Доклад к презентации: "Алгоритмическая модель сражений гладиаторов"

**Слайд 1: Титульный слайд**

Добрый день, уважаемые слушатели!

Сегодня я представлю вашему вниманию программный продукт, реализующий симуляцию боя между различными типами гладиаторов, разработанный в рамках лабораторной работы.

Эта программа имитирует пошаговые бои между гладиаторами с различными характеристиками, визуализирует процесс сражения и демонстрирует работу объектно-ориентированного подхода к разработке программного обеспечения.

Слайд 2: Демонстрация работы

(Если будете показывать программу вживую, можно сказать:)

Сейчас я продемонстрирую работу программы. Мы увидим, как два гладиатора — Берсерк и Стандартный — сражаются на арене, управляя своей выносливостью и здоровьем.

Программа запускает симуляцию боя, где гладиаторы по очереди атакуют, восстанавливают силы и пытаются одержать победу. Визуальная часть включает анимации движения, атак и полоски статусов.

**Слайд 3: Использованные технологии**

Для разработки были выбраны следующие технологии:

Java как основной язык программирования.

* Java Swing — для создания графического интерфейса.
* Java Sampled Sound — для добавления звуковых эффектов в боях.

Эти инструменты обеспечивают кроссплатформенность, гибкость и достаточную производительность для данной задачи.

Слайд 4: Преимущества и недостатки

Почему Java Swing?

Преимущества:

* Кроссплатформенность — работает на Windows, Linux, macOS.
* Богатый набор компонентов для GUI.
* Поддержка MVC, что упрощает структуру кода.

Недостатки:

* Низкая производительность при сложной графике.
* Устаревший внешний вид по умолчанию.

Однако для нашей задачи Swing оказался оптимальным выбором.

Слайд 5: Иерархия классов (UML)

Программа построена на ООП-принципах. Вот ключевые компоненты:

1. Сущности (Entities):

Gladiator (абстрактный класс), Berserk, StandardGladiator.

Weapon, Armor и их реализации.

2. Графический интерфейс (GUI):

MainWindow, Simulator — управление отображением.

MusicPlayer — звуковые эффекты.

Слайд 6-7: Директория Entities

Рассмотрим ключевые абстрактные классы:

1. Armor:

Определяет защиту (например, HeavyArmor поглощает 30% урона).

Метод absorbDamage() рассчитывает полученный урон.

2. Weapon:

Содержит параметры: базовый урон, затраты выносливости.

Метод getDamage() возвращает итоговый урон с учетом случайного множителя.

Слайд 8-9: Класс Gladiator

Это основа логики боя:

Характеристики: здоровье, выносливость, шанс уклонения.

Методы:

attack() — наносит урон, тратит стамину.

takeDamage() — применяет урон с учетом брони.

recoverStamina() — восстанавливает силы при отдыхе.

Например, Берсерк атакует сильнее, но быстрее устает.

Слайд 10: Симуляция боя

Как работает бой?

1. Очередность ходов определяется случайно.

2. Если у гладиатора заканчивается выносливость, он отдыхает.

3. При атаке проверяется:

Попадание (может быть промах).

Критический урон (если повезет).

4. Бой заканчивается при смерти одного из бойцов.

Слайд 11: Визуализация

Для создания эффекта полного погружения в атмосферу гладиаторских боёв в программе реализован комплекс визуальных и звуковых эффектов. В начале схватки на экране появляется динамичная надпись "Fight!", которая плавно исчезает, сопровождаемая соответствующим звуковым эффектом, что задаёт напряжённый тон предстоящему сражению. Бойцы начинают поединок по краям арены и плавно сходятся к центру, имитируя выход на поле. Над каждым гладиатором в реальном времени отображаются индикаторы здоровья и выносливости. Особое внимание уделено звуковому сопровождению - каждый тип атаки имеет уникальный звуковой эффект: агрессивный рёв берсерка контрастирует с более сдержанными звуками атак стандартного гладиатора, а торжественный аккорд возвещает о победе одного из участников.

Слайд 12: Заключение

Эта программа демонстрирует:

1. Применение ООП для моделирования сложных систем.

2. Возможности Java Swing для создания интерактивных приложений.

3. Интересный игровой процесс с элементами стратегии.

Спасибо за внимание! Готов ответить на ваши вопросы.

**Что такое MVC (Model-View-Controller)?**

**MVC** — это архитектурный паттерн (шаблон проектирования), который разделяет приложение на три основных компонента:

1. **Model (Модель)** — отвечает за данные и бизнес-логику.
2. **View (Представление)** — отображает данные пользователю (интерфейс).
3. **Controller (Контроллер)** — обрабатывает пользовательский ввод и управляет взаимодействием между Model и View.

**1. Model (Модель)**

* **Что делает:**
  + Хранит данные (например, здоровье гладиатора, урон оружия).
  + Содержит логику их обработки (например, расчет урона в Gladiator.attack()).
* **Пример из вашего проекта:**
  + Классы Gladiator, Weapon, Armor — это часть модели.
  + Метод takeDamage() в Gladiator обновляет здоровье, учитывая броню.

**2. View (Представление)**

* **Что делает:**
  + Отображает данные модели (например, полоски здоровья в Simulator).
  + Не содержит логики, только визуализацию.
* **Пример из вашего проекта:**
  + Класс Simulator (наследник JPanel) рисует гладиаторов, фон, анимации.
  + Метод paintComponent() отвечает за отрисовку.

**3. Controller (Контроллер)**

* **Что делает:**
  + Принимает ввод пользователя (например, клики, нажатия клавиш).
  + Обновляет модель и триггерит изменения во View.
* **Пример из вашего проекта:**
  + Класс MainWindow может быть контроллером, если обрабатывает кнопки (например, "Начать бой").
  + В текущей реализации контроллер "размазан" между Simulator и таймерами анимаций.

**1. Как организовано взаимодействие объектов?**

Взаимодействие объектов построено на принципах **инкапсуляции** и **делегирования**:

* Класс Simulator управляет процессом боя, координируя действия гладиаторов (Gladiator).
* Гладиаторы (Berserk, StandardGladiator) взаимодействуют через методы:
  + attack() — вызывает weapon.getDamage() для расчета урона.
  + takeDamage() — учитывает броню (armor.absorbDamage()).
* Оружие (Weapon) и броня (Armor) — отдельные классы, которые делегируют логику урона/защиты своим методам.

**Пример:**

java

// Взаимодействие при атаке:

double damage = gladiator1.attack(); // Gladiator вызывает weapon.getDamage()

gladiator2.takeDamage(damage); // Учитывает armor.absorbDamage()

**2. В каком месте (кто ответственен) рассчитывается урон?**

Урон рассчитывается **в два этапа**:

1. **Оружие (Weapon):**
   * Метод getDamage() возвращает базовый урон + случайный множитель.
   * Например, Sword может нанести 15–25 урона.
2. **Гладиатор (Gladiator):**
   * Метод attack() вызывает weapon.getDamage() и тратит выносливость (stamina -= weapon.staminaCost).

**Ответственный:**

* Расчет базового урона — оружие (Weapon).
* Итоговый урон (с учетом критического удара/промаха) — гладиатор (Gladiator).

**3. Почему методы не перенесены в самих героев?**

Методы вынесены в отдельные классы (Weapon, Armor) для:

* **Соблюдения SRP (Single Responsibility Principle):**
  + Класс Gladiator отвечает за общую логику боя.
  + Класс Weapon — только за урон, Armor — только за защиту.
* **Гибкости:**
  + Можно легко добавить новые типы оружия/брони без изменения кода гладиатора.
* **Масштабируемости:**
  + Например, если добавить магию, создается класс Spell, а не модифицируется Gladiator.

**Пример плохого подхода:**

java

// Если бы всё было в Gladiator:

class Gladiator {

double getSwordDamage() { ... } // Нарушает SRP!

double getAxeDamage() { ... } // Придется добавлять методы для каждого оружия.

}

**4. Содержит ли иерархию класс Гладиатора/Оружия/Брони?**

Да, используется **иерархия наследования** и **абстракция**:

1. **Гладиаторы:**
   * Абстрактный класс Gladiator.
   * Конкретные реализации: Berserk, StandardGladiator.
2. **Оружие:**
   * Абстрактный класс Weapon.
   * Конкретные типы: Sword, HeavyMace (с разным уроном/стоимостью выносливости).
3. **Броня:**
   * Абстрактный класс Armor.
   * Подклассы: LightArmor, HeavyArmor (разный процент поглощения урона).

**Диаграмма упрощенно:**

Gladiator

├─ Berserk

└─ StandardGladiator

Weapon

├─ Sword

└─ HeavyMace

Armor

├─ LightArmor

└─ HeavyArmor

**5. Язык программирования, библиотеки, используемые для создания игры?**

* **Язык:** Java (версия 8+).
* **Библиотеки:**
  + **Java Swing** — для графического интерфейса (окно боя, анимации).
  + **Java Sampled Sound** — для воспроизведения звуков (удары, победа).
* **Дополнительно:**
  + Стандартные библиотеки Java (java.util, java.awt).

**Почему не JavaFX?**  
Swing был выбран из-за простоты для небольших проектов и отсутствия необходимости в сложной графике.