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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «ООП»

Тема: логирование, перегрузка операций

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Узнать что такое логирование. Создать механизм логирования в своей программе.

Задание.

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 текста 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

Выполнение работы.

Для реализации программы и выполнения программы были созданы следующие классы :

Класс ***Observable*** :

Класс "наблюдаемый", прописывающий реализацию паттерна *Наблюдатель,* в связке с классом наблюдателя ***Logger***. Имеет ссылку на объект ***Logger*** и функцию ***notify()*** в которой обновляет наблюдателя .

Класс ***GameObject*** :

Класс игрового объекта , который дает возможность наследникам иметь свой личный идентификатор для каждого экземпляра .

Класс ***Logger*** :

Класс логера, который записывает сообщения об изменении объекта в логи. Является наблюдателем относительно объектов наследуемых от интерфейса ***Observable.*** Работает следующим образом - подписывается на все объекты (они должны быть унаследованы от ***Observable***), которые нужно логировать и имеет список логов (места куда записывается сообщение). Наблюдаемые объекты через функцию ***notify()*** передают логерру объект класса ***Event***, который хранит информацию о событии. Логер генерирует сообщение на основе этой информации и записывает его в логи .

Класс ***ILog*** :

Интерфейс для логов. Обязывает лог иметь поток, в который будутся выводится сообщения . Также предоставляет функцию ***update()***, которая если нужно обновляет лог после записи в него.

Класс ***FileLog*** :

Класс файлового лога. Наследуется от интерфейса ***ILog***. Предоставляет поток вывода в файл. Класс соблюдает идиому RAII . В конструкторе открывается файл для записи , в деструкторе закрывается.

Класс ***ConsoleLog*** :

Класс консольного лога. Наследуется от интерфейса ***ILog***. Предоставляет поток вывода в консоль. Так как сама игра отрисовывается в консоли, то сообщения о прошедших изменениях не выводятся в консоль а сохраняются в объекте. Затем после отрисовки вызывается функция ***update()***, которая переносит все сохранные сообщения на данном такте в консоль.

Классы Событий:

Это классы которые хранят информацию о произошедшем событием.
Классы событий:

IEvent - интерфейс любого события. Обязывает любой класс события иметь указатель на объект с которым произошло событие.

EventMove - событие перемещения;

EventReversal - событие разворота;

EventChangeHealth - событие изменения здоровья;

EventChangeArmor - событие изменения кол-ва брони;

EventChangeDamage - событие изменения наносимого урона;

EventToAttack - событие атаки кого-либо ;

EventSetLocation - событие размещения на поле;

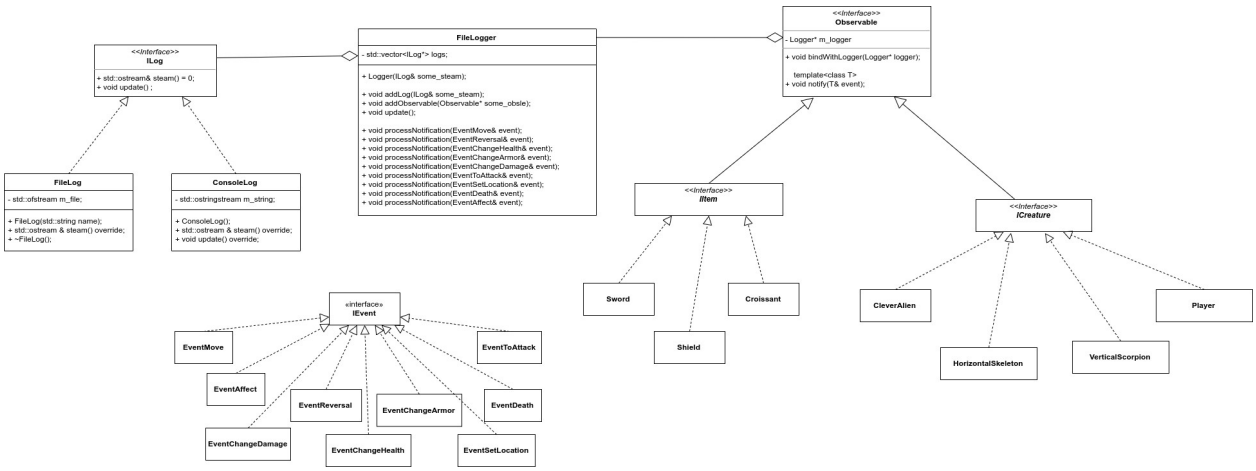
EventDeath - событие смерти объекта;

EventAffect - событие воздействия на другой объект;

Мелкие изменения в уже созданных классах:

Классы ***IItem, ICreature*** на наследуются от ***Observable*** , а следовательно и их потомки (***Sword, Shield, Croissant*** и ***Player, VerticalScorpion, HorizontalSkeleton, CleverAlien***). Дальше в каждом из этих классов в нужных местах прописывается вызов функции ***notify()***, для уведомления логера об изменениях в этих объектах.

UML диаграмма классов:



Тестирование.

main.ccp(самое важное):

```
//инициализация объектов
auto alien = std::make_unique<CleverAlien>();
auto sword = std::make_unique<Sword>();
```

```
unq_p<Player> player ;

//получение игрока
player = builder.buildStartCell(13, 7);

//инициализация логерра
FileLogger logger;
player->addObserver(&logger);
alien->addObserver(&logger);
sword->addObserver(&logger);

alien->follow(player.get());
alien->setLocation(10, 5, field1.get());
sword->setLocation(1, 1, std::move(sword), field1.get());

while ( true ){
    logger.writeEndStep();
    if (player->getAlive()){
        if(player->win()){
            break;
        }
        player->update();
    } else {
        break;
    }

    if (alien->getAlive()){
        alien->update();
    }
    view.rendering();
}
```

Результат:

Утечек памяти не обнаружено.

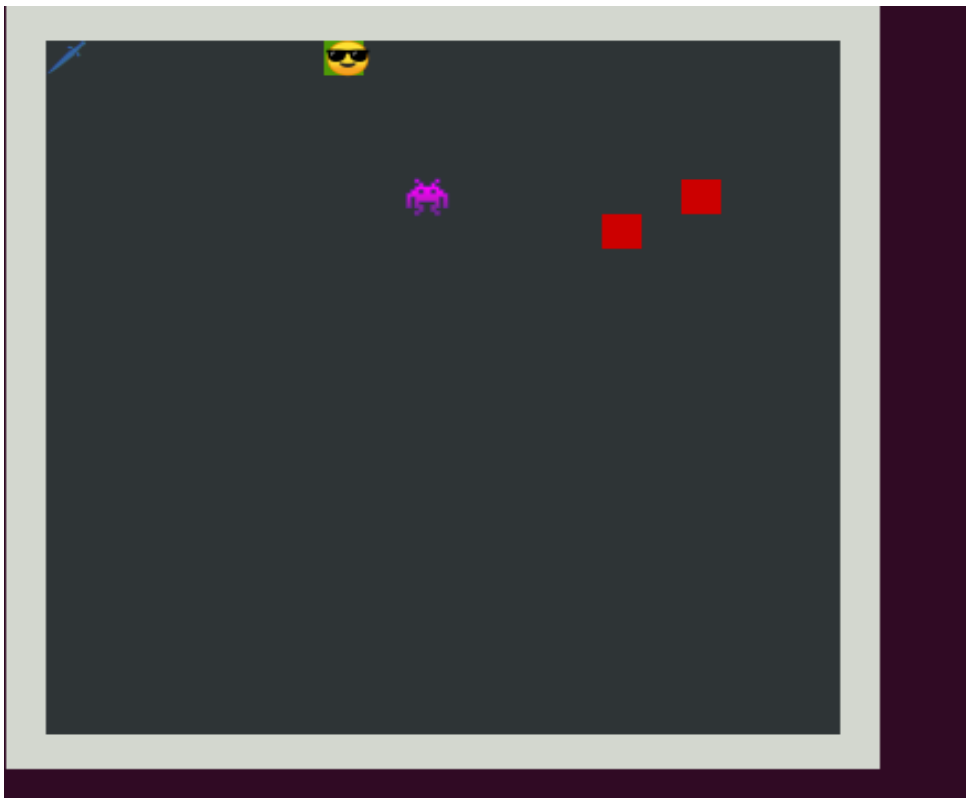


Рисунок 1: Исследование на утечки памяти

Содержание лога :

```
Пришелец 0 поставлен на координаты x = 10, y = 5
Меч 1 поставлен на координаты x = 1, y = 1
---------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 переместился вверх, текущее местоположение x = 10, y = 4
---------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 переместился вверх, текущее местоположение x = 10, y = 3
---------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 переместился влево, текущее местоположение x = 9, y = 3
---------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 переместился вверх, текущее местоположение x = 9, y = 2
---------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 переместился влево, текущее местоположение x = 8, y = 2
```

```
-----------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 50
Пришелец 0 атаковал Игрок 2
Игрок 2 получил урон 18, текущее здоровье 32
-----------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 33
Пришелец 0 атаковал Игрок 2
Игрок 2 получил урон 18, текущее здоровье 15
-----------------------------------------------------------------------
Игрок 2 восстановил здоровье на 1, текущее здоровье 16
Пришелец 0 атаковал Игрок 2
Игрок 2 получил урон 18, текущее здоровье 0
Игрок 2 умер
-----------------------------------------------------------------------
```

Программа работает корректно.

Выводы.

Был изучен механизм логирования и получен опыт в создании его на практике . Разработана программа, выполняющая поставленное задание.