтобы получить его тип и метод, чтобы установить объект 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 Things. Он нужен, чтобы мы могли хранить все вещи в массиве типа Things, а не создавать для каждого типа свой массив. В данном интерфейсе лишь один виртуальный метод IsAvailable(). Он нужен, чтобы мы могли знать, доступна ли вещь или же ее уже

подобрали. 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 Shawarma, Power_Drink и Apple_tree. Имеют одинаковые поля ObjectType _type, bool is_available и int coord[2]. Поле type нужно для того, чтобы мы знали какой тип имеет данный объект; поле _is_available нужно для хранения информации, доступна ли вещь; поле _coord нужно для хранения координаты по x и y, на которой находится данная вещь. Также каждый из классов имеет поле со значением, которое используется при взаимодействии игрока с вещью. Для Power_Drink это поле int _damage; для Shawarma это поле int _hp (когда игрок подбирает конфетку, то именно на это значение увеличивается его здоровье); для Apple_tree это поле int _maxhp (когда игрок ест яблоки с дерева, то его максимальный запас здоровья увеличивается. Фрукты есть полезно!). 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, так как данный метод вызывается при взаимдействии игрока с предметом. Метод описан именно в интерфейсе Object потому, что при взаимодействии мы работаем с полем Object* object клетки, а значит, еслибы был бы в Things, обратиться к нему не получилось бы.

Также от интерфейса Object наследуется интерфейс Unit. Это общий интерфейс для движущихся объектов, а именно для игрока и врагов. Он нужен для того-же самого, что и Things. Помимо set..., 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 иначе. Вторая же нужна для движения юнитов (метод реализован в классах-наследниках).

От интерфейса Unit наследуется класс игрока Player. Он имеет поля int health (здоровье), int force (наносимый урон), bool _is_alive (жив или убит), int maxHealth (максимальное здоровье), ObjectType type (тип объекта), int _coord[2] (координаты x и y). Помимо get... и set..., имеет несколько переопределенных методов. Метод IsAlive() из интерфейса Unit; он возвращает поле _is_alive. Метод Move(Cell** cells, int x, int y), аргументы которого массив клеток игрового поля и координата (x, y), на которую хотим пойти, 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 (при помощи метода GetObjectType() который в этом случае вернёт 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 перемещается на данную клетку. Метод Interaction(Object* object) , аргумент которого вляется указатель на объект, с которым будет взаимодействие: если игрок взаимодействует с врагом, то данному врагу 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 или равно нулю, то юниту устанавливается в поле _is_alive значение falce; если же объект имеет тип POWER_DRINK, то игроку устанавливается новый урон, который он может наносить, методом SetForce(int damage) (новый урон это сумма старого и значения, получаемого методом GetData() объекта); если же объект имеет тип

APPLE_TREE, то игроку прибавляется значение к _maxHealh методом SetMaxHealth(int maxHealth); если же объект имеет тип SHAWARMA, то игроку прибавлеется значение к _health методом SetHealth(int health) объекта; также в методе 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 Children, Car и Redneck. Все они имеют такие же поля как у класса Player. Также так же большинство методов идентично методам класса Player. 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), 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, 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, 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 OCreate, имеющий виртуальный метод CreateUnit() для создания конкретных объектов (реализация в классах-

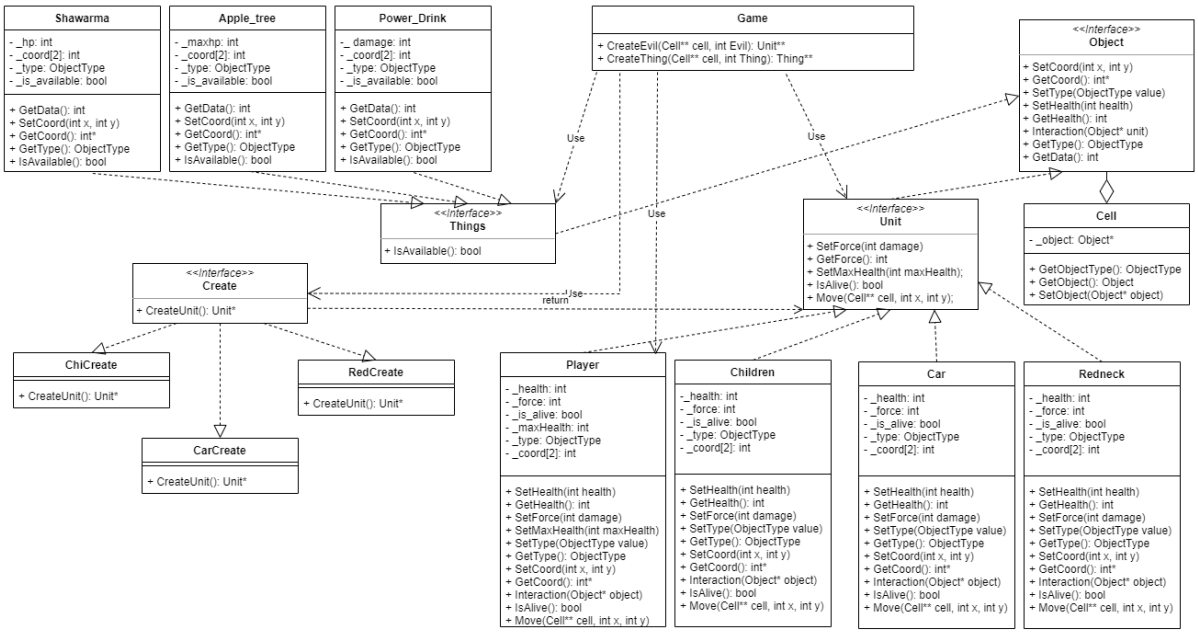
наследниках). Классы-наследники: ChiCreate, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Children; CarCreate, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Car; RedCreate, в данной фабрике переопределенный метод CreateUnit() возвращает указатель на новый объект Redneck. Данные фабрики используются в классе игры Start при генерации врагов.

Для генерации врагов и вещей на карте в классе игры Start были написаны два метода: CreateThing(Cell** cell, int Thing) и CreateEvil(Cell** cell, int Evil), оба метода вызываются в методе класса Start Start() и возвращают соответственно массив Things ** things и Unit** evil. Метод CreateEvil(Cell** cell, int evil), в аргументы которому передают массив клеток игрового поля и количество врагов, которых нужно создать. В методе задаются 3 новых объекта фабрик: ChiCreate, CarCreate и RedCreate, 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 Redneck (его вернет метод CreateUnit() фабрики RedCreate); если оно меньше 6, то evil[i] получает указатель на новый объект Children (его вернет метод CreateUnit() фабрики ChiCrate); если оно меньше 9, то evil[i] получает указатель на новый объект Car (его вернет метод CreateUnit() фабрики CarCreate). 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 HIGHT – 2 и от 1 до WIDTH -2 соответственно, 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, int Thing) реализован схожим образом.

Все новые классы, все измененные классы и их методы представлены и взаимодействие с другими классами представлены в UML диаграмме

Рисунок 1 – UML-диаграмма классов.



Вывод.

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.