МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах
(полное название института)

кафедра «Информационные системы»
(полное название кафедры)

Пояснительная записка

к курсовой работе

на тему Разработка игрового фреймворка для платформы Minecraft Spigot Server с использование объектно-ориентированного подхода.

по дисциплине **Объектно-ориентированное программирование**

Выполнил: студент II курса, группы: **ИС/б-22-о**

Направления подготовки (специальности) 09.03.02

Информационные системы и технологии
(код и наименование направления подготовки (специальности))

профиль (специализация)

Клышко Никиты Александровича
(**фамилия, имя, отчество студента**)

Дата допуска к защите « ____ » _____ 20 19 г.

Руководитель _____ _____
(подпись) (инициалы, фамилия)

20 19 г.

Аннотация

В данной пояснительной записке содержится проектная документация игрового фреймворка для платформы Minecraft Spigot Server, разработанного в рамках курсового проекта по дисциплине «Объектно-ориентированное программирование».

Во введении обозначена тема курсового проекта, проблема, которая должна быть решена в результате разработки, определены цели и задачи.

В основной части документа дается описательная постановка задачи, также представлено проектное решение проблемы, в том числе, диаграмма объектов, диаграмма классов, диаграммы состояний экземпляров, диаграмма потоков данных. После описания проекта, дается информация о деталях реализации, а также обоснование выбранных технологий и средств разработки.

В заключении делается вывод о результатах проделанной работы и дается оценка разработанного проекта.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

API – application programming interface, программный интерфейс приложения, , интерфейс прикладного программирования

CI – Continuous Integration (рус. Непрерывная интеграция)

DDD – Domain-driven design (рус. Предметно-ориентированное проектирование)

JVM – Java Virtual Machine (рус. виртуальная машина Java)

IDE – Integrated development environment (рус. Интегри́рованная среда́ разрабо́тки)

SoC – separation of concerns (рус. Разделение ответственностей)

ПО – программное обеспечение

ЯП – язык программирования

ВВЕДЕНИЕ

В 2009 году была представлена первая версия компьютерной игры Minecraft – инди-игры в жанре песочницы, разработанной шведским программистом Маркусом Перссоном и выпущенной его компанией Mojang AB. Хотя Minecraft была задумана как развлекательная компьютерная игра, по мере роста её популярности многократно обсуждалась и возможность неигрового применения, в частности, в сферах автоматизированного проектирования и образования [2]: в процессе игры дети получают навыки программирования, инженерного дела, архитектуры и математики [4].

Многопользовательский режим игры Minecraft построен на основе клиент-серверной архитектуры. Для начала игры в этом режиме пользователь должен подключиться к удаленному серверу, на котором запущено программное обеспечение (далее – ПО) серверной части игры. Существует множество реализаций игрового сервера: Classic Minecraft Server, Spigot Server, cuberite и др.

Самой популярной реализацией является Spigot Server, но имеет при этом ряд недостатков, главный из которых состоит в том, что реализация основана на классической версии сервера, которая, в свою очередь, является однопоточной. Впрочем, альтернативные реализации либо также являются однопоточными, либо реализуют не всю логику игры, поэтому их почти невозможно использовать. Таким образом, очень сложно достичь высокой производительности сервера для поддержки тысяч одновременно подключенных клиентов, даже используя очень производительное аппаратное обеспечение.

Чтобы решить эту проблему, в 2011 году был создан проект BungeeCord – прокси сервер уровня L7 для Minecraft, позволяющий перемещать игроков между серверами в сети без необходимости выхода в главное меню игры.

Перечисленное выше ПО позволяет построить сеть серверов игры, размещенную на множестве физических серверов, что обеспечивает почти неограниченное горизонтальное масштабирование. На рисунке 1 изображена типичная архитектура коммуникационной части игровой сети. По данной схеме

видно, что каждый из прокси серверов обрабатывает только часть клиентов, но каждый из клиентов имеет возможность подключиться к любому из игровых серверов.

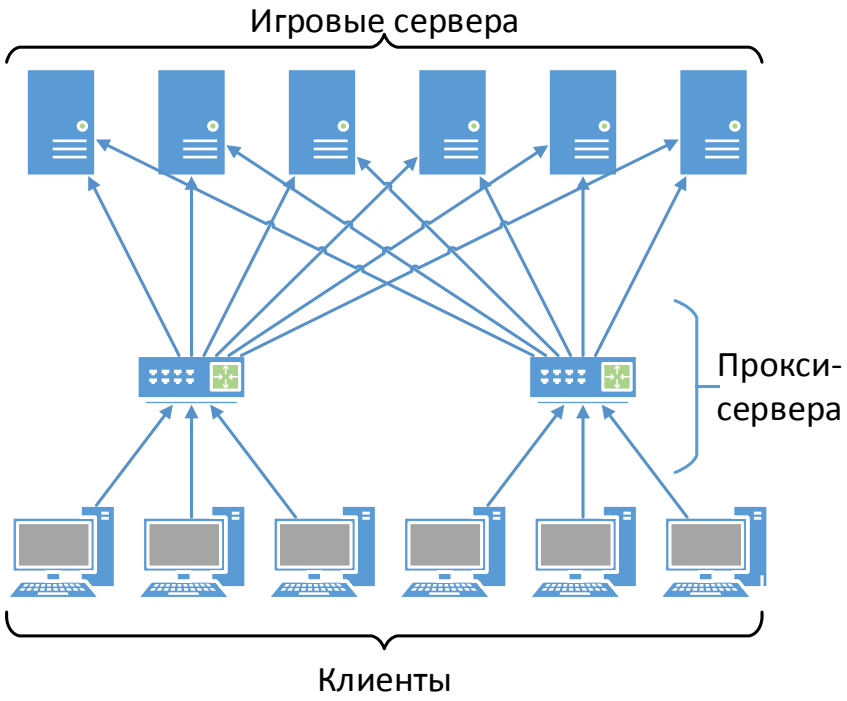

Рисунок 1 – Архитектура коммуникационной части игровой сети

Игровые серверы предоставляют очень гибкий программный интерфейс (далее – API), который позволяет создавать собственные игры внутри Minecraft’a.

Но если возникает необходимость реализовать игру, в которой участвует небольшое количество игроков (2 – 8), то запускать для каждой игры отдельный игровой сервер становится не выгодно. Так как игровой сервер по умолчанию (без подключенных игроков) потребляет определенное количество ресурсов, то при небольшом количестве игроков на одном сервере, удельное потребление ресурсов на одного игрока значительно больше, чем при оптимальном количестве одновременно подключенных клиентов (20-30).

Целью данного курсового проекта является разработка фреймворка для платформы Minecraft, позволяющего создавать игры на платформе Minecraft и запускать несколько параллельных игр на одном игровом сервере. Создание такой системы позволит решить рассмотренные выше проблемы.

Техническое задание на реализацию проекта утверждено Севастопольским государственным университетом.

Для достижения поставленной цели были определены следующие задачи:

1. Разработать протокол высокого уровня для взаимодействия с системой игровых очередей
2. Спроектировать гибкий API фреймворка
3. Реализовать сервисы фреймворка, обеспечивающие необходимый базовый функционал

Фреймворк может быть использован в игровых сетях Minecraft как для оптимизации серверной части существующих игр, так и для разработки новых. Также, реализация фреймворка может быть расширена на другие игровые платформы, что позволит переносить ранее созданные игры в новое окружение.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Цель разработки

Цели разработки системы:

1. Создание игровой платформы на базе Minecraft, позволяющей быстро разрабатывать игры.
2. Запуск множества параллельных игр на одном игровом сервере.
3. Унификация интерфейса взаимодействия игровой платформы с очередью игр (балансировщиком, матчмейкером) и другими системами.

1.2 Описательная постановка задачи

Система представляет из себя плагин (расширение) для сервера Minecraft, выполняющее следующие функции:

1. Обработка запросов на игровые матчи от очереди игр.
2. Автоматическое скачивание из центрального репозитория и загрузка в память игрового мира и игровой реализации. А также загрузка данных об игроках перед началом игры.
3. Управление жизненным циклом игры и контроль переходов состояний объектов игровой реализации.
4. Распределение исходного потока событий платформы на множество запущенных игр.

Загрузка игрового мира должно происходить с учетом максимального времени игрового цикла, при котором не нарушается плавность игры. То есть, мир должен загружать по кускам (чанкам) для обеспечения распределения нагрузки по времени.

Все взаимодействия с блокирующими вызовами ввода/вывода, в том числе, коммуникации с другими системами и хранилищами данных, должны

быть организованы в асинхронном (неблокирующем для основного потока) режиме.

1.3 Ограничения, условия выполнения, функционирования

Текущая реализация проекта ориентирована на конкретную платформу Minecraft Spigot Server, которая в свою очередь, написана на Java, следовательно, требует для работы Java Runtime Environment.

Остальные ограничения являются опциональными, так как для взаимодействия с внешними системами выделены интерфейсы, следовательно, внутренние системы фреймворка не зависят от деталей реализации. Так, например, для взаимодействия с базой данных существует интерфейс GameRepository, для взаимодействия с системой очередей – интерфейс GameFactory.

2 ПРОЕКТНОЕ РЕШЕНИЕ

2.1 Абстрагирование и выделение объектов

Объект manager класса GameManager агрегирует объекты game, который также агрегирует объекты игроков player1, player2, player3.

Создание объектов происходит по порядку: сначала объект manager получает запрос на проведение игры и создает экземпляр игры, после чего, имея список идентификаторов игроков, дает команду игре создать объекты игроков, а также объекты доступных в данном режиме классов.

Каждый игрок ассоциируется с определенном классом.

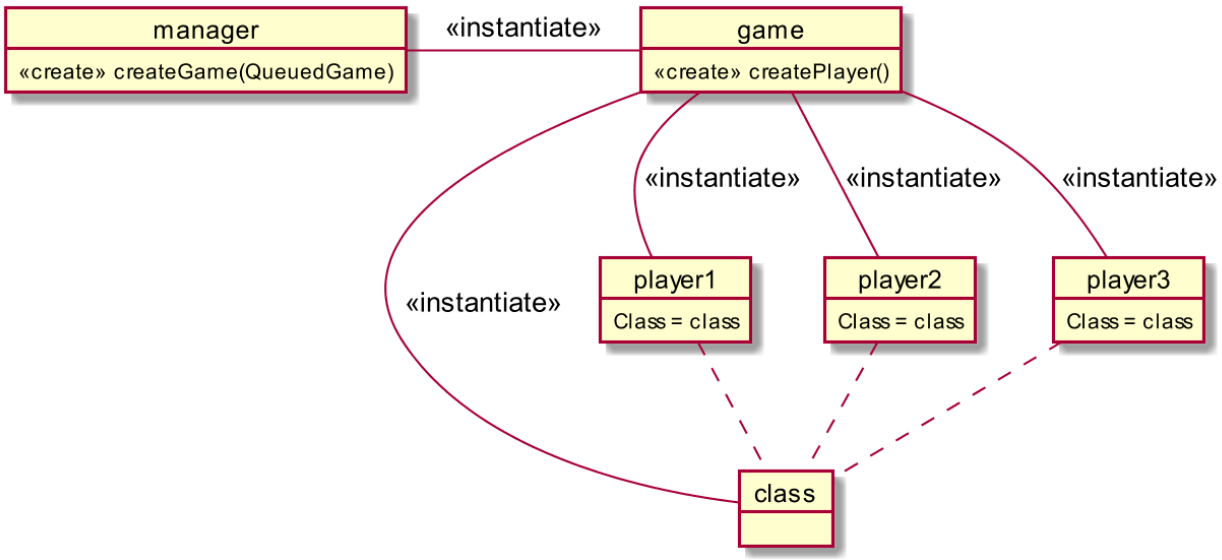

Рисунок 2.1 – Диаграмма объектов

Множество объектов, изображенное на рисунке 2.1, представляет простейший случай работы системы.

2.2 Построение иерархии классов

Исходя из выделенных объектов, была построена иерархия классов.

Сущность игры выделена в абстрактный класс Game. Так как игры могут быть одиночными или командными, то Game имеет два дочерних класса SoloGame и TeamGame.

Для моделирования командной игры был выделен класс Team, а также класс TeamPlayer, расширяющий класс игрока GamePlayer.

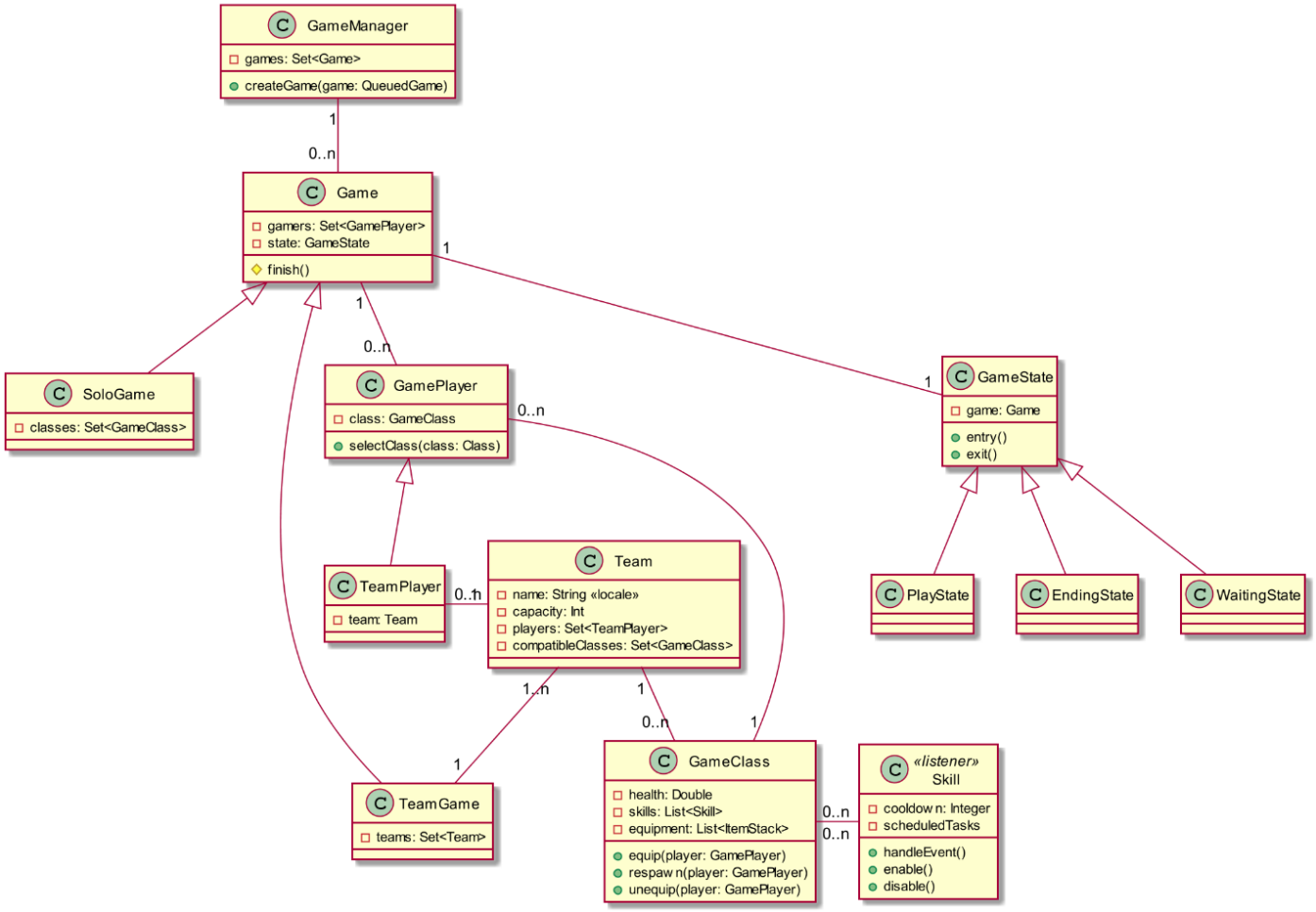

Рисунок 2.2 – Диаграмма классов

Также был выделен класс GameClass, который содержит описание характеристик игрока, имеющего данный игровой класс, и список его способностей (skills).

Кроме этого, нужно отметить, что для удобства доступа к полям и методам унаследованных классов, созданных при разработке новых игр на основе базовой модели, были использованы обобщенные классы. Таким образом, итоговые сигнатуры классов объявлены следующим образом:

```
abstract class Game<P extends GamePlayer>
abstract class Team<P extends TeamPlayer>
abstract class TeamGame<P extends TeamPlayer, T extends Team> extends Game<P>
abstract class GamePlayer<G extends Game>
abstract class TeamPlayer<G extends TeamGame, T extends Team> extends GamePlayer<G> {
```

Использование обобщений позволяет получать экземпляры производных классов частных реализаций из методов основного агрегатного класса (Game) без необходимости приведения типов.

2.5 Диаграммы переходов состояний

2.5.1 Объекты Game

Объекты класса Game могут находиться в одном из 7 состояний. На рисунке 2.3 изображена диаграмма состояний экземпляра Game.

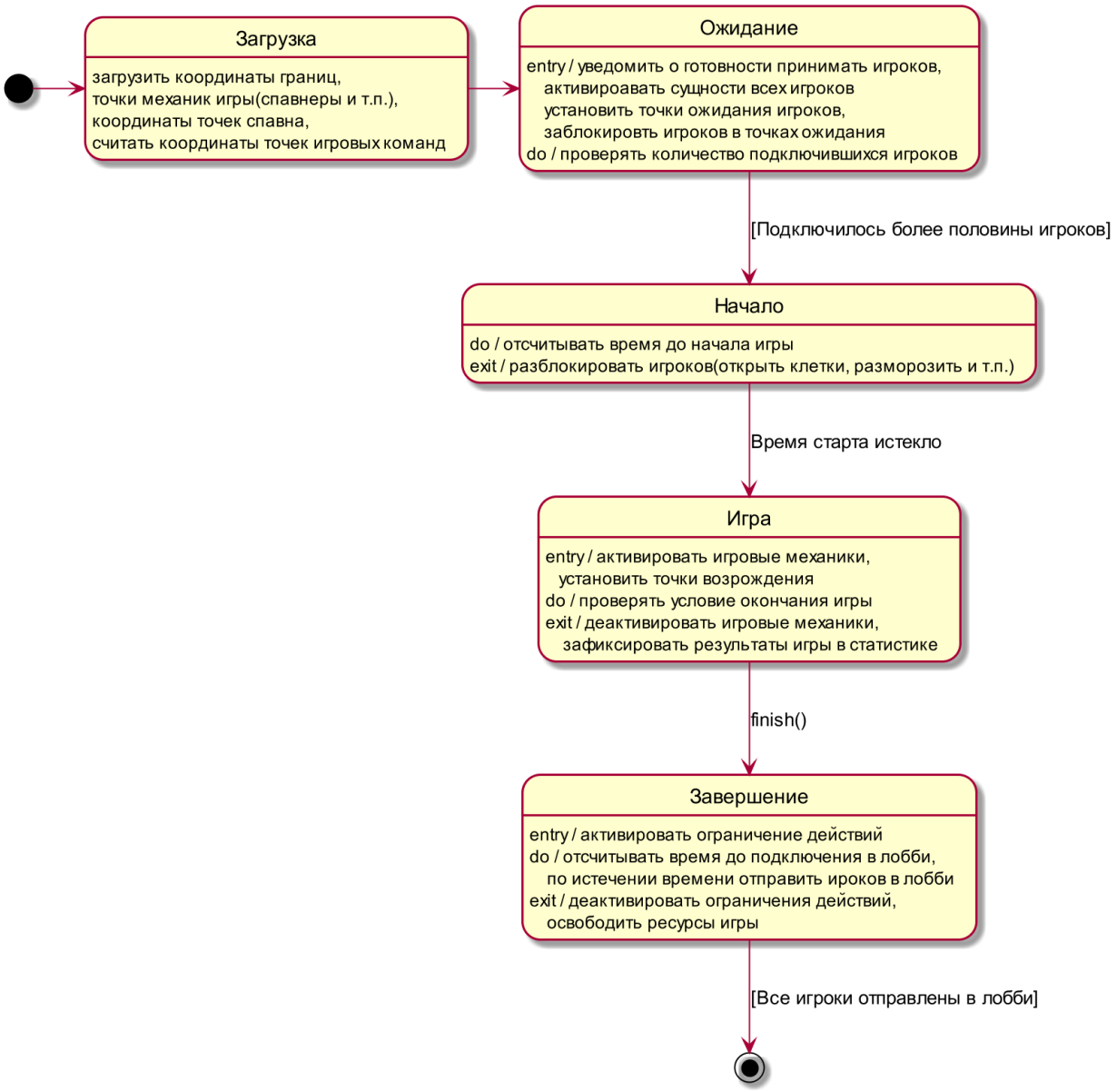

Рисунок 2.3 – Диаграмма состояний объекта игры класса Game

После создания объекта игры, он переходит из начального состояния в состояние загрузки. В этом состоянии синхронно и асинхронно (в зависимости

от требований базовой платформы) загружаются основные данные и ресурсы, необходимые для запуска игры. Например, загружается мир (синхронно, покусочно (по-чанково), с контролем затраченного времени в течение одного игрового цикла) и его параметры (точки спавна игроков, игровых ресурсов, границы игровой зоны и т.п.), загружаются данные всех игроков (асинхронно, из хранилища данных) и т.д. Переход в следующие состояние (состояние ожидания) является не триггерным, то есть произойдет по окончании всех процессов состояния загрузки.

По переходу в состояние ожидания, игра уведомляет внешнюю систему о готовности обрабатывать подключения игроков, а также предварительно инстанцирует (спавнит) сущности игроков в игровом мире. Переход в состояние начала игры произойдет по сторожевому условию: подключилось более половины игроков. По событию истечения времени стартовой фазы, объект перейдет в состояние активной игры.

Состояние активной игры является основным. После перехода в это состояние, активируются все игровые механики и процессы. В течение всего периода нахождения в этом состоянии, происходит проверка условия окончания игры (победы какого-либо участника), срабатывание которого вызывает триггерный переход в состояние завершения игры.

В состоянии завершения игры, все игровые механики и процессы останавливаются и действия игроков ограничиваются. Также происходит вызов функции объявления результатов игры. По истечении времени завершающей фазы, внешняя система будет оповещена о необходимости переместить игроков из игрового мира в лобби игровой сети. Когда это произойдет, то выполнится сторожевое условие и объект перейдет в конечное состояние.

2.5.2 Объекты GamePlayer

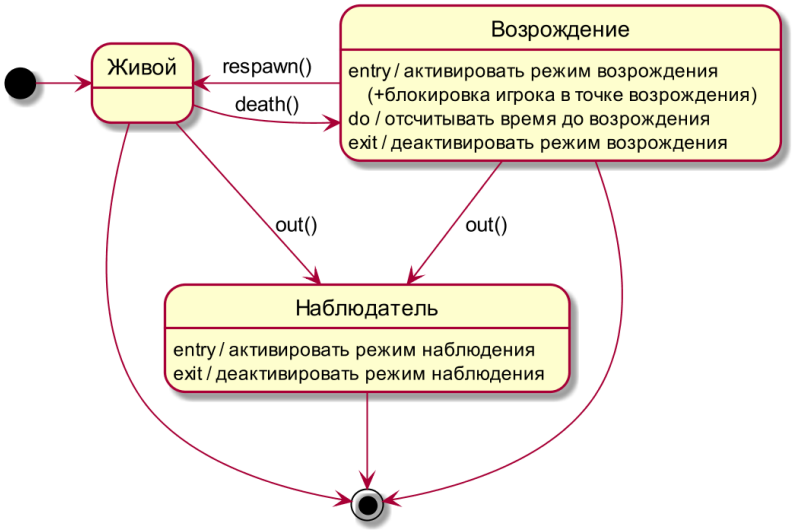

Рисунок 2.4 – Диаграмма состояний объекта игрок класса GamePlayer

Объекты класса GamePlayer переходят из начального состояния в активное при переходе объекта игры в состояние активной игры, и переходят в конечное состояние по выходу объекта игры из активной фазы. Таким образом, состояния экземпляров GamePlayer имеют смысл только во время активной игры.

Игрок может находится в одном из 5 состояний, каждое из которых определяет набор действий, которые он может совершать.

Основное состояние – живой, то есть активный игрок. Из этого состояния, в зависимости от логики игры, он может переходить в состояние возрождения и обратно или же перейти в состояние наблюдателя. Переход игрока в состояние наблюдателя означает, что он выбыл из игры и может только наблюдать за дальнейшим ходом игры.

Все переходы между активными состояниями игрока являются триггерными.

2.6 Диаграмма потоков данных

Одной из целей проекта является обеспечение возможности запуска множества параллельных игр на одном игровом сервере.

Платформа Spigot Server построена на основе событийно-ориентированной модели, которая подразумевает наличие потока событий и множества подписчиков, обрабатывающих эти события.

Рисунок 2.5 – Диаграмма потоков данных

В обычном режиме игровая платформа генерирует единый поток событий для всех взаимодействий, происходящих во всех мирах данного сервера. При условии запуска нескольких параллельных игр, события, относящиеся к одной игре, будут попадать в обработчики событий не только этой игры, но и всех других, запущенных на этом сервере. Поэтому необходимо реализовать “маршрутизатор” событий, который определит принадлежность события к конкретной игре и передаст его обработчикам только этого экземпляра игры. Этой моделью описывается основной поток данных системы. На рисунке 2.5 изображена диаграмма потоков данных.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Обоснование выбора языка программирования

Демонстрационная реализация фреймворка основана на платформе Minecraft Spigot Server, которая разработана на объектно-ориентированном языке Java. Поэтому проект был разработан на этом же языке для упрощения интеграции всех компонентов.

В качестве среды разработки использовалась IDE IntelliJ от компании JetBrains. Данная среда разработки отвечает всем современным стандартам и совместима со всеми популярными системами сборки Java-приложений, включая Maven, Gradle, Ant и др.

Для сборки проекта использовался Gradle, потому что скрипты сборки этой системы описываются на JVM-совместимом языке Groovy, что существенно упрощает непрерывную интеграцию (CI) и тестирование ПО, поставляемого в виде расширений, в частности в виде плагинов для сервера Minecraft.

Для контроля изменений кода и версионирования ПО использовалась система Git. В качестве хостинга удаленного репозитория использовался standalone сервис GitLab.

3.2 Описание взаимодействия программных модулей

Рисунок 2.6 – Диаграмма компонентов

Система разделена на два основных компонента: SubscribeManager и GameManager. Каждый из компонентов взаимодействует с какой либо внешней системой через соответствующий интерфейс. Так, GamManager обрабатывает запросы на проведение игр от игровой очереди и в последующем управляет жизненны циклом объекта игры. SubscribeManager взаимодействует с событийным интерфейсом платформы Spigot Server и осуществляет распределение потока событий по текущим параллельным играм.

3.3 Интерфейс пользователя

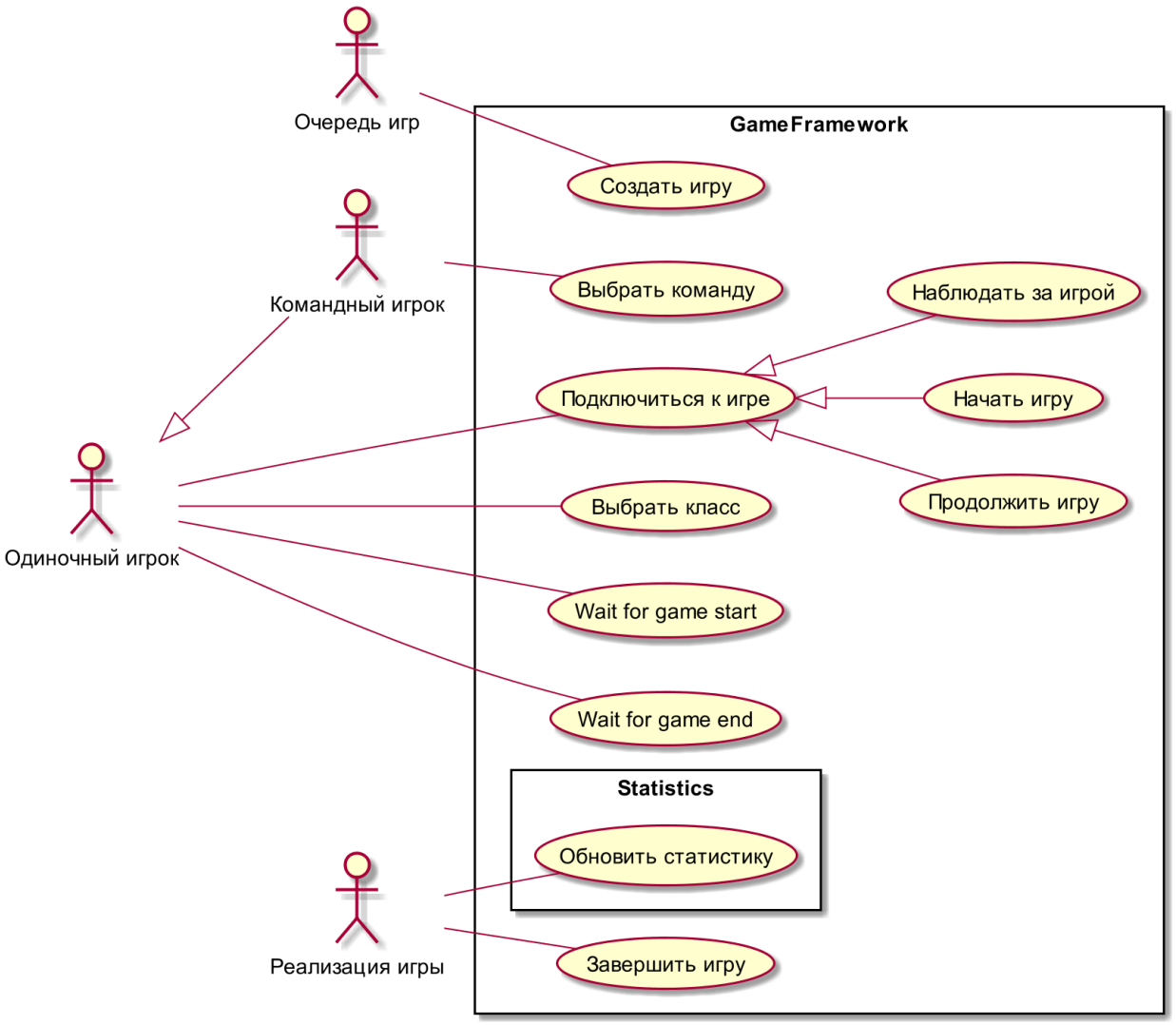

Рисунок 2.7 – Диаграмма вариантов использования системы

Актер «Очередь игр» – внешняя система, отвечающая за формирование запросов на проведение игр (matchmaking), то есть распределение игроков, стоящих в очереди, по игровым матчам. Актер ассоциирован с прецедентом создания игры.

«Реализация игры» – конкретная реализация игры, созданная разработчиком, использующим фреймворк. Процессы подготовки игры к старту и непосредственный запуск игры система производит автоматичиески, но когда необходимо завершить игры (то есть достигнута цель игры) – решает реализация. Поэтому этот актер ассоциирован с прецедентом «Завершить игру», а

также с прецедентом обновления статистики игроков, который также вызывается по окончании игры.

«Одиночный игрок» и его расширение «Командный игрок» представляют непосредственных пользователей системы, игроков. Ассоциированы со множеством прецедентов, например, выбор команды, выбор класса, подключение к игре. При этом есть еще три варианта использования, расширяющие прецедент «Подключиться к игре»: «Наблюдать за игрой», «Начать игру», «Продолжить игру». Это означает, что игрок при подключении может начать или продолжить игру в зависимости от состояния игры или наблюдать за игрой, если он в этом матче не участвует или уже выбыл из игры.

3.4 Критерии качества программной системы

3.4.1 Кросс-платформенность

Кросс-платформенность или межплатформенность — способность программного обеспечения работать с двумя и более аппаратными платформами и (или) операционными системами. Обеспечивается благодаря использованию высокоуровневых языков программирования, сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ. Типичным примером является программное обеспечение, предназначенное для работы в операционных системах Linux и Windows одновременно.

Язык программирования Java и экосистема JVM обеспечивают отличную работа разработанного ПО на различных операционных системах.

3.4.2 Сопровождаемость

Сопровождаемость программного обеспечения — характеристики программного продукта, позволяющие минимизировать усилия по внесению в него изменений:

- для устранения ошибок;
- для модификации в соответствии с изменяющимися потребностями пользователей.

Соблюдение принципов SOLID и Separation of concerns (SoC), а также применение шаблонов проектирования и программирования позволило достичь высокой сопровождаемости ПО.

3.4.3 Совместимость

Совместимостью программ (англ. program compatibility) называется способность программ к взаимодействию друг с другом, возможно, в рамках более крупного программного комплекса.

Абстракция, как один из принципов ООП, обеспечила совместимость с различными серверными платформами, для которых необходимо только предоставить реализацию общего программного интерфейса.

ВЫВОДЫ

Для достижения поставленной цели были изучены различные принципы объектно-ориентированного анализа и проектирования. В процессе разработки активно применялись базовые принципы ООП: абстракция, полиморфизм, инкапсуляция, а также принципы SOLID и подходы предметно-ориентированного проектирования (DDD). Для достижения гибкости API использовались следующие возможности объектно-ориентированного языка Java: интерфейсы, обобщенные классы.

Для оптимизации время-затрат на компиляцию и сборку проекта была изучена система сборки Java-приложений Gradle. Для упрощения интеграции и тестирования был также изучен JVM-язык Groovy и разработаны дополнительный скрипты для Gradle-проекта.

Контроль изменений кода осуществлялся системой контроля версий Git, а в качестве хостинга использовался собственный экземпляр сервиса GitLab. Для применения этих инструментов была изучена система Git, а также способы хостинга удаленных репозиториев этой системы.

В заключении можно отметить, что все поставленные задачи были выполнены. Платформа GameFramework позволяет быстро и удобно создавать игры, а также запускать их на сервере Spigot в параллельном режиме. Система корректно взаимодействует со всеми внешними интерфейсами и выполняет все необходимые функции. То есть цель курсового проекта успешно достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Arlow, J. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design [Текст] / J. Arlow, I. Neustadt. – 2-е изд. – Boston: Addison–Wesley Professional, 2005. – 624 с.

2. Blockbusting Minecraft gamers event Minecon recognised with largest single videogame convention record [Электронный ресурс]. – Режим доступа: http://www.guinnessworldrecords.com/news/2015/7/blockbusting-minecraft-gamers-event-minecon-recognised-with-largest-single-videog. - Blockbusting Minecraft gamers event Minecon recognised with largest single videogame convention record | Guinness World Records. – (Дата обращения: 11.04.2019)

3. Fowler, M. Patterns of Enterprise Application Architecture [Текст] / M. Fowler, D. Rice, M. Foemmel. – Boston: Addison-Wesley, 2011 – 559 с.

4. Minecraft In Education: How Video Games Are Teaching Kids [Электронный ресурс]. – Режим доступа: https://www.gamespot.com/articles/minecraft-in-education-how-video-games-are-teaching-kids/1100-6400549/. – Minecraft In Education: How Video Games Are Teaching Kids - GameSpot . – (Дата обращения: 4.03.2019).

5. Фримен, Э. Паттерны проектирования [Текст] / Э. Фримен, Э. Робсон. – Санкт-Петербург: Питер, 2019 – 656 с.

6. Эванс, Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем / Э. Эванс. – Киев: Диалектика, 2016 – 448 с.

Приложение А. Исходный код программы

Файл GameInteractor.java:

```
package ru.cerebus.mc.game.framework.usecase;

import ru.cerebus.mc.game.framework.*;
import ru.cerebus.mc.game.framework.core.GamePlayer;
import ru.cerebus.mc.game.framework.core.game.Game;

public class GameInteractor {

    private GameRegistry registry;
    private MapManager mapManager;

    public GameInteractor(GameRegistry registry, MapManager mapManager) {
        this.registry = registry;
        this.mapManager = mapManager;
    }

    public void createGame(QueuedGame request) {
        GamePlugin gPlugin = registry.getGamePlugin(request.getGame());
        mapManager.loadMap(request.getId(), request.getGame(), request.getMap())
                .thenApplyAsync((world) -> {
                    Game<? extends GamePlayer> game = gPlugin.getGameFactory().create(request);
                    game.getPlayers().forEach(p -> registry.registerPlayer(p.getId(), game));
                    registry.registerWorld(world, game);
                    return game;
                }, Schedulers.io())
//                .thenApply((game) -> {
//                    configureGame(game, request);
//                    return game;
//                })
                .thenAccept(game -> {
                    gPlugin.addGame(game);
                    game.ready();
                    System.out.println("Game READY");
                }).exceptionally((e) -> {
                    e.printStackTrace();
                    return null;
                });
        //загрузить карту
        //создать экземпляр игры(Game.class)
        //загрузить данные игроков (алгоритм загрузки определяет игра)
        //уведомить о готовности принимать игроков этой игры
    }

    private void configureGame(Game game, QueuedGame config) {
        //TODO: moved logic to GameFactory
//        if (config.getTeams() != null) {
//            TeamGame teamGame = (TeamGame) game;
//            for (QueuedGame.Team t : config.getTeams()) {
//                Team team = teamGame.getTeam(t.getId());
//                for (UUID p : t.getPlayers()) {
//                    team.addPlayer((TeamPlayer) teamGame.getPlayer(p));
//                }
//            }
//        }
    }

}
```

Файл GameFramework.java:

```
package ru.cerebus.mc.game.framework;

import com.destroystokyo.paper.event.player.PlayerHandshakeEvent;
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.core.config.Configurator;
import org.bukkit.Bukkit;
import org.bukkit.configuration.file.FileConfiguration;
import org.bukkit.event.EventHandler;
import org.bukkit.event.Listener;
```

```
import org.bukkit.plugin.java.JavaPlugin;
import org.slf4j.LoggerFactory;
import ru.cerebus.mc.game.framework.core.game.Game;
import ru.cerebus.mc.game.framework.infrastructure.WorldManager;
import ru.cerebus.mc.game.framework.infrastructure.event.BukkitSubscribeManager;
import ru.cerebus.mc.game.framework.infrastructure.event.SubscribeManager;
import ru.cerebus.mc.game.framework.usecase.GameInteractor;
import ru.cerebus.mc.game.framework.usecase.GameRegistry;
import ru.cerebus.mc.game.framework.usecase.QueuedGame;
import ru.cerebus.mc.game.framework.usecase.Schedulers;

import java.util.*;

public class GameFramework extends JavaPlugin implements Listener, GameRegistry {

    private GameInteractor gameInteractor;
    private GameRegistry gameRegistry = this;
    private WorldManager worldManager;
    private static SubscribeManager subscribeManager;

    @Override
    public void onDisable() {

    }

    UUID testUUID = UUID.randomUUID();

    @Override
    public void onEnable() {
        Configurator.setRootLevel(Level.DEBUG);

        Bukkit.getPluginManager().registerEvents(this, this);
        Schedulers.init(this);
        subscribeManager = new BukkitSubscribeManager(this, gameRegistry);
        worldManager = new WorldManager(this, gameRegistry);
        gameInteractor = new GameInteractor(gameRegistry, worldManager);
        QueuedGame stub = new QueuedGame();
        stub.setGame("bedwars");
        stub.setId(1);
        Set<UUID> players = new HashSet<>();
        players.add(testUUID);
        stub.setPlayers(players);
        stub.setMap("test");
        Set<QueuedGame.Team> teams = new HashSet<>();
        QueuedGame.Team team = new QueuedGame.Team();
        team.setPlayers(players);
        team.setId("red");
        teams.add(team);
        stub.setTeams(teams);
        System.out.println("Test UUID " + testUUID);
        Bukkit.getScheduler().runTaskLater(this, () -> {
            System.out.println("CreateGame");
            gameInteractor.createGame(stub);
        }, 20);
    }

    @EventHandler
    public void onHandshake(PlayerHandshakeEvent event) {
        System.out.println("TESTLOGGER  DEBUG"  +  LoggerFactory.getLogger("Test").getClass().get-
Name());
        System.out.println("PlayerHandshakeEvent");
        if (testUUID != null) {
            event.setServerHostname("localhost");
            event.setSocketAddressHostname("localhost");
            event.setUniqueId(testUUID);
            event.setCancelled(false);
            testUUID = null;
        }
    }

    @Override
    public GamePlugin getGamePlugin(String game) {
        return (GamePlugin) Bukkit.getPluginManager().getPlugin(game);
    }

    private Map<String, Game> worlds = new HashMap<>();
```

```
    private Map<UUID, Game> players = new HashMap<>();

    @Override
    public void registerWorld(String world, Game game) {
        worlds.put(world, game);
    }

    @Override
    public void registerPlayer(UUID playerId, Game game) {
        players.put(playerId, game);
    }

    @Override
    public void unregisterGame(Game game) {
        worlds.entrySet().removeIf(entry -> entry.getValue() == game);
        players.entrySet().removeIf(entry -> entry.getValue() == game);
    }

    @Override
    public Game getGame(String world) {
        return worlds.get(world);
    }

    @Override
    public Game getGame(UUID playerId) {
        return players.get(playerId);
    }

    public static SubscribeManager getSubscribeManager() {
        return subscribeManager;
    }

    @Override
    public FileConfiguration getConfig() {
        return super.getConfig();
    }
}
```

Файл Game.java:

```
package ru.cerebus.mc.game.framework.core.game;

import lombok.Getter;
import ru.cerebus.mc.game.framework.core.GamePlayer;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

public abstract class Game<P extends GamePlayer> {

    @Getter
    private final long id;
    private Map<UUID, P> players = new HashMap<>();
    @Getter
    private GameState state;

    public Game(long id) {
        this.id = id;
    }

    public Collection<P> getPlayers() {
        return players.values();
    }

    /**
     * Получение данных игрока
     * @param playerId - идентификатор игрока
     * @return Объект игрока
     */
    public P getPlayer(UUID playerId) {
        return players.get(playerId);
    }

    public void addPlayer(P player) {
```

```
            players.put(player.getId(), player);
        }

        /**
         * Разобрать конфигурацию карты, создать необходимые команды
         */
        public abstract void loadMap();

        /**
         * Переход игры в другое состояние
         * @param state новое состояние игры
         */
        public void setState(GameState state) {
            if (this.state != null) {
                this.state.exit();
            }
            this.state = state;
            if (this.state != null) {
                this.state.entry();
            }

        }

    //    Observable<GameState> state();

        // set Waiting state
        // calls at end of loadMap method
        public void ready() {

        }

        public void finish() {

        }

        public abstract int getCapacity();
    }
```

Файл StartingState.java:

```
    package ru.cerebus.mc.game.framework.core.game.starting;

    import io.reactivex.Observable;
    import io.reactivex.subjects.BehaviorSubject;
    import ru.cerebus.mc.game.framework.core.*;
    import ru.cerebus.mc.game.framework.core.game.Game;
    import ru.cerebus.mc.game.framework.core.game.GameState;

    import java.util.concurrent.TimeUnit;

    public class StartingState<G extends Game<? extends GamePlayer>> extends GameState<G> {

        private GameStateFactory playStateFactory;

        private BehaviorSubject<Integer> timeRemaining;
        private CancellableTask countingTask;
        private TaskProvider taskProvider;
        private SpawnCageProvider spawnCageProvider;

        public StartingState(G game, GameStateFactory playStateFactory, TaskProvider taskProvider,
SpawnCageProvider spawnCageProvider) {
            super(game);
            this.taskProvider = taskProvider;
            this.spawnCageProvider = spawnCageProvider;
            timeRemaining = BehaviorSubject.createDefault(15 + 1); //TODO: get value from config repos-
itory
        }

        @Override
        public void entry() {
            countingTask = taskProvider.runTaskInterval(1, TimeUnit.SECONDS, this::timer);
        }

        @Override
        public void exit() {
            countingTask.cancel();
```

```
            timeRemaining.onComplete();

            spawnCageProvider.freeAll();
        }

        public int getTimeRemaining() {
            return timeRemaining.getValue();
        }

        public Observable<Integer> timeRemaining() {
            return timeRemaining;
        }

        private void timer() {
            timeRemaining.onNext(timeRemaining.getValue() - 1);
            if (getTimeRemaining() <= 0) {
                game.setState(playStateFactory.create(game));
            }
        }

    }
```

Файл PlayState.java:

```
    package ru.cerebus.mc.game.framework.core.game.play;

    import ru.cerebus.mc.game.framework.core.*;
    import ru.cerebus.mc.game.framework.core.game.Game;
    import ru.cerebus.mc.game.framework.core.game.GameState;

    import java.util.concurrent.TimeUnit;

    public class PlayState<G extends Game<? extends GamePlayer>> extends GameState<G> {

        private GameMechanicsProvider gameMechanicsProvider;
        protected TaskProvider taskProvider;
        private CancellableTask timeExpires;

        public PlayState(G game, GameMechanicsProvider gameMechanicsProvider, TaskProvider taskPro-
vider) {
            super(game);
            this.gameMechanicsProvider = gameMechanicsProvider;
            this.taskProvider = taskProvider;
        }

        @Override
        public void entry() {
            //TODO: get time from config (or game map)
            timeExpires = taskProvider.runTaskLater(60, TimeUnit.MINUTES, this::forceEndGame);
            gameMechanicsProvider.enableAll();
        }

        @Override
        public void exit() {
            timeExpires.cancel();
            gameMechanicsProvider.disableAll();
        }

        private void forceEndGame() {
            game.finish();
        }

    }
```

Файл WaitingState.java:

```
    package ru.cerebus.mc.game.framework.core.game.waiting;

    import ru.cerebus.mc.game.framework.core.*;
    import ru.cerebus.mc.game.framework.core.game.Game;
    import ru.cerebus.mc.game.framework.core.game.GameState;

    import java.util.concurrent.TimeUnit;
```

```
public class WaitingState<G extends Game<? extends GamePlayer>> extends GameState<G> {

    private TaskProvider taskProvider;
    private GameQueueNotifier gameQueueNotifier;
    private SpawnCageProvider spawnCageProvider;
    private GameStateFactory startingStateFactory;
    private CancellableTask playersMonitoring;

    public WaitingState(G game,
                        TaskProvider taskProvider,
                        GameQueueNotifier gameQueueNotifier,
                        SpawnCageProvider spawnCageProvider,
                        GameStateFactory startingStateFactory) {
        super(game);
        this.taskProvider = taskProvider;
        this.startingStateFactory = startingStateFactory;
        this.spawnCageProvider = spawnCageProvider;
        this.gameQueueNotifier = gameQueueNotifier;
    }

    @Override
    public void entry() {
        //TODO: check it is core logic
        gameQueueNotifier.ready(game);
        spawnCageProvider.imprisonAll();
        playersMonitoring = taskProvider.runTaskInterval(1, TimeUnit.SECONDS, this::checkOnline-
Players);
        //TODO broken logic: game.players() always contains all players data; monitor online players
count
//        playersMonitoring = game.players().subscribe((t) -> {
//            if (((Set) t).size() >= (game.getMap().getCapacity() / 2)) {
//                playersMonitoring.dispose();
//                playersMonitoring = null;
//                game.setState(startingStateFactory.create(game));
//            }
//        });
    }

    @Override
    public void exit() {
        playersMonitoring.cancel();
    }

    private void checkOnlinePlayers() {
        if (game.getPlayers().stream().filter(GamePlayer::isOffline).count() >= game.getCapacity()
/ 2) {
            game.setState(startingStateFactory.create(game));
        }
    }

}
```

Файл EntityManager.java:

```
package ru.cerebus.mc.game.framework.infrastructure;

import net.citizensnpcs.api.npc.NPC;
import net.citizensnpcs.api.npc.NPCRegistry;
import net.citizensnpcs.api.trait.trait.Equipment;
import net.citizensnpcs.api.trait.trait.Inventory;
import org.bukkit.Bukkit;
import org.bukkit.Location;
import org.bukkit.entity.EntityType;
import org.bukkit.entity.LivingEntity;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.Listener;
import org.bukkit.event.player.PlayerJoinEvent;
import org.bukkit.event.player.PlayerQuitEvent;
import org.bukkit.inventory.EntityEquipment;
import org.bukkit.inventory.ItemStack;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import ru.cerebus.mc.game.framework.core.game.Game;
import ru.cerebus.mc.game.framework.core.GamePlayer;
import ru.cerebus.mc.game.framework.core.data.Position;
```

```
        import ru.cerebus.mc.game.framework.usecase.GameRegistry;
        import ru.cerebus.mc.game.framework.util.LivingEntityUtil;

        import java.util.HashMap;
        import java.util.Map;
        import java.util.UUID;

        public class EntityManager implements Listener {

            private Logger log = LoggerFactory.getLogger(EntityManager.class);
            private WorldManager worldManager;
            private GameRegistry gameRegistry;
            private NPCRegistry npcs; //anonymus NPCRegistry with MemoryDataStore
            private Map<GamePlayer, NPC> entites = new HashMap<>();

            @EventHandler
            public void onPlayerJoin(PlayerJoinEvent e) {
                Player entity = e.getPlayer();
                NPC npc = getNPC(entity.getUniqueId());
                entity.teleport(npc.getEntity().getLocation());
                entity.getInventory().setContents(npc.getTrait(Inventory.class).getContents());
                EntityEquipment entityEquipment = entity.getEquipment();
                if (entityEquipment != null) {
                    Map<Equipment.EquipmentSlot, ItemStack> equipment = npc.getTrait(Equipment.class).getE-
quipmentBySlot();
                    entityEquipment.setHelmet(equipment.get(Equipment.EquipmentSlot.HELMET));
                    entityEquipment.setChestplate(equipment.get(Equipment.EquipmentSlot.CHESTPLATE));
                    entityEquipment.setLeggings(equipment.get(Equipment.EquipmentSlot.LEGGINGS));
                    entityEquipment.setBoots(equipment.get(Equipment.EquipmentSlot.BOOTS));
                    entityEquipment.setItemInMainHand(equipment.get(Equipment.EquipmentSlot.HAND));
                    entityEquipment.setItemInOffHand(equipment.get(Equipment.EquipmentSlot.OFF_HAND));
                }
                LivingEntity npcEntity = (LivingEntity) npc.getEntity();
                LivingEntityUtil.copyAttributes(npcEntity, entity);
                entity.setHealth(npcEntity.getHealth());
                entity.setVelocity(npcEntity.getVelocity());
                npc.despawn();
            }

            @EventHandler
            public void onPlayerQuit(PlayerQuitEvent e) {
                Player entity = e.getPlayer();
                NPC npc = getNPC(entity.getUniqueId());
                npc.spawn(entity.getLocation());
                npc.getTrait(Inventory.class).setContents(entity.getInventory().getContents());
                EntityEquipment entityEquipment = entity.getEquipment();
                if (entityEquipment != null) {
                    Equipment equipment = npc.getTrait(Equipment.class);
                    equipment.set(Equipment.EquipmentSlot.HELMET, entityEquipment.getHelmet());
                    equipment.set(Equipment.EquipmentSlot.CHESTPLATE, entityEquipment.getChestplate());
                    equipment.set(Equipment.EquipmentSlot.LEGGINGS, entityEquipment.getLeggings());
                    equipment.set(Equipment.EquipmentSlot.BOOTS, entityEquipment.getBoots());
                    equipment.set(Equipment.EquipmentSlot.HAND, entityEquipment.getItemInMainHand());
                    equipment.set(Equipment.EquipmentSlot.OFF_HAND, entityEquipment.getItemInOffHand());
                }
                LivingEntity npcEntity = (LivingEntity) npc.getEntity();
                LivingEntityUtil.copyAttributes(entity, npcEntity);
                npcEntity.setHealth(entity.getHealth());
                npcEntity.setVelocity(entity.getVelocity());
            }

            /**
             * Активировать сущность игрока
             * @param player игрок
             */
            public void spawn(Game game, GamePlayer player, Position position) {
                NPC npc = npcs.createNPC(EntityType.PLAYER, player.getName());
                npc.addTrait(new Inventory());
                npc.addTrait(new Equipment());
                entites.put(player, npc);
                Location location = worldManager.getLocation(game, position);
                Player entity = Bukkit.getPlayer(player.getId());
                if (entity != null && entity.isOnline()) {
                    log.warn("Try spawn player entity, when it is online!");
                } else if (!npc.spawn(location)) {
                    log.warn("Can't spawn NPC at given position: " + position);
```

```
        }
    }

    /**
     * Деактивировать сущность игрока
     * @param player игрок
     */
    public void despawn(GamePlayer player) {
        Player entity = Bukkit.getPlayer(player.getId());
        if (entity != null) {
            entity.kickPlayer("Despawn");
            log.warn("Try to despawn entity, when player is online!");
        }
        NPC npc = getNPC(player.getId());
        npc.despawn();
        npc.destroy();
    }

    public void teleport(GamePlayer player, Position position) {

    }

    public void destroy(GamePlayer player) {

    }

    private NPC getNPC(UUID playerId) {
        Game game = gameRegistry.getGame(playerId);
        GamePlayer player = game.getPlayer(playerId);
        return entites.get(player);
    }

}
```

Файл WorldManager.java:

```
package ru.cerebus.mc.game.framework.infrastructure;

import javafx.util.Pair;
import org.apache.commons.io.FileUtils;
import org.bukkit.*;
import org.bukkit.event.EventHandler;
import org.bukkit.event.Listener;
import org.bukkit.event.world.WorldInitEvent;
import org.bukkit.plugin.java.JavaPlugin;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import ru.cerebus.mc.game.framework.core.game.Game;
import ru.cerebus.mc.game.framework.core.data.Position;
import ru.cerebus.mc.game.framework.usecase.GameRegistry;
import ru.cerebus.mc.game.framework.usecase.MapManager;
import ru.cerebus.mc.game.framework.usecase.Schedulers;
import ru.cerebus.mc.game.framework.util.CompletableFutureUtil;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
import java.util.Queue;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.LinkedBlockingQueue;

public class WorldManager implements MapManager, Listener {

    private Logger log = LoggerFactory.getLogger(WorldManager.class.getSimpleName());

    private Server server;
    private GameRegistry gameRegistry;
    private Map<Game, World> gameWorlds = new HashMap<>();
    private Queue<LoadingWorld> loadingWorlds = new LinkedBlockingQueue<>();

    public WorldManager(JavaPlugin plugin, GameRegistry gameRegistry) {
        this.server = plugin.getServer();
        this.gameRegistry = gameRegistry;
```

```
                System.out.println("LOGGER LEVEL " + log.isDebugEnabled());
                server.getPluginManager().registerEvents(this, plugin);
                server.getScheduler().runTaskTimer(plugin, () -> {
                    if (!loadingWorlds.isEmpty()) {
                        long start = System.currentTimeMillis();
                        int cnt = 0;
                        LoadingWorld world = loadingWorlds.peek();
                        while (world != null) {
                            while (world.hasNext() && System.currentTimeMillis() - start < MAX_LOAD_TIME) {
                                Pair<Integer, Integer> next = world.next();
                                world.getWorld().loadChunk(next.getKey(), next.getValue());
                                cnt++;
                            }
                            if (world.hasNext()) {
                                // приостановить загрузку, т.к. загрузка завершилась по превышению времени,
мир из очереди не удалять
                                world = null;
                            } else {
                                loadingWorlds.remove(world);
                                world.loaded();
                                world = loadingWorlds.peek();
                            }
                        }
                        log.info("Load {} chunks in {}ms.", cnt, System.currentTimeMillis() - start);
                    }
                }, 0, 1);
            }

            @EventHandler
            public void onWorldInit(WorldInitEvent event) {
                event.getWorld().setKeepSpawnInMemory(false);
            }

            public World getWorld(Game game) {
                return gameWorlds.get(game);
            }

            public Location getLocation(Game game, Position p) {
                return new Location(getWorld(game), p.getX(), p.getY(), p.getZ(), p.getYaw(), p.getPitch());
            }

            @Override
            public CompletableFuture<String> loadMap(long gameId, String game, String map) {
                String worldId = createWorldId(gameId, game, map);
                return CompletableFuture.runAsync(() -> copyWorld(game, map, worldId), Schedulers.io())
                        .thenComposeAsync((nil) -> loadWorld(worldId), Schedulers.bukkit());
            }

            public CompletableFuture<String> loadWorld(String worldId) {
                World world = new WorldCreator(worldId).createWorld();
                if (world == null) {
                    return CompletableFutureUtil.
                            completedExceptionally(new  RuntimeException("Can't  load  "  +  worldId  +  "
world!"));
                } else {
                    //TODO: read world configuration file and queue needed chunks
                    LoadingWorld loadingWorld = new LoadingWorld(world, 0, 0, 16, 16);
                    loadingWorlds.add(loadingWorld);
                    return loadingWorld.getCallback();
                }
            }

            public void copyWorld(String game, String map, String worldId) {
                log.debug("CopyWorld called!");
                Path repoPath = mapRepository.resolve(game).resolve(map);
                Path worldPath = server.getWorldContainer().toPath().resolve(worldId);
                try {
                    FileUtils.deleteDirectory(worldPath.toFile());
                    log.debug("Old world deleted.");
                    Files.copy(repoPath, worldPath);
                    log.debug("Map {} copied to world {} successfully.", map, worldId);
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
```

```
            private String createWorldId(long gameId, String game, String map) {
                return String.format("%s-%d-%s", game, gameId % GAME_ID_TRIM, map);
            }

            private static final int GAME_ID_TRIM = 10000;
            private static final int MAX_LOAD_TIME = 25;
            private static final Path mapRepository = Paths.get("D:\\Games\\Minecraft\\Repo");

        }
```

Файл DelayedRespawningService.java:

```
        package ru.cerebus.mc.game.framework.infrastructure;

        import org.bukkit.Bukkit;
        import org.bukkit.entity.Player;
        import org.bukkit.event.EventHandler;
        import org.bukkit.event.player.PlayerRespawnEvent;
        import org.bukkit.plugin.java.JavaPlugin;
        import org.bukkit.potion.PotionEffect;
        import org.bukkit.potion.PotionEffectType;
        import org.bukkit.scheduler.BukkitTask;
        import ru.cerebus.mc.game.framework.core.GamePlayer;
        import ru.cerebus.mc.game.framework.core.RespawningService;
        import ru.cerebus.mc.game.framework.core.game.Game;
        import ru.cerebus.mc.game.framework.infrastructure.event.SubscribeManager;
        import ru.cerebus.mc.game.framework.util.MinecraftTimeUtil;

        import java.util.HashMap;
        import java.util.Map;
        import java.util.UUID;
        import java.util.concurrent.TimeUnit;

        public class DelayedRespawningService implements RespawningService {

            private WorldManager worldManager;
            private Game game;
            private JavaPlugin plugin;

            private Map<UUID, RespawningState> respawningPlayers = new HashMap<>();

            public DelayedRespawningService(WorldManager worldManager, SubscribeManager subscribeManager,
Game game, JavaPlugin plugin) {
                this.worldManager = worldManager;
                this.game = game;
                subscribeManager.subscribe(game, this);
            }

            @Override
            public void respawn(GamePlayer player) {
                Player entity = Bukkit.getPlayer(player.getId());
                if (entity == null) return;

                RespawningState state = new RespawningState(5);
                BukkitTask timer = Bukkit.getScheduler().runTaskTimer(plugin, () -> {
                    int delayRemining = state.tick();
                    if (delayRemining <= 0) {
                        state.exit();
                        player.respawn();
                    } else {

                        //show title message
                    }
                },  MinecraftTimeUtil.toTicks(1,  TimeUnit.SECONDS),  MinecraftTimeUtil.toTicks(1,  TimeU-
nit.SECONDS));
                state.setDelayTimer(timer);
                respawningPlayers.put(player.getId(), state);

                entity.spigot().respawn();
                //.handleRespawnLocation()
        //        entity.teleport(worldManager.getLocation(game, player.getSpawnPosition()));
                entity.setWalkSpeed(0); //TODO проверить прыжки
                entity.addPotionEffect(new  PotionEffect(PotionEffectType.INVISIBILITY,  Integer.MAX_VALUE,
1));
            }
```

```
    @Override
    public void complete(GamePlayer player) {
        //TODO реализовать действия по контракту (JavaDoc)
        respawningPlayers.remove(player.getId()).exit();
    }

    @Override
    public void cancel(GamePlayer player) {
        complete(player);
    }

    @EventHandler
    public void handleRespawnLocation(PlayerRespawnEvent e) {
        GamePlayer player = game.getPlayer(e.getPlayer().getUniqueId());
        e.setRespawnLocation(worldManager.getLocation(game, player.getSpawnPosition()));
    }

}
```

Файл TeamSpawnCageProvider.java:

```
package ru.cerebus.mc.game.framework.infrastructure.local;

import org.bukkit.Location;
import org.bukkit.Material;
import org.bukkit.World;
import org.bukkit.event.EventHandler;
import org.bukkit.event.block.BlockBreakEvent;
import org.bukkit.event.block.BlockPlaceEvent;
import ru.cerebus.mc.game.framework.core.SpawnCageProvider;
import ru.cerebus.mc.game.framework.core.Team;
import ru.cerebus.mc.game.framework.core.TeamGame;
import ru.cerebus.mc.game.framework.core.TeamPlayer;
import ru.cerebus.mc.game.framework.infrastructure.EntityManager;
import ru.cerebus.mc.game.framework.infrastructure.WorldManager;

public class TeamSpawnCageProvider implements SpawnCageProvider {

    private WorldManager worldManager;
    private EntityManager entityManager;

    private TeamGame<TeamPlayer, Team<TeamPlayer>> teamGame;

    public TeamSpawnCageProvider(TeamGame teamGame) {
        this.teamGame = teamGame;
    }

    @Override
    public void imprisonAll() {
        teamGame.getTeams().forEach(team -> {
            Location location = worldManager.getLocation(teamGame, team.getSpawn()).add(0, 1, 0);
            formCage(location, 3 /*TODO: get from config*/, Material.GLASS);
            team.getPlayers().forEach(player -> entityManager.teleport(player, team.getSpawn()));
        });
    }

    @Override
    public void freeAll() {

    }

    @EventHandler
    public void onBlockBreak(BlockBreakEvent event) {
        event.setCancelled(true);
    }

    @EventHandler
    public void onBlockPlace(BlockPlaceEvent event) {
        event.setCancelled(true);
    }

    private void formCage(Location location, int size, Material material) {
        int offset = (size - 1) / 2;
        int floor = location.getBlockY();
        int roof = location.getBlockY() + size + 1;
        World world = worldManager.getWorld(teamGame);
```

```
                for (int x = location.getBlockX() - offset - 1; x <= location.getBlockX() + offset + 1; x++)
{
                    for (int z = location.getBlockZ() - offset - 1; z <= location.getBlockZ() + offset + 1;
z++) {
                        world.getBlockAt(x, floor, z).setType(material);
                        world.getBlockAt(x, roof, z).setType(material);
                    }
                }
                formWallX(world, location.getBlockX() - offset - 1, floor + 1, location.getBlockZ() - offset
- 1,
                        size, size + 2, material);
                formWallX(world, location.getBlockX() - offset - 1, floor + 1, location.getBlockZ() + offset
+ 1,
                        size, size + 2, material);
                formWallZ(world, location.getBlockX() - offset - 1, floor + 1, location.getBlockZ() - off-
set,
                        size, size, material);
                formWallZ(world, location.getBlockX() + offset + 1, floor + 1, location.getBlockZ() - off-
set,
                        size, size, material);

            }

            private void destroyCage(Location location, int size) {

            }

            private void formWallX(World world, int x0, int y0, int z0, int height, int width, Material
material) {
                for (int x = x0; x <= x0 + width; x++) {
                    for (int y = y0; y <= y0 + height; y++) {
                        world.getBlockAt(x, y, z0).setType(Material.GLASS);
                    }
                }
            }

            private void formWallZ(World world, int x0, int y0, int z0, int height, int width, Material
material) {
                for (int z = z0; z <= z0 + width; z++) {
                    for (int y = y0; y <= y0 + height; y++) {
                        world.getBlockAt(x0, y, z).setType(Material.GLASS);
                    }
                }
            }
        }
```