## Общее описание

* Все лабораторные работы являются работой над одним проектом. Лабораторные работы выполняются индивидуально. **Перед выполнением рекомендуется ознакомиться с текстом всех лаб. работ.**
* Лабораторная работа считается сданной, если выполнены все требования.
* За лабораторную работу может быть поставлено не полное количество баллов. Для каждой лаб. работы существует **одна попытка** повысить баллы за счет исправления замечаний.
* В конце файла присутствуют требования, которые относятся ко всему проекту сразу. Данные требования можно не выполнять, но они дают дополнительные баллы.

## Требования к коду

* Обязательное соблюдение codestyle: не должно быть транслитераций, названия должны быть логичными, код не должен быть перегружен комментариями, в одном файле должен быть описан/реализован один класс *(в файле с функцией main классов не должно быть)*. (ориентир для соблюдения codestyle<https://google.github.io/styleguide/cppguide.html>).
* Проект должен быть написан полностью в ООП стиле. Все функции должны быть методами классов (кроме main). Минимизация static функций. Функция main должна содержать только вызов метода запускающего всю бизнес-логику.
* Реализация лаб. работ должна быть на чистом C++ с использованием стандартной библиотеки. Сторонние библиотеки можно использовать только для реализации GUI.
* Необходимо использовать C++ стандарта не менее 17.
* Названия .h и .cpp файлов должны совпадать с названием класс, который описан/реализован в этих файлах.
* Файлы должны быть иерархически упорядочены по директориям.
* Все классы должны гарантировать сохранение инварианта.
* Не должно быть прямого взаимодействия классов бизнес-логики и классов интерфейса.
* Запрещается дублирование кода, в том числе смысловое.

## Требования к отчету

* Отчет должен быть оформлен согласно шаблону с ЛЭТИ
* Для каждой лаб. работы должны быть UML-диаграммы. Диаграммы должны отображать только то, что было сделано в рамках лаб. работы.
* Отчет должен содержать описание архитектурных решений. Почему были сделаны классы в лаб. работе, почему так связаны, за что отвечают, и т.д. **Простое перечисление полей и методов классов этому не удовлетворяют.**
* В отчете должна быть отражена проверка классов и то, что программа работает.

## Сроки сдачи лаб. работ

В таблице указана дата, в которую лаб. работу можно сдать без потери баллов. Если лаб. работа сдана после указанной даты, то максимальное кол-во баллов за нее уменьшается в 2 раза.

| № ЛР | 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- |
| Дата | 10.10 | 24.10 | ~~7.11~~ 14.11 | 28.11 | 12.12 | 26.12 |

## Получение баллов за семестр

| Лаб. работы | 10 баллов за одну лаб. работу. Суммарно 60 |
| --- | --- |
| Дополнительные требования | Суммарно 30 |
| Контрольная работа | 10 |
| **Можно набрать баллов за курс** | **100** |

## Оценка за семестр

**Оценка выставляется только в случае если все лаб. работы сданы**

| **Оценка** | **Минимальное необходимое кол-во баллов** |
| --- | --- |
| Отлично | 77 |
| Хорошо | 55 |
| Удовлетворительно | 33 |

# Лаб. работа №1 - Создание классов, конструкторов и методов классов

Игровое поле представляет из себя прямоугольную плоскость разбитую на клетки. На поле на клетках в дальнейшем будут располагаться игрок, враги, элементы взаимодействия. Клетка может быть проходимой или непроходимой, в случае непроходимой клетки, на ней ничего не может располагаться. На поле должны быть две особые клетки: вход и выход. В дальнейшем игрок будет появляться на клетке входа, а затем выполнив определенный набор задач дойти до выхода.

*При реализации класса поля запрещено использовать контейнеры из stl*

Требования:

* Реализовать класс поля, который хранит набор клеток в виде двумерного массива.
* Реализовать класс клетки, которая хранит информацию о ее состоянии, а также того, что на ней находится.
* Создать интерфейс элемента клетки (объект, который хранится на клетке).
* Обеспечить появление клеток входа и выхода на поле. Данные клетки не должны быть появляться рядом.
* Для класса поля реализовать конструкторы копирования и перемещения, а также соответствующие операторы.
* Гарантировать отсутствие утечки памяти.

*Потенциальные паттерны проектирования, которые можно использовать:*

* *Итератор (Iterator) - обход поля по клеткам и получение косвенного доступа к ним*
* *Строитель (Builder) - предварительное конструирование поля с необходимым параметрами. Например, предварительно задать кол-во непроходимых клеток и алгоритм их расположения*

# Лаб. работа №2 - интерфейсы, полиморфизм

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"*

# Лаб. работа №3 - логирование, перегрузка операций

Необходимо проводить логирование того, что происходит во время игры.

Требования:

* Реализован класс логгера, который будет получать объект, который необходимо отслеживать, и при изменении его состоянии записывать данную информацию.
* Должна быть возможность записывания логов в файл, в консоль или одновременно в файл и консоль.
* Должна быть возможность выбрать типа вывода логов
* Все объекты должны логироваться через перегруженный оператор вывода в поток.
* Должна соблюдаться идиома RAII

*Потенциальные паттерны проектирования, которые можно использовать:*

* *Адаптер (Adapter) - преобразование данных к нужному формату логирования*
* *Декоратор (Decorator) - форматирование тВВЦекста для логирования*
* *Мост (Bridge) - переключение между логированием в файл/консоль*
* *Наблюдатель (Observer) - отслеживание объектов, которые необходимо логировать*
* *Синглтон (Singleton) - гарантия логирования в одно место через одну сущность*
* *Заместитель (Proxy) - подстановка и выбор необходимого логирования*

# Лаб. работа №4 - шаблонные классы, управление

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 параметризуется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

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

*Потенциальные паттерны проектирования, которые можно использовать:*

* *Компоновщик (Composite) - выстраивание иерарихии правил*
* *Фасад (Facade) - предоставления единого интерфейса игры для команд управления*
* *Цепочка обязанностей (Chain of Responsibility) - обработка поступающих команд управления*
* *Состояние (State) - отслеживание состояние хода / передача хода от игрока к врагам*
* *Посредник (Mediator) - организация взаимодействия элементов бизнес-логик*и

# Лаб. работа №5 - управление, разделение на уровни абстракции

Необходимо организовать управление игрой (номинально через CLI). При управлении игрой с клавиатуры должна считываться нажатая клавиша, после чего происходит перемещение игрок или его взаимодействия с другими элементами поля.

Требования:

* Реализовать управление игрой. Считывание нажатий клавиш не должно происходить в классе игры, а должно происходить в отдельном наборе классов.
* Клавиши управления не должны жестко определяться в коде. Например, это можно определить в отдельном классе.
* Классы управления игрой не должны напрямую взаимодействовать с элементами игры (поле, клетки, элементы на клетках)
* Игру можно запустить и пройти.

*Потенциальные паттерны проектирования, которые можно использовать:*

* *Команда (Command) - передача команд с информацией через единый интерфейс. помещение команд в очередь*
* *Посредник (Mediator) - организация взаимодействия различных модулей*

# Лаб. работа №6 - сериализация, исключения

Сериализация - это сохранение в определенном виде состоянии программы с возможностью последующего его восстановления даже после закрытия программы. В рамках игры, это сохранения и загрузка игры.

Требования:

* Реализовать сохранения всех необходимых состояний игры в файл
* Реализовать загрузку файла сохранения и восстановления состояния игры
* Должны быть возможность сохранить и загрузить игру в любой момент
* При запуске игры должна быть возможность загрузить нужный файл
* Написать набор исключений, который срабатывают если файл с сохранением некорректный
* Исключения должны сохранять транзакционность. Если не удалось сделать загрузку, то программа должна находится в том состоянии, которое было до загрузки. То есть, состояние игры не должно загружаться частично

*Потенциальные паттерны проектирования, которые можно использовать:*

* *Снимок (Memento) - получение и восстановления состояния объектов при сохранении и загрузке*

# Дополнительные требования к проекту

Данные требования не обязательно выполнять

* Разработан и реализован GUI для программы - 6 баллов
* Соблюдены принципы SOLID - 8 баллов
* Для всех критических ситуаций (кроме описанных в л.р.6) написаны и обрабатываются исключения - 5 баллов
* Везде вместо обычных указателей используются умные указатели - 3 балла
* Корректно используются паттерны проектирования (не менее 5 штук) - 8 баллов