Общее описание

1. Лабораторные работы - это поэтапная работа над одним проектом. **В связи с этим, перед выполнением работ, рекомендуется предварительно ознакомиться с текстом всех работ.**
2. Лабораторные работы выполняются индивидуально.
3. За лаб. работу может быть выставлено неполное количество баллов в случае наличия ошибок, невыполнения требований или замечаний.

Требования к коду

1. Обязательное соблюдение стиля кода. Отсутствие транслитераций в названиях переменных, функций, классов и.т.д. Все идентификаторы должны быть “говорящими”, то есть по названию должен быть понятен смысл идентификатора. Вложенность блоков должна быть обозначена табуляцией. *Подробный гайд* *https://google.github.io/styleguide/cppguide.html*
2. Весь проект должен быть написан в ООП стиле. Все функции кроме main должны быть методами классов. Количество static методов должно быть минимально, а их использование четко обосновано. В функции main должна быть только инициализация необходимых объектов и запуск логики.
3. Весь код должен быть написан на чистом C++ с использованием стандартной библиотеки. Сторонние библиотеки можно использовать только для написание GUI или CLI.
4. Для каждого класса должна быть своя пара файлов формата .cpp и .h, названия которых совпадает с названием класса.
5. Все классы должны реализованы так, чтобы гарантировалось сохранение инварианта.
6. Запрещается дублирование кода, в том числе смысловое.
7. Не должно быть “божественных” классов.
8. Проект должен быть разбить на логические модули. Модуль отвечающий за логику игры, модуль с набором классов для отображения на экране и вводом пользователя, модуль с вспомогательными классами, так далее. **В классах связанной с логикой игры не должно быть ничего связанного с вводом и выводом информации для пользователя.**

Требования к отчету

1. Отчет должен быть оформлен согласно шаблону с ЛЭТИ
2. Для каждой лаб. работы должны быть UML-диаграммы. Диаграммы должны отображать только то, что было сделано в рамках лаб. работы.
3. 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.
4. В отчете должна быть отражена проверка классов и то, что программа работает.

Сроки сдачи лаб. работ

Для каждой лаб. работы есть дедлайн. Для получения полного балла за лаб. работу, необходимо сдать лаб. работу до дедлайна. Если лаб. работа сдана через неделю после дедлайна, то максимум можно получить половину баллов. Работы, сданные через 2 недели после дедлайна, максимально оцениваются в 1 балл.
Если за лаб. работу получен неполный балл и она показывалась до дедлайна, то есть 1 попытка повысить балл устранив недочеты в течении недели после дедлайна.
Для допуска к зачету необходимо сдать все лаб. работы.
Невыполнение (отсутствие реализации) какого-либо требования означает, что лаб. работа не сделана, и соответственно за нее не ставится какого-либо балла.

Л.р.	1	2	3	4	5	6	**7-9**
Дата	25.09	~~09.10~~ 16.10	~~23.10~~ 30.10	~~13.11~~ 20.11	~~04.12~~ 11.12	~~18.12~~ 25.12	***25.12***

Оценка за семестр

Получение баллов

- Каждая лаб. работа оценивается в 10 баллов
- Каждая контрольная оценивается в 10 баллов
- Корректное и обоснованное применение паттернов проектирования. 2 балла за паттерн, максимум 14.

Получение оценки

Для допуска к зачету необходимо сдать все лаб. работы

Оценка	**Мин. кол-во баллов**
Удовлетворительно	~~20~~ 37
Хорошо	~~42~~ 50
Отлично	~~64~~ 73

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

Реализовать прямоугольное игровое поле, состоящее из клеток. Клетка - элемент поля, которая может быть проходима или нет (определяет, куда может стать игрок), а также содержит какое-либо событие, которое срабатывает, когда игрок становится на клетку.

Для игрового поля при создании должна быть возможность установить размер (количество клеток по вертикали и горизонтали). Игровое поле должно быть зациклено по вертикали и горизонтали, то есть если игрок находится на правой границе и идет вправо, то он оказывается на левой границе (аналогично для всех краев поля).

Реализовать класс игрока. Игрок - сущность контролируемая пользователем. Игрок должен иметь свой набор характеристик и различный набор действий (например, разные способы перемещения, попытка избежать событие, и так далее).

Требования:

- Реализован класс игрового поля
- Для игрового поля реализован конструктор с возможностью задать размер и конструктор по умолчанию (то есть конструктор, который можно вызвать без аргументов)
- Реализован класс интерфейс события (в данной лабораторной это может быть пустой абстрактный класс)
- Реализован класс клетки с конструктором, позволяющим задать ей начальные параметры.
- Для клетки реализованы методы реагирования на то, что игрок перешел на клетку.
- Для клетки реализованы методы, позволяющие заменять событие. (То есть клетка в ходе игры может динамически меняться)
- Реализованы конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки
- Реализован класс игрока минимум с 3 характеристиками. И соответствующие ему конструкторы.
- Реализовано перемещение игрока по полю с проверкой допустимости на переход по клеткам.
- Игровое поле должно быть зациклено по вертикали и горизонтали

Примечания:

- При написании конструкторов учитывайте, что события должны храниться по указателю для соблюдения полиморфизма
- Для управления игроком можно использовать медиатор, команду, цепочку обязанностей

Лаб. работа №2 - Интерфейсы, динамический полиморфизм

Реализовать систему событий. Событие - сущность, которая срабатывает при взаимодействии с игроком. Должен быть разработан класс интерфейс общий для всех событий, поддерживающий взаимодействие с игроком. Необходимо создать несколько групп разных событий реализуя унаследованные от интерфейса события (например, враг, который проверяет условие, будет ли воздействовать на игрока или нет; ловушка,

которая безусловно воздействует на игрока; событие , которое меняет карту; и.т.д.). Для каждой группы реализовать конкретные события, которые по разному воздействуют на игрока (например, какое-то событие заставляет передвинуться игрока в определенную сторону, а другое меняет характеристики игрока). Также, необходимо предусмотреть событие “Победа/Выход”, которое срабатывает при соблюдении определенного набора условий.
Реализовать ситуацию проигрыша (например, потери всего здоровья игрока) и выигрыша игрока (добрался и активировал событие “Победа/Выход”)

Требования:

- Разработан интерфейс события с необходимым описанием методов
- Реализовано минимум 2 группы событий (2 абстрактных класса наследников события)
- Для каждой группы реализовано минимум 2 конкретных события (наследники от группы события)
- Реализовано минимум одно условное и безусловное событие (условное - проверяет выполнение условий, безусловное - не проверяет).
- Реализовано минимум одно событие, которое меняет карту (меняет события на клетках или открывает расположение выхода или делает какие-то клетки проходимыми (на них необходимо добавить события) или не непроходимыми
- Игрок в гарантированно имеет возможность дойти до выхода

Примечания:

- Классы событий не должны хранить никакой информации о типе события (никаких переменных и функций дающих информацию о типе события)
- Для создания события можно применять абстрактную фабрику/прототип/строитель

Лаб. работа №3 - логирование, перегрузка операций

Реализовать класс/набор классов отслеживающих изменения состояний в программе. Отслеживание должно быть 3-х уровней:

1. Изменения состояния игрока и поля, а также срабатывание событий
2. Состояние игры (игра начата, завершена, сохранена, и.т.д.)
3. Отслеживание критических состояний и ошибок (поле инициализировано с отрицательными размерами, игрок попытался перейти на непроходимую клетку, и.т.д.)

Реализованы классы для вывода информации разных уровней для в консоль и в файл с перегруженным оператором вывода в поток.

Требования:

- Разработан класс/набор классов отслеживающий изменения разных уровней

- Разработаны классы для вывода в консоль и файл с соблюдением идиомы RAII и перегруженным оператором вывода в поток.
- Разработанные классы спроектированы таким образом, чтобы можно было добавить новый формат вывода без изменения старого кода (например, добавить возможность отправки логов по сети)
- Выбор отслеживаемых уровней логирования должен происходить в runtime
- В runtime должен выбираться способ вывода логов (нет логирования, в консоль, в файл, в консоль и файл)

Примечания:

- Отслеживаемые сущности не должны ничего знать о сущностях, которые их логируют
- Уровни логирования должны быть заданными отдельными классами или перечислением
- Разные уровни в логах должны помечаться своим префиксом
- Рекомендуется сделать класс сообщения
- Для отслеживания изменений можно использовать наблюдателя
- Для вывода сообщений можно использовать адаптер, прокси и декоратор

Лаб. работа №4 - уровни абстракции, управление игроком

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы (начать новую игру, завершить игру, сохраниться, управление игроком, и.т.д.). Команды/клавиши определяющие управление должны считываться из файла.

Требования:

- Реализован класс/набор классов обрабатывающие команды
- Управление задается из файла (определяет какая команда/нажатие клавиши отвечает за управление. Например, w - вверх, s - вниз, и.т.д)
- Реализованные классы позволяют добавить новый способ ввода команд без изменения существующего кода (например, получать команды из файла или по сети). По умолчанию, управление из терминала или через GUI, другие способы реализовывать не надо, но должна быть такая возможность.
- Из метода считывающего команду не должно быть “прямого” управления игроком

Примечания:

- Для реализации управления можно использовать цепочку обязанностей, команду, посредника, декоратор, мост, фасад

Лаб. работа №5 - шаблонные классы, генерация карты

Реализовать шаблонный класс генерирующий игровое поле. Данный класс должен параметризироваться правилами генерации (расстановка непроходимых клеток, как и в каком количестве размещаются события, расположение стартовой позиции игрока и выхода, условия победы, и.т.д.). Также реализовать набор шаблонных правил (например, событие встречи с врагом размещается случайно в заданном в шаблоне параметре, отвечающим за количество событий)

Требования:

- Реализован шаблонный класс генератор поля. Данный класс должен поддерживать любое количество правил, то есть должен быть variadic template.
- Класс генератор создает поле, а не принимает его.
- Класс генератор не должен принимать объекты классов правил в каком-либо методе, а должен сам создавать (в зависимости от реализации) объекты правил из шаблона.
- Реализовано не менее 6 шаблонных классов правил
- Классы правила должны быть независимыми и не иметь общего класса-интерфейса
- При запуске программы есть возможность выбрать уровень (не менее 2) из заранее заготовленных шаблонов
- Классы правила не должны быть только “хранилищем” для данных.
- Так как используются шаблонные классы, то в генераторе не должны быть dynamic_cast

Примечания:

- Для задания способа генерации можно использовать стратегию, компоновщик, прототип
- Не рекомендуется делать static методы в классах правилах

Лаб. работа №6 - сериализация, исключения

Реализовать систему классов позволяющих проводить сохранение и загрузку состояния игры. При загрузке должна соблюдаться транзакционность, то есть при неудачной загрузки, состояние игры не должно меняться. Покрыть программу обработкой исключительных состояний.

Требования:

- Реализована загрузка и сохранение состояния игры
- Сохранение и загрузка могут воспроизведены в любой момент работы программы.
- Загрузка может произведена после закрытия и открытия программы.

- Программа покрыта пользовательскими исключениями.
- Пользовательские исключения должны хранить полезную информацию, например значения переменных при которых произошло исключение, а не просто сообщение об ошибке. Соответственно, сообщение об ошибке должно учитывать это поля, и выводить информацию с учетом значений полей.
- Исключения при загрузке обеспечивают транзакционность.
- Присутствует проверка на корректность файла сохранения. (Файл отсутствует; в файле некорректные данные, которые нарушают логику; файл был изменен, но данные корректны с точки зрения логики).

Примечания:

- Исключения должны обрабатываться минимум на фрейм выше, где они были возбуждены
- Для реализации сохранения и загрузки можно использовать мементо и посетителя
- Для проверки файлов можно рассчитывать хэш от данных.

Лаб. работа №7 - Реализация простой системы персонажей, управляемых компьютером

Реализовать систему NPC (персонажей, управляемых компьютером). Такого набора сущностей, которые перемещаются по игровому полю и могу взаимодействовать с игроком. Данные сущности должны иметь общий интерфейс. NPC должны быть следующих типов: враждебные, мирные и нейтральные. Враждебные влияют на игрока негативно, мирные позитивно, а нейтральные могут влиять на игрока негативно или позитивно в зависимости от условий. Если игрок должен перейти на клетку с NPC, то он остается на месте, а открывается диалоговое окно взаимодействия с NPC. Диалоговое окно должно позволять игроку решать как взаимодействовать с NPC. Например, с враждебным NPC (враг) можно решать сражаться или попытаться избежать боя (с какой-то вероятностью) или враг может попытаться сбежать сам. Мирным NPC может быть какой-то торговец, а нейтральным сущность, которая мирная к игроку, пока он не активировал какое-то событие. Перемещение NPC по полю должно задавать через шаблон, то есть, враги с разными параметрами шаблона перемещаются по разному (например, один пытается дойти до игрока, второй просто патрулирует территорию, а третий стоит на месте). Также, враждебные NPC могут сами инициировать взаимодействие с игроком.

Требования:

- Разработан интерфейс NPC
- Реализованы минимум три вида NPC: враждебный, мирный, нейтральный
- Перемещение NPC задается через параметр шаблона
- Реализована система диалога с NPC

- Генератор поля дополнен правилами размещения NPC на поле
- Добавлена возможность уничтожения игроком враждебных NPC

Примечания:

- Можно использовать паттерн состояние для передачи хода
- Можно использовать паттерн итератор для управления всеми NPC
- Можно использовать паттерн строитель для более глубокого конструирования NPC
- Можно использовать паттерн легковес для уменьшения количества потребляемой памяти NPC

Лаб. работа №8 - Разработка динамической системы инвентаря игрока

Разработать систему инвентаря игрока и его экипировки (вещей). Класс игрока должен содержать инвентарь, который содержит список экипировки, который у него есть. Инвентарь ограничен суммарным весом вещей. В случае, если игрок пытается получить новую вещь, и веса не хватает, то он не может получить вещь в инвентарь. Разработать систему классов, реализующих систему вещей. Все вещи могут храниться в инвентаре. Вещи делятся на две категории: можно надеть на игрока и можно использовать. Вещи, которые можно надеть, можно снять или заменить на другую. Вещи, которые используются, после использования исчезают.. При использовании вещи, которая используется, игрок тратит ход и никуда не двигается (например, использование зелья лечения), при надевании вещи, ход не должен тратиться. Вещи, которые надеваются, каким-то образом улучшают характеристики игрока пока надеты. Предусмотреть систему получения игроком экипировки (в ходе взаимодействия событий или NPC).

Требования:

- Разработан интерфейс экипировки
- Реализовано минимум 2 класса экипировки, которую можно надеть (например, оружие и броня)
- Реализовано минимум 2 класса вещей, которые можно использовать
- Реализована система инвентаря игрока с возможность надевать/использовать вещи
- Реализована система получения экипировки игроком с учетом ограничений инвентаря

Примечания:

- Можно использовать фабрики для выдачи экипировки
- Можно использовать шаблонный метод для влияния экипировки
- Можно использовать мост для надеваемой экипировки

Лаб. работа №9 - Разработка системы динамического изменения карты

Разработать набор классов, реализующим систему динамического изменения части игрового поля (например, погода). Данные классы должны менять игровую карту разными способами: изменять передвижение игрока по полю, делать игровые события неактивными, менять проходимость клеток. Данная система должна срабатывать случайно вне зависимости от действий игрока, и менять лишь часть игрового поля. Например, туман делает передвижения игрока случайным, гроза делает неактивными события, сильный ветер позволяет проходить непроходимые клетки. Эффект от такого изменения должен длиться ограниченное количество ходов.

Требования:

- Разработан интерфейс изменения поля
- Реализованы три разных типа изменения поля, по разному влияющих на игровой процесс, и влияющих на разные систему
- Изменение поля должно длиться ограниченное количество ходов, после исчезновения, возвращать к изначальным состояниям. Но, если было подобрано событие, оно не должно восстанавливаться
- Изменение поля должно срабатывать случайным образом

Примечания:

- Можно использовать паттерн стратегия для изменения игрового поля
- Можно использовать паттерн состояние для переключения между статусом клеток/поля