

Архитектурная документация

1. Общие сведения о системе

Данный документ содержит описание архитектуры компьютерной игры в жанре Roguelike. Основные цели игры – принести удовольствие игрокам, позволить проникнуться атмосферой старых консольных игр.

Разрабатываемая игра является однопользовательской и десктопной. На текущей итерации предполагается только один случай использования. После входа в приложение пользователь вводит свое имя и создает героя, после чего переходит непосредственно к прохождению уровней.

Игра работает все время локально на компьютере пользователя и не взаимодействует ни с какими внешними системами.

2. Architectural drivers

Существенное влияние на архитектуру оказало требование быть разработанной в течение одной недели. В связи с этим она не претендует на максимальную полноту.

Одним из нефункциональных требований является отсутствие ощутимых для пользователя задержек. С другой стороны, должна быть возможность быстрой разработки, чтобы представлять новую версию каждую неделю. Поэтому в качестве языка разработки был выбран язык Java, так как он сочетает в себе как хорошую производительность, так и удобство вместе с высокой скоростью разработки.

Для отрисовки графики планируется использовать библиотеку Lanterna, так как она является наиболее современной и присутствует в Maven Central. В качестве альтернатив рассматривались такие библиотеки как Charva и Java Curses Library.

Архитектура игры разрабатывалась из соображений, что она должна быть расширяема. Была предусмотрена возможность простой модификации системы при получении новых требований, причем для некоторых аспектов игры (например, внутриигровые объекты) модификация должна быть возможна без внесения изменений в исходный код программы.

3. Роли и случаи использования

Предполагается существование только одного актора: игрока. Игрок - любитель олдскульных игр, который решил поиграть в данную игру. Он скачивает её себе на компьютер и играет локально.

3.1 Прецедент П1. Игра в Roguelike игру

Основной исполнитель. Игрок.

Заинтересованные лица и их требования.

- Игрок. Хочет получить удовольствие от игры и проникнуться духом старых игр.

Основной успешный сценарий.

1. Игрок запускает игру.
2. Система генерирует новый уровень для пользователь.
3. Пользователь убивает всех мобов на уровне.
4. Пользователь переходит на новый уровень

Действия, описанные в пп. 2-3, повторяются, пока пользователь не пройдет все уровни.

5. Система поздравляет пользователя с прохождением игры.

Расширения.

*а. При закрытии пользователем терминала с игрой.

1. Приложение завершается, не делая при этом никаких дополнительных действий

4. Диаграмма компонентов

Система состоит из 6 основных компонентов. Компонент EventProducers ответственен за генерацию событий, на которые система должна реагировать. В частности, компонент может слушать события нажатия на клавиатуру и создавать соответствующее событие, либо генерировать события с заданной периодичностью. Компонент Core ответственен за обработку системного события, обновление состояния конечного автомата системы и вызов перерисовки графического интерфейса. UI занимается отрисовка текущего состояния системы, инкапсулируя выбранную UI-библиотеку. Компонент Models представляет из себя набор классов, который используются другими компонентами системы и представляют сущности предметной области. Lib представляет из себя набор алгоритмов, с помощью которых Core реализует обновление состояния системы. Так как Lib конфигурируется отдельно, мы решили вынести его в отдельный от Core модуль. Repositories представляет из себя набор репозитория, содержащие конфигурируемые параметры системы, как то: набор предопределенных уровней, типы существующих предметов и их параметры и так далее.

Диаграмма компонентов представлена на рисунке 1.

5. Диаграмма классов

Компоненты пакета EventProducers отвечают за генерацию событий в системе. Класс UserActionEventProducer при пользовательском событии (например, нажатие на клавишу) генерирует событие и отправляет его GameController.

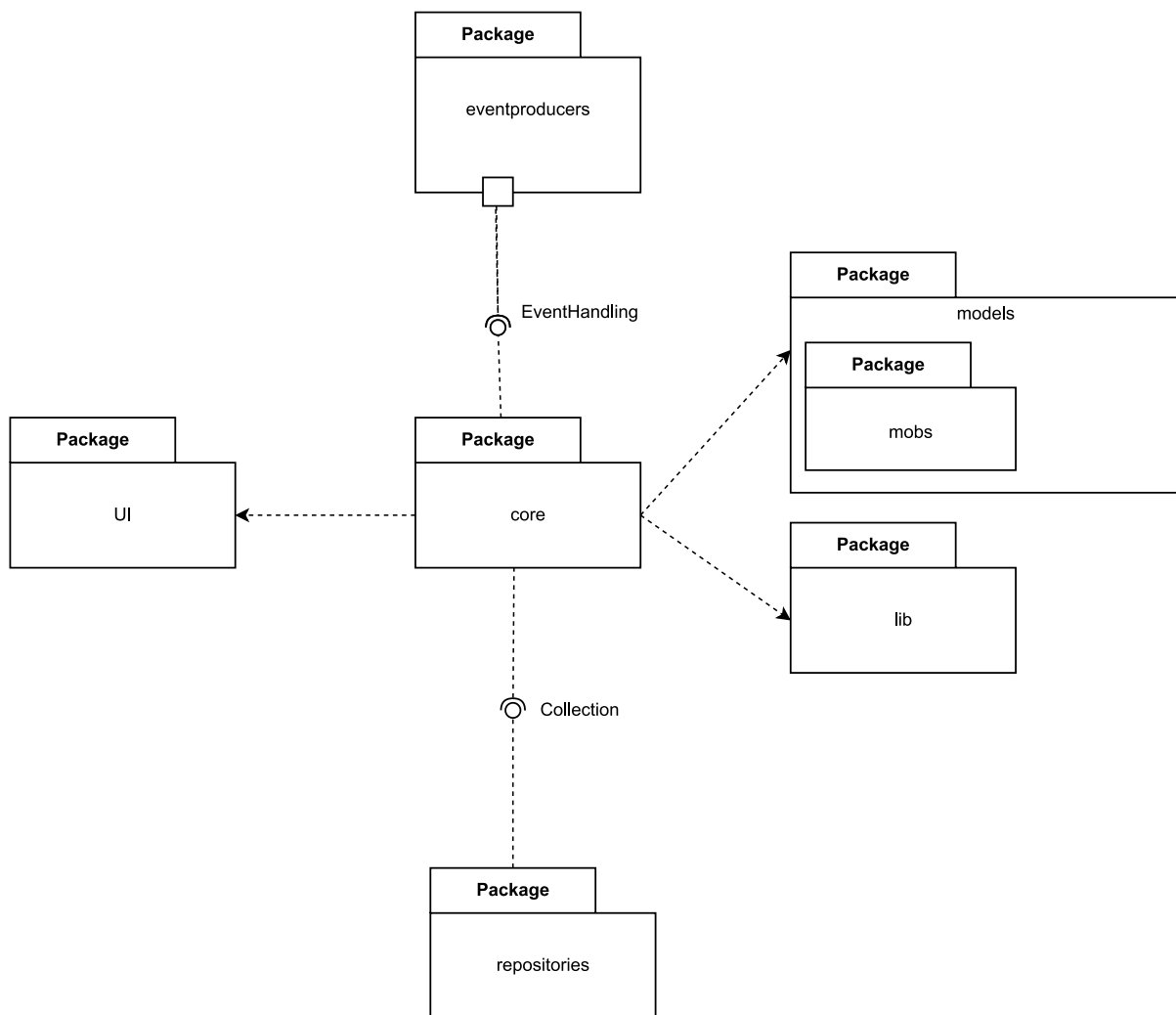


Рис. 1. Диаграмма пакетов приложения

Классы пакета Models используются для хранения информации об игровых объектах. Игровой объект может быть либо вещью (Item), либо мобом (игрок либо программно управляемый враг), либо переходом между комнатами на одном уровне. Каждый игровой объект обладает названием и координатами (относительно уровня). Также каждый игровой объект обладает методом stepOn(Mob), который определяет, что делать с мобом, наступившим на этот игровой объект.

Поведение врагов реализовано с помощью паттерна стратегия. Есть интерфейс MobBehavior с методом makeNextMove(Mob, Mob, Room) и три конкретных типа поведения: AggressiveMobBehavior, CowardMobBehavior и PassiveMobBehavior.

Мобы могут быть оглушены. Эта механика реализована с помощью класса ConfusedMob, декорирующего класс Mob.

Класс Level содержит комнаты Rooms, которые в свою очередь содержат объекты типа GameObject.

Пакет Core содержит классы, содержащие игровую логику, обработку событий и изменение состояния игры. Класс GameController на основе полученного события,

используя GameStateChanger, обновляет GameState и, используя классы пакета UI, обновляет пользовательский интерфейс. GameState агрегирует состояние текущей игры, в том числе текущий уровень.

В lib содержатся реализации алгоритмов, необходимых для работы пакета Core и вспомогательные утилиты, например, генератор уровней, генератор карты, класс Logger, ответственный за логгирование, класс Event.

Пакет Repositories содержит ряд классов-репозитория, содержащих данные, который необходимы в остальных пакетах. Репозиторий LevelRepository содержит предопределенные уровни. Configuration содержит конфигурируемые параметры игры.

Пакет UI отвечает за отображение графического интерфейса. Класс Renderer отвечает за отображение состояния уровня.

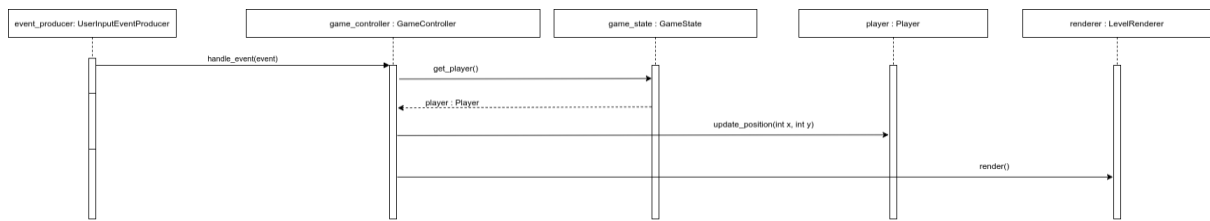
Инициализацией игры будет заниматься GameInitializer (пакет Core). Он будет инициализировать GameController и все остальные необходимые объекты.

За генерацию мобов будет отвечать фабрика MobGeneratorFactory. У неё есть две конкретных реализации: FantasyMobGeneratorFactory - генерирует фэнтези мобов, SciFiMobGeneratorFactory - генерирует научно-фантастических мобов. Фабрики умеют генерировать 2 типа мобов: StrongMob и WeakMob.

Реализующий паттерн “строитель” класс MapGenerator ответственен за генерацию карт. Строителю можно указать размер карты, тип генерации (из заранее заданного набора карт или с помощью рандомизированного алгоритма генерации), количество уровней на карте, фабрику для генерации мобов.

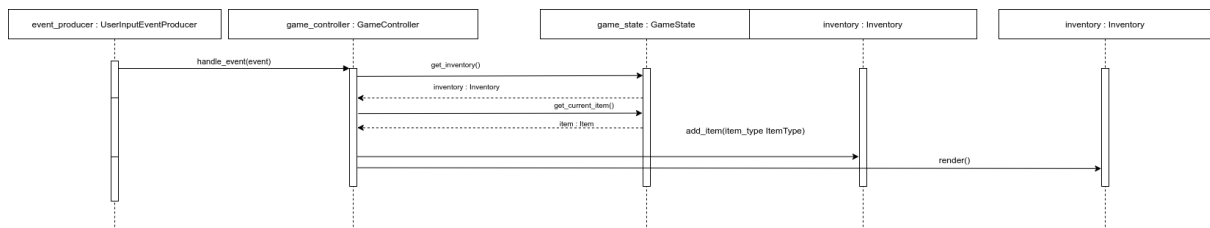
Поддержана возможность репликации для мобов с помощью паттерна “прототип” и интерфейса ClonableMob.

Диаграмма классов представлена на рисунке 2.



Прецедент “Пользователь подбирает вещь”

Экземпляр класса *UserInputEventProducer* при событии, которое инициировано пользователем, генерирует системное событие event класса *Event*. Экземпляр класса *GameController* обрабатывает событие следующим образом: получив экземпляр *inventory* класса *Inventory* и экземпляр *item* класса *Item*, он вызывает публичный метод *add_item* у *inventory*, передав туда *item*. Затем экземпляр *GameController* вызывает публичный метод *render()* класса *Renderer* чтобы отобразить изменение состояния игры визуально.



6.2 Диаграмма состояний

Изначально игрок находится в состоянии "Walking", в рамках которого он может передвигаться по игровому пространству.

Когда игрок доходит на тайла, на котором расположена вещь, он ее подбирает (событие "took an item") и остается в состоянии "Walking".

Когда игрок встречается моба и переходит в состояние "In fight" (событие "Met a mob"), он либо побеждает его в бою и возвращается в состояние "Walking", либо проигрывает и переходит в финальное состояние ("Game over").

Когда на текущем уровне не остается ни одного моба, игрок переходит на следующий уровень.

Если уровень является финальным, игрок переходит в финальное состояние (игра заканчивается).

