Министерство образования Самарской области

АССОЦИАЦИЯ ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ "ИНСТИТУТ НАПРАВО.РУ"

УТВЕРЖДЕНО
Приказ директора
от 07.08.2024 № 012/2/с

Воробьев К.А.

УЧЕБНО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ

ПМ.01 Разработка программных модулей

Специальность 09.02.10 Разработка компьютерных игр, дополненной и виртуальной реальности
Квалификация: разработчик компьютерных игр, дополненной и виртуальной реальности
Форма обучения: очная
На базе основного общего образования
Срок получения образования: 3 года 10 месяцев

Тольятти, 2024г.

Содержание

1. Назначение и организация практикума 3

2. Состав работ и представление результатов 3

3. Практические работы 4

Работа 1. Контракт функции и обработка границ 4

Работа 2. Коллекция предметов 4

Работа 3. Модульное сохранение и восстановление 5

Работа 4. Отладка и минимальное воспроизведение 6

Работа 5. Контроль версий и конфликт 6

Работа 6. Измерение и рефакторинг 7

Развёрнутая лабораторная работа. Разработка и передача модуля инвентаря 8

4. Комплексное учебное задание 9

5. Термины и понятия 9

6. Исходные данные и учебные примеры 10

7. Использованные источники 10

1. Назначение и организация практикума

Реализация и сопровождение программных модулей игрового проекта: алгоритмы, код, отладка, версии и производительность.

Указания предназначены для обучающихся по специальности 09.02.10 «Разработка компьютерных игр, дополненной и виртуальной реальности», квалификация — разработчик компьютерных игр, дополненной и виртуальной реальности.

Компонент	Учебная нагрузка
Индекс и наименование	ПМ.01 Разработка программных модулей
Общий объём	340 академических часов
Практические занятия	146 академических часов
Промежуточная аттестация	Экзамен — семестр 5; Дифференцированный зачёт — семестр 5

Перед работой изучаются её исходные данные и проверяемые результаты. Затем выполняются последовательные действия, сопоставляется учебный пример и решаются самостоятельные варианты. Ответ включает ход работы и подтверждение результата. Использование готового примера без выполнения своего варианта не подтверждает освоение действия.

Подготовленные материалы: Исходный код и тесты; Контракты и инструкция запуска; История версий и результаты измерений.

2. Состав работ и представление результатов

Работа	Тема и результат
1	Контракт функции и обработка границ. Отрицательные аргументы не превращаются молча в допустимые.
2	Коллекция предметов. Не возникает отрицательное количество.
3	Модульное сохранение и восстановление. Все обязательные поля проверяются.
4	Отладка и минимальное воспроизведение. Исправлена причина повторного вызова.
5	Контроль версий и конфликт. История показывает вклад участников.
6	Измерение и рефакторинг. Результаты двух реализаций совпадают.

Профессиональные результаты модуля

Код	Профессиональная компетенция
ПК 1.1	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.
ПК 1.2	Разрабатывать программные модули в соответствии с техническим заданием.
ПК 1.3	Оформлять программный код в соответствии с установленными требованиями.
ПК 1.4	Использовать систему контроля версий программного кода для коллективной разработки программного кода.
ПК 1.5	Выполнять отладку программных модулей с использованием специализированных программных средств.
ПК 1.6	Осуществлять рефакторинг и оптимизацию программного кода.
ПК 1.7	Разрабатывать процедуры интеграции программных модулей и платформы/фреймворка.
ПК 1.8	Выполнять интеграцию программных модулей и платформы/фреймворка.
ПК 1.9	Осуществлять инсталляцию, настройку и обслуживание программного обеспечения.

В отчёте указываются тема, вариант, исходные данные, ход выполнения, проверка

результата и вывод. Таблицы и графики снабжаются названиями и единицами измерения, где они применимы. Результаты оцениваются по критериям фонда оценочных средств. Приложите исходный код модуля и результаты проверок.

3. Практические работы

Работа 1. Контракт функции и обработка границ

Краткие теоретические сведения

Контракт определяет допустимые аргументы, результат и поведение при ошибке. Тесты выводятся из условий, а не только из примера успешного запуска.

Исходные данные

Функция Damage(attack,defense): аргументы целые неотрицательные; результат max(0,attack−defense). Отрицательные аргументы отклоняются.

Порядок выполнения

1. Записать предусловия и постусловие.

2. Реализовать функцию отдельно от сцены.

3. Составить таблицу границ и ошибок.

4. Проверить независимость функции от интерфейса.

Разбор учебного примера

Для (12,5) результат 7; (5,12) — 0; (0,0) — 0. Для (−1,0) возникает ошибка аргумента. Полный класс C# и проверки приведены в приложении.

Задания для самостоятельного выполнения

А: добавить минимальный урон 1 только для положительной атаки.

Б: добавить верхнюю границу атаки 1000.

Проверка результата

— Отрицательные аргументы не превращаются молча в допустимые.

— После изменения контракта обновлены тесты.

Вопросы для самоконтроля

1. Где должен находиться контроль аргументов?

2. Почему вывод на экран не равен возврату результата?

Работа 2. Коллекция предметов

Краткие теоретические сведения

Коллекция отвечает за хранение и изменение, а представление — за показ. Удаление последней единицы и попытка удалить больше имеющегося требуют явных правил.

Исходные данные

Инвентарь: key=2, shield=1; операции Add и Remove возвращают состояние либо признак успешности.

Порядок выполнения

1. Определить инвариант количества.

2. Реализовать добавление положительного количества.

3. Реализовать удаление без отрицательного остатка.

4. Составить последовательность операций и ожидаемых состояний.

Разбор учебного примера

Remove(key,1) даёт key=1; повторное удаление — 0 и удаление записи. Следующая попытка возвращает отказ без изменения остальных предметов.

Задания для самостоятельного выполнения

А: добавить предмет с новым идентификатором.

Б: запросить удаление 3 единиц при наличии 2.

Проверка результата

— Не возникает отрицательное количество.

— Отказ не изменяет коллекцию.

Вопросы для самоконтроля

1. Кому принадлежит ответственность за вывод?

2. Как проверить неизменность после отказа?

Работа 3. Модульное сохранение и восстановление

Краткие теоретические сведения

Формат сохранения отделяется от внутренней структуры объектов. Версия данных позволяет обнаружить несовместимость, а проверка — отказаться от повреждённой записи.

Исходные данные

Состояние: version=1, level=3, score=120, inventory={key:2}.

Порядок выполнения

1. Описать поля и ограничения.

2. Сериализовать в текстовый формат.

3. Прочитать в новый объект и сравнить поля.

4. Проверить отсутствующее поле и неверную версию.

Разбор учебного примера

После восстановления новый объект должен содержать те же значения, но не быть той же ссылкой в памяти. Повреждённые данные не подменяются успешным сообщением.

Задания для самостоятельного выполнения

А: добавить настройку звука.

Б: прочитать запись version=99.

Проверка результата

— Все обязательные поля проверяются.

— Ошибка не уничтожает исходный файл.

Вопросы для самоконтроля

1. Зачем нужна версия формата?

2. Почему важен отдельный объект после чтения?

Работа 4. Отладка и минимальное воспроизведение

Краткие теоретические сведения

Отладка начинается с воспроизводимого условия и гипотезы. Исправление считается подтверждённым после повторения исходного случая и проверки соседних сценариев.

Исходные данные

Ошибка: счёт прибавляется дважды после повторного входа на сцену; обработчик события подключается при каждом входе.

Порядок выполнения

1. Составить шаги воспроизведения.

2. Проверить число подписок или вызовов.

3. Устранить повторное подключение либо корректно отключать его.

4. Выполнить три цикла входа и выхода с проверкой счёта.

Разбор учебного примера

Если обработчик подписан дважды, один сигнал вызывает два начисления. Ожидаемый результат каждого события — одно изменение; после трёх входов это правило сохраняется.

Задания для самостоятельного выполнения

А: повторная подписка создаётся после паузы.

Б: обработчик вызывается для уже уничтоженного объекта.

Проверка результата

— Исправлена причина повторного вызова.

— Есть отрицательный и регрессионный сценарии.

Вопросы для самоконтроля

1. Чем симптом отличается от причины?

2. Какие данные полезны в журнале?

Работа 5. Контроль версий и конфликт

Краткие теоретические сведения

Изменения фиксируют небольшими законченными порциями. Конфликт решают по смыслу требований, после чего выполняют сборку и проверки объединённой версии.

Исходные данные

Два участника изменяют правило начисления очков в одной функции: один меняет множитель, другой добавляет проверку отрицательного входа.

Порядок выполнения

1. Зафиксировать базовую версию.

2. В отдельных ветвях выполнить две правки.

3. Воспроизвести конфликт объединения.

4. Сохранить обе согласованные функции и проверить итог.

Разбор учебного примера

Правильное объединение не выбрасывает проверку входа ради нового множителя. В сообщении фиксации объясняют изменение поведения и связанные проверки.

Задания для самостоятельного выполнения

А: изменения касаются разных функций.

Б: требования действительно противоречат друг другу.

Проверка результата

— История показывает вклад участников.

— После разрешения конфликта нет служебных маркеров и падающих тестов.

Вопросы для самоконтроля

1. Почему «взять мою версию» не универсальное решение?

2. Что следует проверить после объединения?

Работа 6. Измерение и рефакторинг

Краткие теоретические сведения

Оптимизация требует измерения исходной версии, одинаковой нагрузки и проверки эквивалентности результата. Рефакторинг меняет структуру без намеренного изменения внешнего поведения.

Исходные данные

Поиск предмета повторяется для 10 000 запросов; сравниваются линейный просмотр и индекс по идентификатору.

Порядок выполнения

1. Подготовить одинаковые входные данные.

2. Зафиксировать время и результаты нескольких повторов.

3. Выполнить изменение структуры хранения.

4. Повторить измерения и проверить отсутствующий идентификатор.

Разбор учебного примера

Индекс может сократить поиск, но имеет стоимость создания и памяти. Нельзя сравнивать первый холодный запуск одного варианта с прогретым запуском другого без описания условий.

Задания для самостоятельного выполнения

А: число запросов 10.

Б: частые изменения списка между запросами.

Проверка результата

— Результаты двух реализаций совпадают.

— Вывод учитывает время подготовки и память.

Вопросы для самоконтроля

1. Когда индекс не окупается?

2. Как доказать сохранение поведения?

Развёрнутая лабораторная работа. Разработка и передача модуля инвентаря

Средства выполнения

Редактор C#, PowerShell для проверки учебного класса, система контроля версий. Файл TrainingModule.cs приведён полностью в разделе исходников.

Задание

Реализовать модуль с операциями добавления и удаления предметов, не зависящий от игрового интерфейса. Требуется сохранение неотрицательного количества, отсутствие побочного изменения при отказе и возможность повторить проверки другим участником.

Последовательность выполнения

1. Создайте каталог InventoryLab с подкаталогами src, tests и docs. Сохраните TrainingModule.cs в src. В README запишите назначение класса и способ запуска. Не включайте в репозиторий пароли и личные сведения.

2. Выполните в PowerShell команду Add-Type -Path ./src/TrainingModule.cs, затем [TrainingModule]::RunAll(). Исходный класс должен пройти включённые проверки. При повторной компиляции изменённого класса используйте новый сеанс PowerShell, чтобы ранее загруженный тип не скрывал изменения.

3. Запишите контракт Remove: amount>0; предмет должен существовать в нужном количестве; успешное удаление уменьшает количество; отказ оставляет словарь неизменным. Последняя единица удаляет запись. Укажите различие отказа по состоянию и ошибки недопустимого аргумента.

4. Составьте таблицу минимум из восьми проверок до изменения кода: частичное удаление, полное удаление, отсутствующий предмет, нехватка количества, нулевое и отрицательное amount, неизменность другого предмета, повторный вызов после удаления.

5. Добавьте функцию Add. Она принимает непустой идентификатор и положительное количество. Для нового идентификатора создаётся запись, для существующего количество увеличивается. Задайте и проверьте поведение при переполнении целого числа.

6. Расширьте RunAll: используйте отдельный начальный словарь для каждого независимого сценария. Снимок состояния до ошибочной операции сравнивайте со всем словарём после неё, а не только с возвращённым bool.

7. Внесите изменения двумя отдельными фиксациями: сначала контракт и проверки, затем реализация. В сообщении фиксации кратко укажите изменённое поведение. После изменения тестов убедитесь, что они действительно обнаруживают намеренно внесённый учебный дефект.

8. Проведите парную инспекцию. Проверяющий объясняет предусловия и находит все места изменения словаря. Если чтение и изменение смешаны с выводом сообщений на экран, выделите представление в отдельный код.

9. Передайте другому участнику чистую копию исходников и инструкцию. Проверяющий запускает класс без доступа к вашему рабочему каталогу. Зафиксируйте обнаруженные зависимости и исправьте инструкцию.

10. В итоговом отчёте приложите контракт, таблицу фактических результатов, фрагмент

истории версий и пример одного обнаруженного дефекта. Код сопровождается объяснением личного вклада и ограничений модуля.

Контрольные сценарии и ожидаемые результаты

Сценарий	Ожидаемый результат
key=2; Remove(key,1)	Успех, key=1
key=1; Remove(key,1)	Успех, записи key нет
key=2; Remove(key,3)	Отказ, key остаётся 2
Предмет отсутствует	Отказ без создания записи
amount=0 или −1	Определённая ошибка аргумента
Удаляется key, существует shield=1	Количество shield не меняется

Усложнённый вариант

Добавьте ограничение общего веса инвентаря. Сначала определите, где хранится вес единицы предмета и как считается вес всей коллекции, затем подготовьте три проверки: точная граница, превышение и добавление существующего предмета. Не меняйте правило удаления при отказе.

4. Комплексное учебное задание

Разработайте библиотеку игрового состояния с расчётом урона, инвентарём, сохранением, журналом ошибок и тестами. Передайте библиотеку другому участнику с контрактами функций и воспроизводимым запуском. Покажите историю не менее пяти осмысленных фиксаций и один разбор дефекта.

Этапы и предъявляемые материалы

Этап	Результат
1	Исходный код и тесты
2	Контракты и инструкция запуска
3	История версий и результаты измерений

Выполненная работа представляется с исходными данными, объяснением принятых решений и проверками. Изменения исходного условия согласуются до выполнения расчётов или реализации. При защите обучающийся воспроизводит один основной и один граничный либо отрицательный сценарий и объясняет полученный результат.

Контроль готовности

— Отрицательные аргументы не превращаются молча в допустимые.

— Не возникает отрицательное количество.

— Все обязательные поля проверяются.

— Исправлена причина повторного вызова.

— История показывает вклад участников.

— Результаты двух реализаций совпадают.

5. Термины и понятия

Термин	Содержание
Контракт	Условия вызова и гарантируемый результат операции.

Код	Профессиональная компетенция
Инвариант	Свойство состояния, сохраняемое допустимыми операциями.
Рефакторинг	Изменение внутренней структуры с сохранением внешнего поведения.
Регрессия	Нарушение ранее работавшего поведения после изменения.

6. Исходные данные и учебные примеры

Пример программного модуля предназначен для разбора структуры кода и выполнения проверок, предусмотренных практической работой.

TrainingModule.cs

```
using System;
using System.Collections.Generic;

public static class TrainingModule
{
    public static int Damage(int attack, int defense)
    {
        if (attack < 0 || defense < 0)
            throw new ArgumentOutOfRangeException();
        return Math.Max(0, attack - defense);
    }

    public static bool Remove(Dictionary<string,int> items,
                              string id, int amount)
    {
        if (amount <= 0) throw new ArgumentOutOfRangeException();
        int current;
        if (!items.TryGetValue(id, out current) || current < amount)
            return false;
        if (current == amount) items.Remove(id);
        else items[id] = current - amount;
        return true;
    }

    public static void RunAll()
    {
        if (Damage(12,5) != 7 || Damage(5,12) != 0)
            throw new Exception("damage test failed");
        var items = new Dictionary<string,int>();
        items["key"] = 2;
        if (!Remove(items,"key",1) || items["key"] != 1)
            throw new Exception("remove test failed");
        if (Remove(items,"key",3) || items["key"] != 1)
            throw new Exception("failed removal changed state");
    }
}
```

Файл содержит законченный учебный класс C#. Для проверки вызывается TrainingModule.RunAll(); отсутствие исключения означает прохождение включённых проверок. Дополните набор отрицательными аргументами и операциями с отсутствующим предметом.

7. Использованные источники

1. Подбельский В. В. Программирование. Базовый курс C# : учебник для вузов. Москва : Юрайт, 2024. 369 с. ISBN 978-5-534-10616-9. https://urait.ru/bcode/536775.

2. Кувшинов Д. Р. Основы программирования : учебное пособие для вузов. Москва : Юрайт, 2024. 104 с. ISBN 978-5-534-07559-5. https://urait.ru/bcode/540199.

3. Chacon S., Straub B. Pro Git. Second edition. Apress, 2014. https://git-scm.com/book/en/v2.

4. Unity Technologies. Unity User Manual 2022.3 (LTS). https://docs.unity3d.com/2022.3/Documentation/Manual/UnityManual.html.

5. Unity Technologies. Unity User Manual 2022.3. Creating and Using Scripts. URL: https://docs.unity3d.com/2022.3/Documentation/Manual/CreatingAndUsingScripts.html