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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «ООП»

Тема: Интерфейсы, полиморфизм

Студент гр. 0383 _____ Козлов Т.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Могут быть три типа элементов располагающихся на клетках:

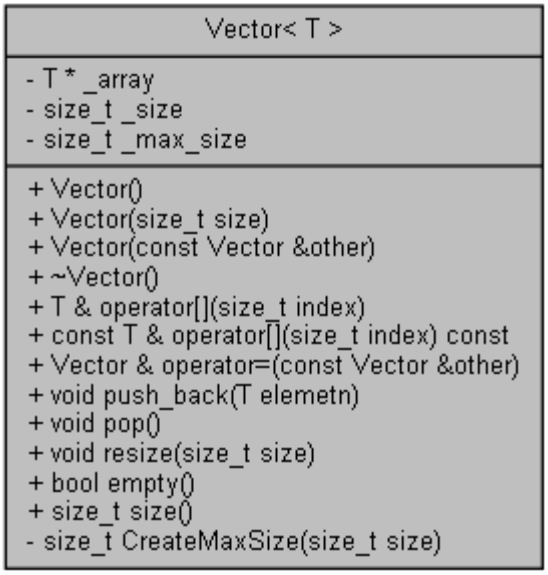
1. Игрок - объект, которым непосредственно происходит управление. На поле может быть только один игрок. Игрок может взаимодействовать с врагом (сражение) и вещами (подобрать).
2. Враг - объект, который самостоятельно перемещается по полю. На поле врагов может быть больше одного. Враг может взаимодействовать 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"*

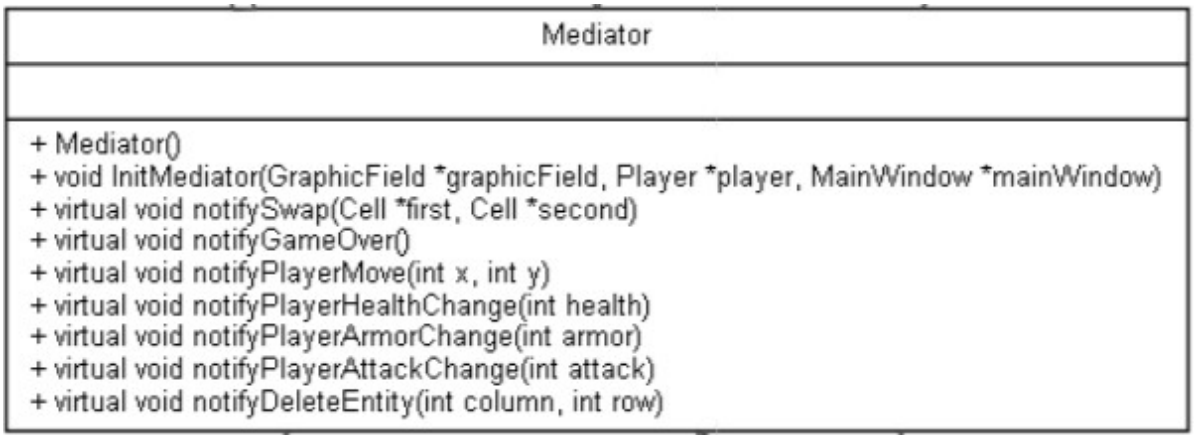


“Vector”

Ход работы:

В ходе выполнения лабораторной работы был реализован контейнер вектор для удобного взаимодействия с массивами данных (см. рис. «Vector»)

В ходе выполнения работы был создан класс Mediator, реализующий паттерн «Медиатор» (Посредник, Контроллер), который осуществляет посредничество между бизнес-логикой и GUI (вкл. считывание клавишь). На момент выполнения работы класс содержит сслыку на Player (см.далее), ссылку на MainWindow и ссылку на GraphicField (графическое поле). По функционалу – класс имеет необходимые методы для передачи информации от одного объекта к другому (например от игрока Player в методе notifyGameOver() экземпляру MainWindow передается информация о том, что игра закончена).



Mediator

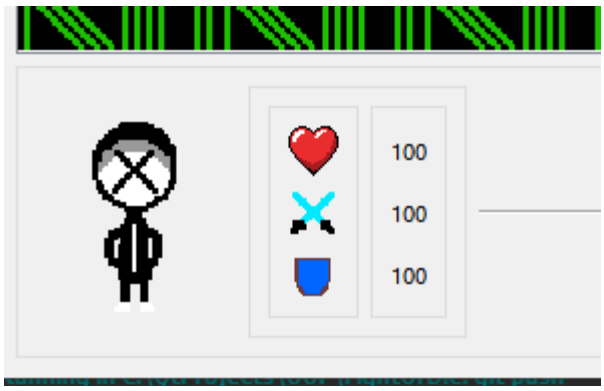
Был создан класс ICharacter, наследуемый от IEntity и имеющий поле-указатель Mediator, ICharacter будут наследовать «Персонажи» -

сущности, которые являются «живыми». Этот класс наследуют абстрактный класс Enemy и класс Player – класс главного игрока.

Класс Player имеет три поля-характеристики (_health, _attack, _armor), ссылки на клетку(_cell) и игровое поле(_gameField), на котором он находится.

По функционалу Игрок способен двигаться (за это отвечает метод Move(), в котором используются текущее игровое поле и текущая клетка. От GUI поступает информация о нажатии клавиши – значения передаются в медиатор, который вызывает метод Move() у игрока (в методе notifyPlayerMove()), где происходят необходимые вычисления и проверки (проверка на то, что новые координаты клетки, на которую надо переместиться, находятся в границах поля – с помощью метода CheckOnInclusion()), для того, чтобы понять может ли игрок переместиться.

Так же в этом же слоте реализовано взаимодействие с классом Врагов (подробности далее) и классом Предметов (подробности далее). Взаимодействие осуществляется посредством проверки, используя функция typeid() – для определение конкретного вида объекта, с которым будет осуществляться взаимодействие. Три функции (GetHealth (int health), AttackChange(int attack), ArmorChange(int armor)) принимают количество у.е. полученных от предмета (получение у.е. происходит через обращение к соответствующему предмету в методу Move() далее эта информация передается в медиатор, а медиатор передает информацию в GUI, который отображает изменения в специальной зоне под игровым полем с характеристиками, см. рис “GUI Игрока”)



GUI Игрока

Конкретное взаимодействие зависит от сущности, находящейся в клетке.

Так же Игрок имеет геттеры для передачи информации о себе.

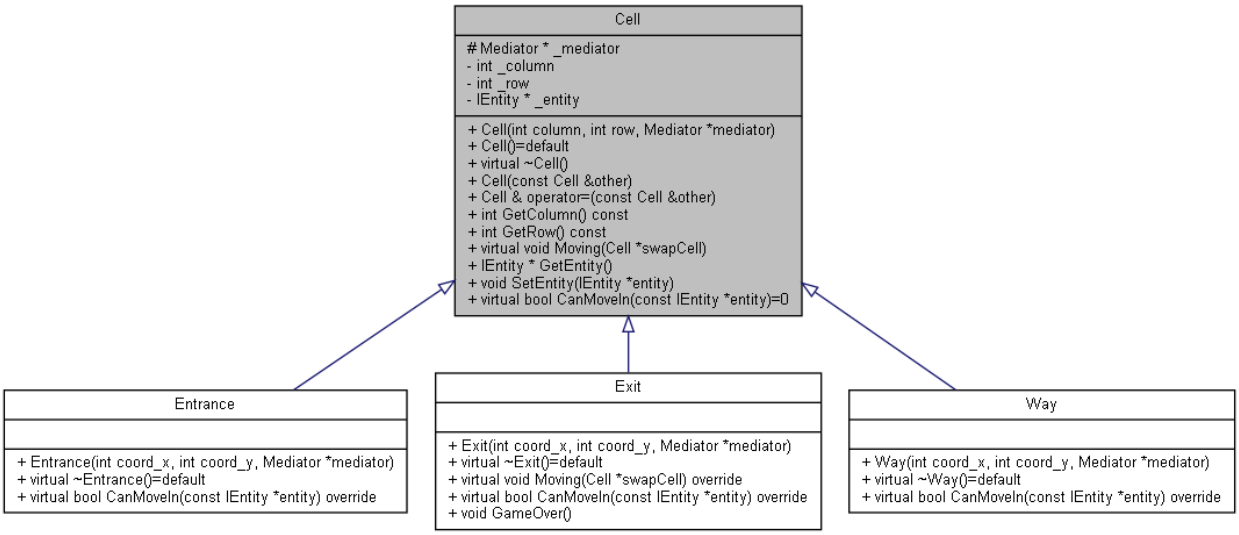
Абстрактный класс Enemy содержит поля-ссылки на клетку и игровое поле, на котором он находится, вектор векторов _direction (подразумевается вектор пар значений), поле _directionCount – отвечающее за возможное количество направлений, который хранит в каких направлениях может перемещаться Враг, ссылку на

EnemyMoveController – класс, отвечающий за выбор момента перемещения игрока, он содержит ссылку на Enemy, за которого он отвечает. От EnemyMoveController наследуется QtEnemyMoveController, чья логика работы зависит от фрейморка Qt и использует QTimer и QRandom для контролирования времени и рандомного выбора направления следующего хода Enemy (из возможно количества _directionCount). В конструкторе создается и запускается таймер на время _movingTime, по истечению которого используя систему сигналов-слотов запускается слот CallEnemyMove() – который запускает метод Move у соответствующего Enemy.

По функционалу класс Enemy дан имеет чистую функцию Move() – отвечающую за перемщение Врагов. Так же Enemy имеет геттеры для передачи информации о себе.

Данный класс наследуют три типа игрока (отличающиеся главным образом способом перемещения, а так же взаимодействия с игроком). У них есть общая черта: при проверке при перемещении они все используют специальную функцию CanMoveIn() клетки, которая

переопределена в клетках Входа, Выхода и Пути (напоминание о данной части проекта на рис. «Cells»



Cells

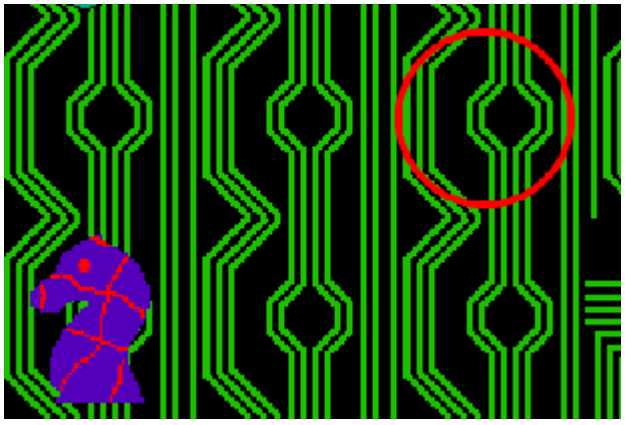
Функция CanMoveIn() переопределена в дочерних классах клетки таким образом, что Враги не могут пойти на клетку Входа и Выхода (а так же на клетки, в которых находятся предметы и другие враги, чтобы избежать поедания друг друга и случайное использование предметов, т.к на данный момент все Враги двигаются рандомно (хоть и по заданным напралениям))

Класс Virus – самый простой тип Врага, передвигается раз в 1 секунду, имеет всего 4 направления (вверх, вниз, влево, вправо на 1 клетку). В случае столкновения с игроком – игрок удаляется и игра

завершается посредством системы сигналов и слотов. В случае хода игрока на Вирус – Вирус удаляется, считается что игрок победил данного врага (игра НЕ завершается).

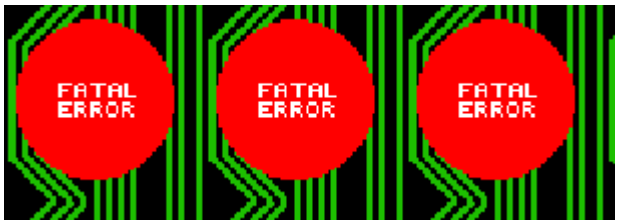
Virus

Класс Trojan – более сложныи тип ьрага, передвигается раз в 3 секунды, имеет всего 8 направлений (ходит как конь на шахматной доске). В случае столкновения с игроком – игрок удаляется и игра завершается посредством системы сигналов и слотов. В случае хода игрока на Троян – Троян удаляется, считается что игрок победил данного врага (игра НЕ завершается).



Trojan

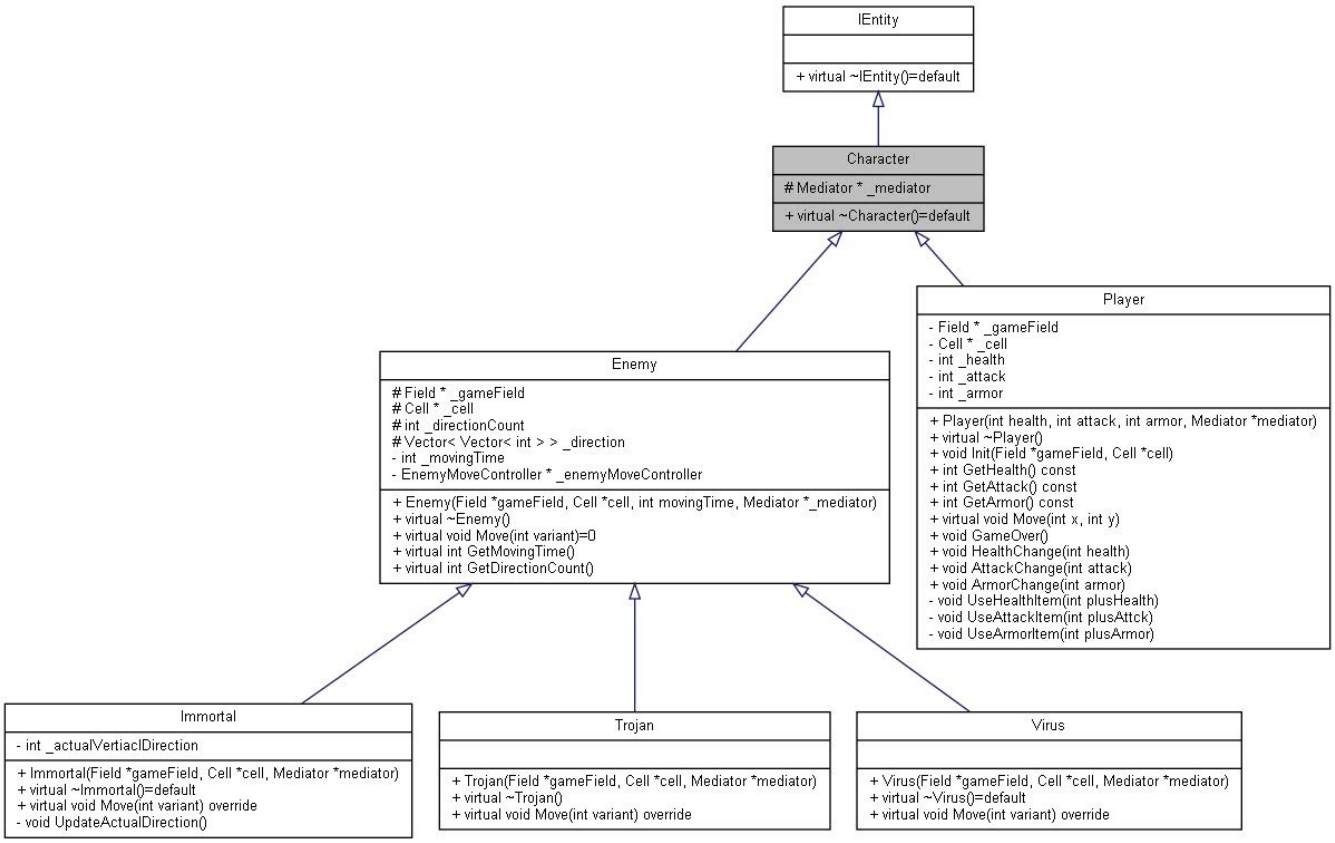
Класс Immortal – более сложныи тип ьрага, передвигается раз в 5 секунд, имеет всего 2 направления (вверх и вниз). Каждый ход данный класс рассчитывает возможные ходы и если оказывается, что ход он совершить не может, то направление меняется на противоположное. Данного Врага невозможно убить – при столкновении с Игроком в любом случае игра завершается, и считается, что Игрок проиграл.



Immortal

Многие данные (такие как) на момент 2 л.р. захаркодены, в последующем будет придумана более совершенная система инициализации этих данных.

Таким образом, получилась следующая структура по этим данным (в виде UML):



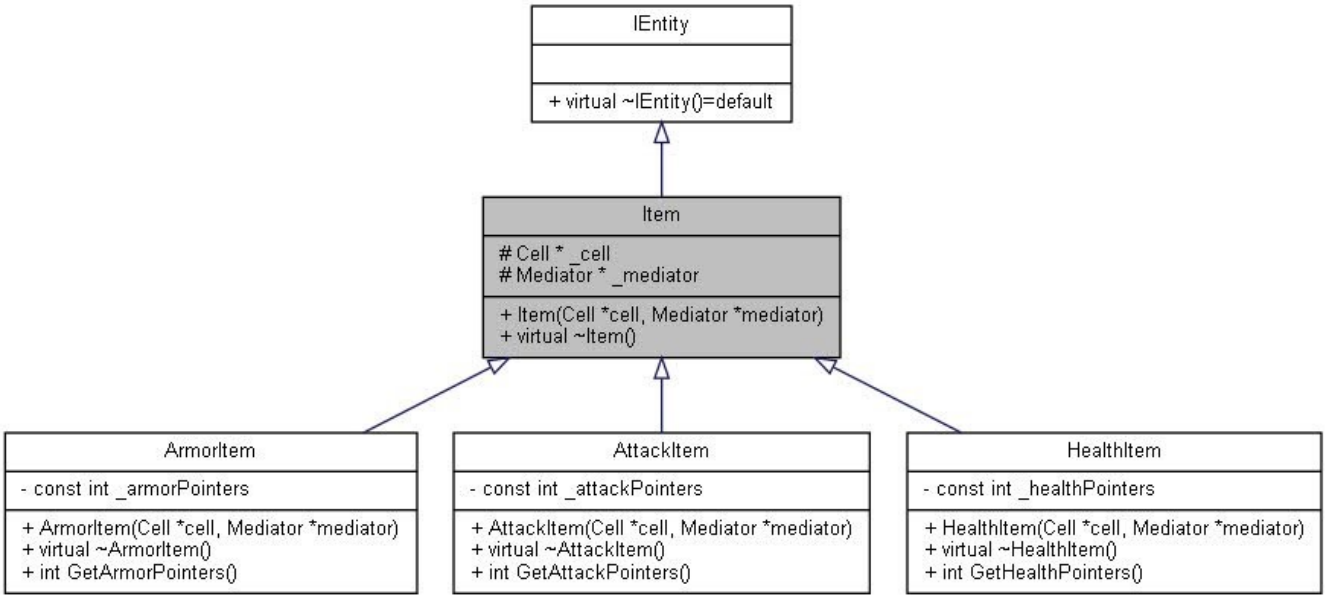
Entities

Так же класс IEntity реализует класс Item – предметы, которые так же могут находится на клетках. Сам класс имеет всего два поля: _mediator – ссылка на Медиатор и _cell – ссылка на клетку, в которой находится предмет (т.е. ассоциативная связь).

Класс Item наследуют три вида предметов, связанных с тремя характеристиками игрока (здоровьем, уроном и броней): HealthItem, AttackItem, ArmorItem. Три данных класса на момент 2 л.р. устроены очень схоже (в последующем вероятно будет больше изменений) – все они имеют очки, которые прибавляют к соответствующей характеристике игрока, геттер, с помощью которого происходит передача этих данных.

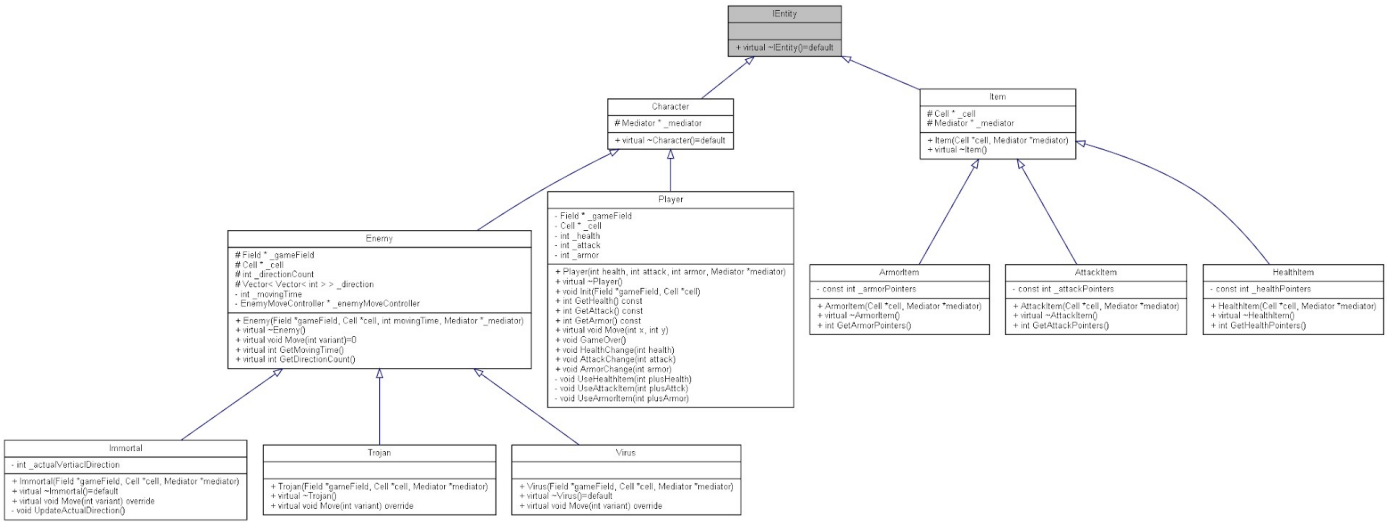
HealthItem/AttackItem/ArmorItem

Таким образом, получилась следующая структура по этим данным (в виде UML)



Items

Или:



Entities(full)

Взаимодействие с клеткой выхода у Игрока реализовано следующим образом: логика программы настроена так, что на клетку Выхода может попасть только игрок (это осуществляется проверкой CanMoveIn() клетки, на тип сущности, которая хочет попасть на клетку). У клетки Выхода переопределена виртуальная функция Move(), которая при вызове посылает сигнал в медиатор о том, что игра закончена.

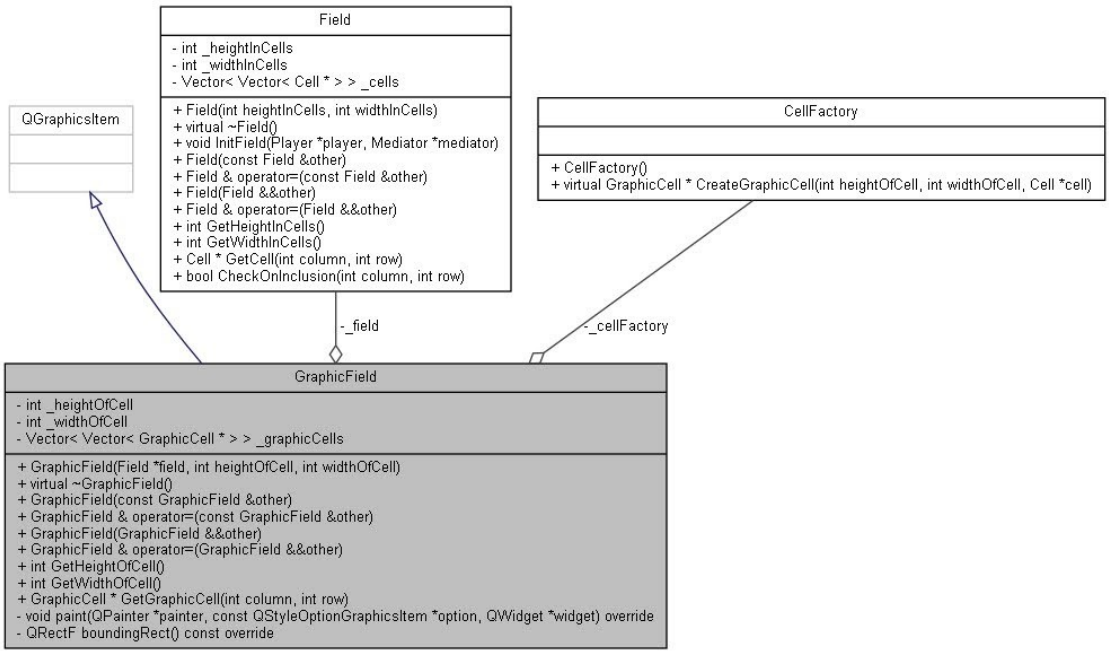
Для GUI были созданы отдельные классы GraphicCell, GraphicEntity и GraphicField.

GraphicEntity имеет два поля: ссылку на сущность, которую он отображает и поле Avatar. По функционалу класс имеет метод Draw, отвечающий за отрисовку. Создаются экземпляры данного класса с помощью фабрики EntutyFactory, которая вызывается при создании экземпляра GraphicCell, который так же имеет ссылку на клетку, которую он отображает, поле Avatar и поле GraphicCell – графической сущности, которая хранится в данной графической клетке.

По функционалу GraphicCell имеет метод Draw, отвечающий за отрисовку клетки на поле, метод EntitySwap, отвечающий за обмен графическими сущностями между графическими клетками (необходимо для соответствия изменений бизнес-логики в обычых клетках) и DeleteGraphicEntity – так же наобходимый для соответствия изменений бизнес логики в обычых клетках (а именно при удалении какой-то сущности).

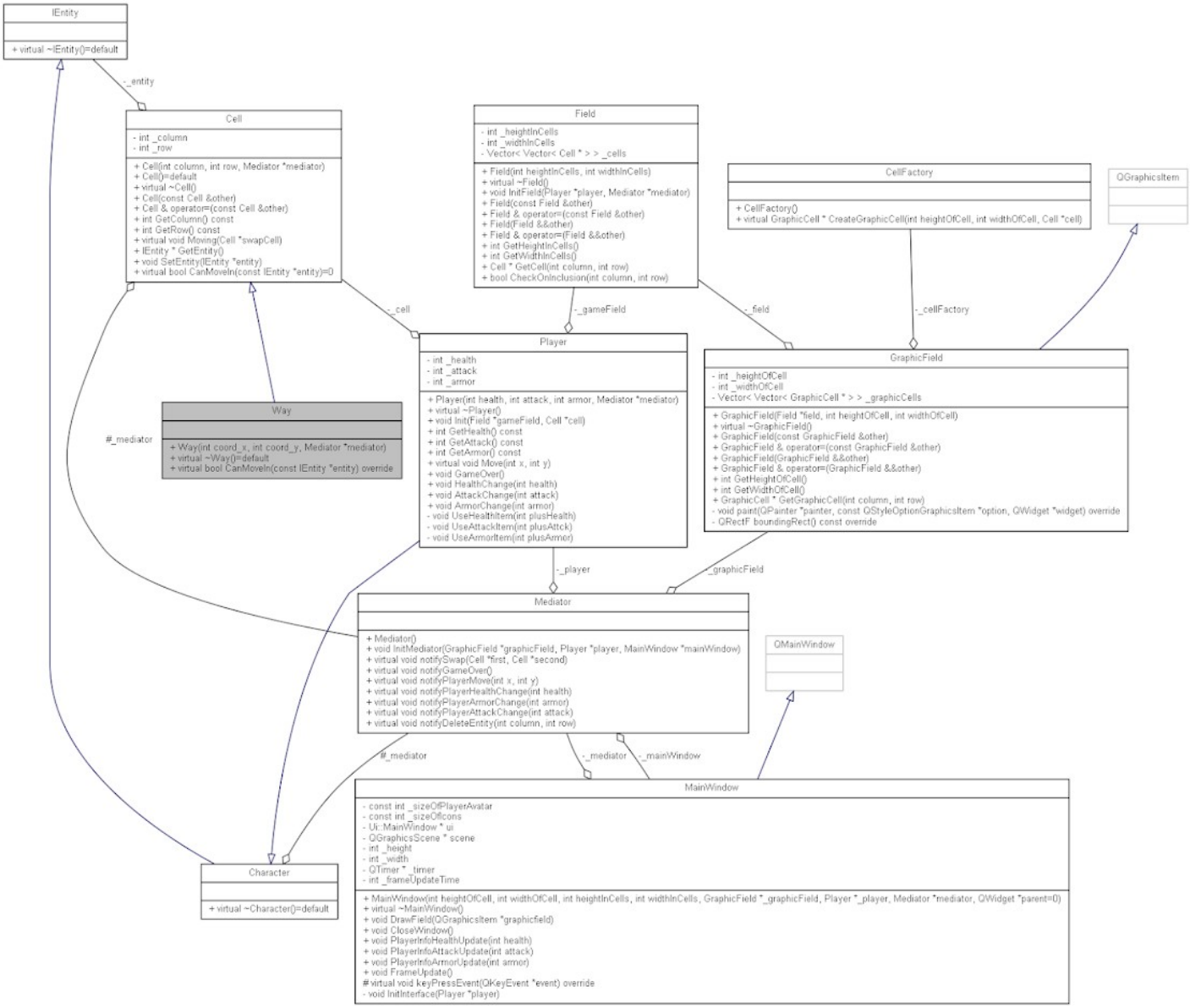
Метод CreateGraphicCell() класса CellFactory вызывается при создании GraphicField- класса, который наследуется от QGraphicsItem и имеет вектор векторов графических клеток.

Связь бизнес логики и графики осуществляется через Медиатор, который от бизнес-логики получает сообщение об изменении (например об удалении с поля какой-то сущности) и передает эту информацию графическому полю, вызывая соответствующий метод у нужной клетки (координаты которой так же передаются через Медиатор).



Field & GraphicField

Для того, чтобы объединить главные объекты, необходимые для игры (Field, GraphicField, Player, MainWindow и Mediator) был создан класс Game с полями-ссылками на вышеупомянутые классы. Конструктору класса Game передаются параметры поля (размеры клетки и размеры окна «в клетках»), после чего происходит создание медиатора, поля и игрока, после чего поле инициализируется, затем создается графическое поле и «окно». Затем необходимыми объектами (полем, игроком и графическим окном). Класс Game имеет метод Start() который использует метод show() класса QMainWindow – для создания самого окна.



Связи с Mediator

Выводы:

В рамках изучения полиморфизма были освоены интерфейсы, абстрактные классы, виртуальные функции. Созданы классы Персонажей и Предметов, а так же создано из взаимодействие между собой и полем. Был создан класс Медиатор, реализующий паттерн «Медиатор»(интерфейс, посредник) – через который осуществляется передача данных между разными модулями. Был создан контейнер по типу «Вектора», составлена UML таблица.