СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные технологии и мобильные приложения играют ключевую роль в повседневной жизни современного человека, предоставляя разнообразные возможности для развлечения, обучения и общения. Одной из популярных многопользовательских онлайн-игр является «Slither.io», которая привлекает внимание игроков своей динамикой и соревновательным геймплеем.

«Slither.io» — это онлайн-игра, в которой игрок управляет маленьким змеем, который становится все длиннее по мере того, как он употребляет еду на игровом поле. Цель игры - выжить как можно дольше, избегая столкновения с другими игроками и контролируя пространство вокруг себя. Игра получила широкое признание благодаря своей простоте и захватывающему игровому процессу, а также возможности играть в режиме реального времени с игроками со всего мира.

Цель данной курсовой работы заключается в исследовании и разработке программного клиента для игры «Slither.io», который подключается к оригинальным серверам игры и позволяет пользователю играть с реальными игроками. В рамках проекта будут изучены протоколы связи с серверами «Slither.io», реализованы основные механики игрового процесса и разработан интуитивно понятный пользовательский интерфейс. Основное внимание уделяется обеспечению плавного и стабильного игрового процесса, а также интеграции с оригинальными серверами для поддержки многопользовательских сессий.

В данной пояснительной записке отражены следующие разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию;

1 Анализ литературных источников

1.1 История

Игры – один из важнейших факторов человеческой культуры. Существует множество игр, но немногие могут похвастаться такой широкой популярностью и значимостью, как «Slither.io». Изначально задуманная как веб-игра, «Slither.io» быстро завоевала популярность среди пользователей благодаря своему простому, но захватывающему игровому процессу и возможности соревноваться с реальными игроками со всего мира.

«Slither.io» — многопользовательская компьютерная игра, разработанная Стивом Хаузом и выпущенная для iOS, Android и веб-браузеров в марте 2016 года. По своей концепции «Slither.io» похожа на классическую «Змейку» и другую популярную браузерную игру «Agar.io». Игроки управляют «змеями» — длинными существами, которые передвигаются по обширному игровому полю и поедают разбросанные по нему разноцветные гранулы, становясь длиннее. Цель игрока — сделать свою змею как можно длиннее, избегая столкновений со змеями других игроков.

С момента своего появления «Slither.io» стала феноменом, привлекая миллионы игроков и порождая множество аналогов и клонов. Успех игры можно объяснить её доступностью (она доступна в веб-версии и на мобильных платформах), простотой управления и возможностью соревноваться в реальном времени с игроками со всего мира.

Игра возымела огромную популярность после многочисленных летсплеев на YouTube. Согласно рейтингу Alexa за июль 2016 года, браузерная версия Sither.io попала в топ самых посещаемых сайтов, а версия для iOS возглавила чарт самых загружаемых игр в App Store. Игра получила в основном положительные отзывы от критиков — основными её плюсами называли дизайн и кастомизацию змеи, но в качестве недостатков отмечали низкую реиграбельность и высокую цену за удаление рекламы.
[1]

1.2 Анализ прототипов

Перед разработкой программного клиента для игры «Slither.io» важно провести анализ прототипов, чтобы понять их особенности, преимущества и недостатки. Этот анализ поможет определить лучшие практики и основные характеристики, которые должны быть учтены при создании программного клиента.

1.2.1 Оригинальный «Slither.io»

Оригинальная версия «Slither.io», разработанная Стивом Хаусом, представляет собой многопользовательскую онлайн-игру, в которой игроки управляют змеями, соревнуясь за размер и выживание. Игровой процесс заключается в поедании светящихся точек и избегании столкновений с другими змеями. [2] Простота игрового процесса и возможность мгновенного участия сделали slither.io популярной среди игроков всех возрастов. Основные особенности игры включают:

- Многопользовательский режим, позволяющий соревноваться с игроками со всего мира.
- Простое управление, подходящее для игроков любого уровня опыта.
- Возможность играть как на веб-платформе, так и на мобильных устройствах.

Рисунок 1.2.1 – Slither.io

1.2.2 Snake (1976)

Оригинальная версия игры "Snake" (1976), созданная компанией Gremlin Industries, представляет собой классическую аркадную игру, в которой игрок управляет змейкой, пытающейся выжить и расти, путем поедания еды и избегания столкновений со стенами и собственным телом.[3] Простота игрового процесса и захватывающий геймплей сделали Snake популярной иконой в мире видеоигр. Основные особенности игры включают:

- Простое управление обеспечивает игроку возможность управлять направлением движения змейки с помощью клавиш на клавиатуре, что делает управление интуитивно понятным и доступным для игроков любого уровня опыта.
- Простое управление обеспечивает игроку возможность управлять направлением движения змейки с помощью клавиш на клавиатуре или

джойстика на аркадном автомате, что делает управление интуитивно понятным и доступным для игроков любого уровня опыта.

- Избегание столкновений становится важной частью игрового процесса, поскольку игрок должен аккуратно управлять змейкой, чтобы избежать столкновений со стенами и собственным телом змейки. В противном случае игра завершается.

Рисунок 1.2.1 – Snake

1.3 Постановка задачи

В своей курсовой работе я поставила следующие задачи:

1. Проанализировать требования, указанные в задании на курсовое проектирование.
2. Изучить существующие прототипы, их достоинства и недостатки.
3. Разработать программное средство, позволяющее пользователю подключаться к оригинальным серверам «Slither.io» и играть с реальными игроками.
4. Обеспечить плавный и стабильный игровой процесс.
5. Создать интуитивно понятный пользовательский интерфейс.
6. Отладить и проанализировать программное средство.

Для разработки программного клиента будут использоваться язык программирования Java и библиотека Swing для реализации игрового интерфейса.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Описание функциональных требований

На основе проведенного анализа и с учетом требований, указанных в задании на курсовое проектирование, выделяются следующие основные требования к функционалу программного средства:

1. Создание подключения к оригинальным серверам игры «Slither.io».
2. Изучение и реализация протокола, по которому взаимодействуют клиент и сервер игры «Slither.io».
3. Реализация механизма обновления игрового состояния с сервера на клиенте с минимальной задержкой для обеспечения плавного игрового процесса.
4. Отображение игрового поля с учетом данных, полученных от серверов игры.
5. Реализация управления змейкой с помощью мыши или клавиатуры.
6. Визуализация движения змейки и других игровых объектов на экране.
7. Создание интуитивно понятного пользовательского интерфейса, включая элементы управления, информационные панели и меню.

2.2 Спецификация функциональных требований

1. Создание подключения к оригинальным серверам игры «Slither.io»:
    1. Программное средство должно уметь устанавливать соединение с серверами игры «Slither.io» по протоколу TCP/IP.
    2. Требуется обработка аутентификации и обмена данных между клиентом и сервером для корректной игровой сессии.
2. Изучение и реализация протокола, по которому взаимодействуют клиент и сервер игры «Slither.io»:
    1. Требуется изучение протокола взаимодействия клиента и сервера игры «Slither.io» для корректной интерпретации и обработки данных.
    2. На основе изученного протокола требуется разработать механизмы отправки и приема сообщений между клиентом и сервером. Это включает в себя кодирование и декодирование данных в соответствии с протоколом[4].
3. Реализация механизма обновления игрового состояния с сервера на клиенте с минимальной задержкой для обеспечения плавного игрового процесса:

1. Требуется реализация эффективного механизма обновления игрового состояния с сервера на клиенте с минимальной задержкой для обеспечения плавного и непрерывного игрового процесса. Логика игры:

4. Отображение игрового поля с учетом данных, полученных от серверов игры:
    1. Программное средство должно обеспечить отображение игрового поля, а также всех игровых объектов, полученных от серверов игры «Slither.io».
    2. Требуется корректная синхронизация отображаемых данных с актуальным состоянием игры на сервере.
5. Реализация управления змейкой с помощью мыши или клавиатуры:
    1. Пользователь должен иметь возможность управлять змейкой с помощью мыши.
    2. Пользователь должен иметь возможность ускорять змею с помошью нажатия клавиши мыши.
6. Визуализация движения змейки и других игровых объектов на экране.
    1. Программное средство должно обеспечить плавную визуализацию движения змейки и других игровых объектов на экране пользователя.
    2. Для создания плавного и реалистичного визуального опыта необходима анимация движения змейки и других игровых объектов при их перемещении по игровому полю.
    3. Программное средство должно корректно определять и обрабатывать коллизии между змейкой и другими объектами игры, например, при соприкосновении со стенами или другими змейками.
7. Создание интуитивно понятного пользовательского интерфейса, включая элементы управления, информационные панели и меню.
    1. Программное средство должно иметь удобный и понятный пользовательский интерфейс с элементами управления, информационными панелями и меню.

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Алгоритм получения списка игровых серверов

Этот алгоритм выполняет HTTP-запрос к удаленному серверу для получения текстового файла, содержащего данные о серверах игры. Затем он преобразует полученные данные в массив URI-адресов серверов, используя вычисленные IP-адреса и порты, и возвращает этот массив для дальнейшего использования в клиентском коде.

Рисунок 3.1 – Блок-схема алгоритма получения списка игровых серверов

3.2 Алгоритм обработки полученных пакетов от сервера

Данный алгоритм принимает массив байтов, извлекает команду из них и в зависимости от этой команды вызывает соответствующий метод для обработки данных сообщения.

Рисунок 3.2 – Блок-схема алгоритма обработки полученных пакетов от сервера

3.3 Алгоритм отображения мини-карты

Этот алгоритм обеспечивает визуальное отображение мини-карты игрового мира и текущего положения змеи на этой карте, что помогает игроку ориентироваться в игровом пространстве.

Рисунок 3.3 – Блок-схема алгоритма отображения мини-карты

3.4 Алгоритм подключения к серверу игры «Slither.io»

Этот алгоритм обеспечивает подключение клиента к серверу игры, позволяя выбирать случайный сервер из списка или ввести его самому и потом подключаться к серверу по указанному URI.

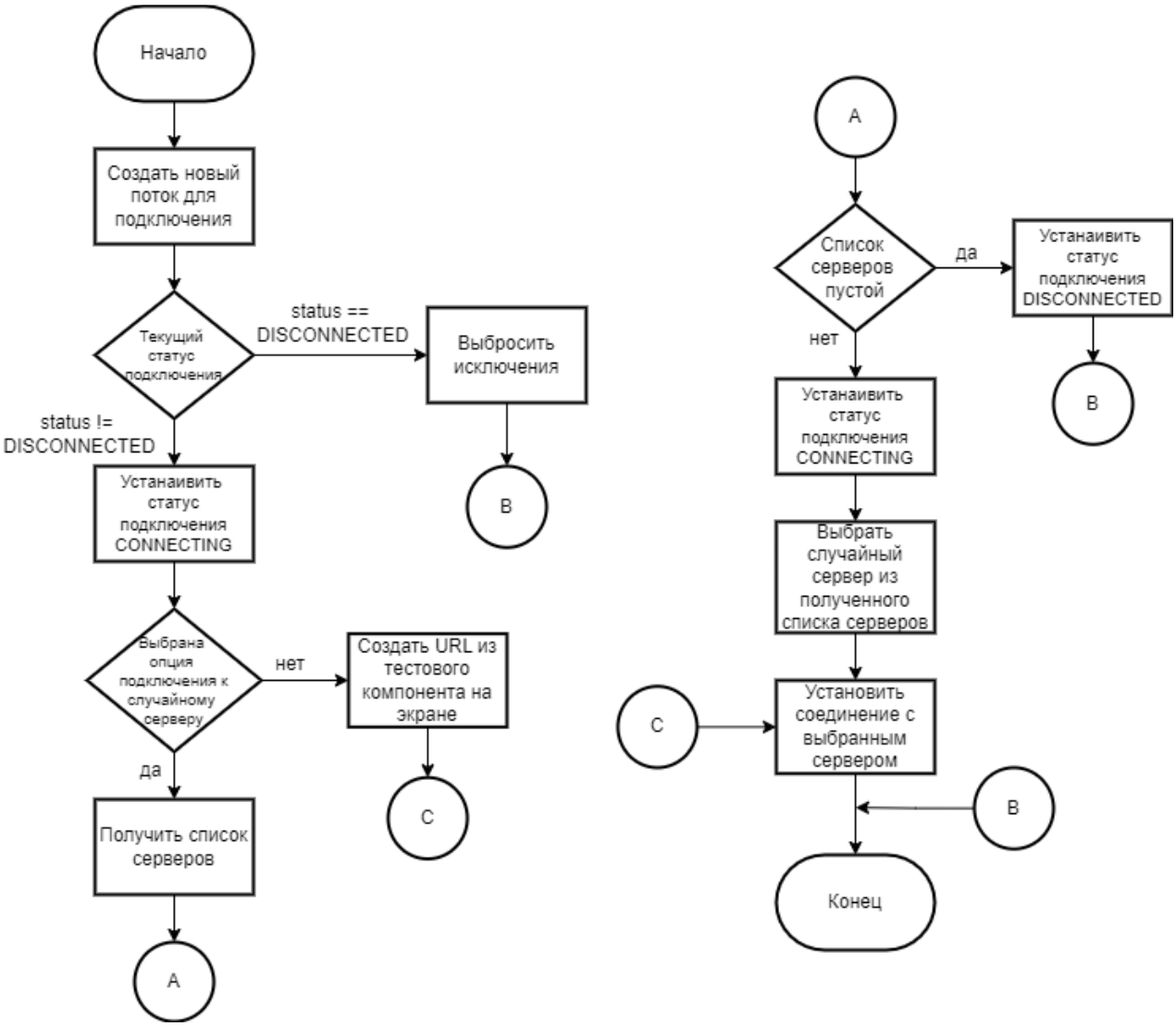

Рисунок 3.3 – Блок-схема алгоритма подключения к серверу игры «Slither.io»

3.5 Алгоритм декодирование «секретного» сообщения

Этот код реализует декодирование «секретного» сообщения, которое клиент отправляет обратно на сервер после его первоначального получения.

Рисунок 3.3 – Блок-схема алгоритма декодирование «секретного» сообщения

4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Проектирование модулей программного средства

Разработка программного средства выполняется на основе алгоритмов, которые были представлены в разделе 3, а также на основе функциональных требований, представленных в разделе 2.

4.2 Структура модулей программы

Программа состоит из следующих файлов:

1. Файл Main.java входной точки программы;
2. Файл Snake.java представляет собой класс, который описывает объект змеи
3. Файл SnakeBodyPart.java представляет собой класс, который описывает объект тела змеи
4. Файл Prey.java представляет собой класс, который описывает объект добычи
5. Файл Food.java представляет собой класс, который описывает объект еды
6. Файл Pleyer.java представляет собой абстрактный класс, который описывает объект игрока, также содержит вложенный статический класс Wish, который представляет желание игрока.
7. Файл MySlitherJFrame.java представляет собой класс, который является главным окном графического интерфейса для игры «MySlither.io»
8. Файл MySlitherCanvas.java представляет собой класс, который отвечает за отображение состояния игры на экране и обработку ввода пользователя через мышь.
9. Файл MySlitherWebSocketClient.java представляет собой класс, который обрабатывает взаимодействие с сервером.
10. Файл MySlitherModel.java представляет собой класс, который отвечает за управление моделью игры Slither.io. Он содержит логику, связанную с обработкой состояния игрового мира.

4.3 Описание модуля Main

Модуль Main является точка входа в программу. Он запускает приложение «MySlither.io», инициализируя оконный интерфейс и начиная его выполнение.

Таблица 4.3 - Методы модуля Main

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
main	Точка входа в программу. В нем создается главное окно программы, и начинается выполнение работы программы.	public static void main(String[] args) throws IOException	args	Массив аргументов командной строки

4.4 Описание модуля Snake

Модуль Snake представляет собой объект змеи в игре. Он используются для управления поведением и характеристиками змеи в игре.

Таблица 4.4 - Методы модуля Snake

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
Snake	Метод, запускающийся при создании объекта класса (конструктор)	public Snake(int id, String name, Color skin, double x, double y, double wang, double ang, double speed, double fam, Deque<SnakeBodyPart> body, MySlitherModel model)	id	Идентификатор змеи
			name	Имя змеи
			skin	Цвет змеи
			x	Координата x змеи

Продолжение таблицы 4.4

			y	Координата y змеи
			wang	Направление движения змеи
			ang	Текущий угол поворота змеи
			speed	Желаемый угол поворота змеи
			fam	Соотношение частей тела змеи
			body	Последовательность частей тела змеи
			model	Модель игры
getSc	Вычисляет и возвращает размер змеи на основе количества элементов в ее теле.	private double getSc()	-	-
getFsp	Вычисляет и возвращает фактор скорости змеи	private double getFsp()	-	-
isBoosting	Проверяет, ускоряется ли змея в данный момент	boolean isBoosting()	-	-

Продолжение таблицы 4.4

getFam	Возвращает ускорение змеи.	double getFam()	-	-
setFam	Устанавливает ускорение змеи	void setFam(double fam)	fam	Ускорение змеи

4.5 Описание модуля Prey

Модуль Prey представляет собой добычу. которую змеи могут съесть в игре.

Таблица 4.5 - Методы модуля Prey

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
Prey	Метод, запускающийся при создании объекта класса (конструктор)	public Prey(double x, double y, int dir, double wang, double ang, double speed, double size)	x	Координата x добычи
			y	Координата y
			dir	Направление движения добычи
			wang	Желаемый угол поворота добычи
			ang	Текущий угол поворота добычи
			speed	Скорость движения добычи

Продолжение таблицы 4.5

			Size	Размер добычи
getRadius	Возвращает радиус добычи	public double getRadius()	-	-

4.6 Описание модуля SnakeBodyPart

Модуль Snake представляет собой отдельные части тела змеи. Он используются для управления поведением и характеристиками змеи в игре.

Таблица 4.6 - Методы модуля SnakeBodyPart

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
SnakeBodyPart	Метод, запускающийся при создании объекта класса (конструктор)	public SnakeBodyPart(double x, double y)	x	Координата x части тела змеи
			y	Координата y части тела змеи

4.7 Описание модуля Food

Модуль Food представляет собой еду, которую змеи могут съесть в игре.

Таблица 4.7 - Методы модуля Food

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
getRadius	Возвращает радиус еды, который изменяется со временем.	public double getRadius()	-	-

Продолжение таблицы 4.7

Food	Метод, запускающийся при создании объекта класса (конструктор)	public Food(int x, int y, double size, boolean fastSpawn)	x	Координата x еды
			y	Координата y еды
			size	Размер еды
			fastSpawn	Флаг, указывающий, должна ли еда появляться быстро

4.8 Описание модуля Player

Модуль Player представляет абстракцию игрока в игре и определяет интерфейс для выполнения действий в игровой модели, включая направление движения и использование ускорения.

Таблица 4.8 - Методы модуля Player

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
Player	Метод, запускающийся при создании объекта класса (конструктор)	public Player(String name)	name	Имя игрока

Продолжение таблицы 4.8

action	Абстрактный метод, представляющий действие игрока. Метод должен быть реализован в подклассах.	public abstract Wish action(MySlitherModel model);	model	Модель игры
Wish	Метод, запускающийся при создании объекта класса (конструктор)	public Wish(Double angle, Boolean boost) {	angle	Угол направления движения
			boost	Флаг, указывающий, желает ли игрок использовать ускорение

4.9 Описание модуля MySlitherJFrame

Модуль MySlitherJFrame представляет собой главное окно приложения «MySlither.io» и управляет его интерфейсом и взаимодействием с пользователем..

Таблица 4.9 - Методы модуля MySlitherJFrame

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
MySlitherJFrame	Метод, запускающийся при создании объекта класса (конструктор)	public MySlitherJFrame() throws IOException	-	-

Продолжение таблицы 4.9

setStatus	Установливает текущего состояния приложения	private void setStatus(Status newStatus)	newStatus	Новое состояние приложения
log	Записиывет сообщений в лог-файл приложения	void log(String text)	text	Строка текста для записи в лог
print	Выводит текст в интерфейс приложения.	void print(String text)	text	Строка текста для вывода
onOpen	Обрабатывает действия, которые должны быть выполнены при подключении к серверу	void onOpen()	-	-
onClose	Обрабатывает действия, которые должны быть выполнены при отключении от сервера	void onClose()	-	-
connect	Запускает новый поток для выполнения операции подключения	public void connect()	-	-

Продолжение таблицы 4.9

trySingleConnect	Выполняет попытку подключения к серверу	void trySingleConnect()	-	-
disconnect	Обрабатывает процедуру разрыва соединения	public void disconnect()	-	-
setModel	Установливает игровой модели приложения	void setModel(MySlitherModel model)	model	Модель игры
setMap	Установливает карту игрового поля	void setMap(boolean[] map)	map	Массив булевых значений, представляющих состояние клеток поля
setRank	Установливает рейтинга игрока	void setRank(int newRank, int playerCount)	newRank	Новый рейтинг игрока
			playerCount	Общее количество игроков
setKills	Установливает количество убийств игрока	void setKills(int newKills)	newKills	Новое количество убийст
setHighScoreData	Установливает данные в таблице лидеров	void setHighScoreData(int row, String name, int length, boolean highlighted)	row	Номер строки
			name	Имя игрока
			length	Длина змеи

Продолжение таблицы 4.9

			highlighted	Флаг выделения строки
initColorMap	Инициализирует карту цветов для змеек	void initColorMap()	-	-
getColorName	Получает имя цвета	public String getColorName(Color color)	color	Представление в виде класса Color

4.10 Описание модуля MySlitherCanvas

Модуль MySlitherCanvas представляет полотно игры в виде панели, которая отображает графические элементы игрового мира.

Таблица 4.10- Методы модуля MySlitherCanvas

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
MySlitherCanvas	Метод, запускающийся при создании объекта класса (конструктор)	public MySlitherCanvas(MySlitherJFrame view)	view	Вид игры
setMap	Устанавливает мини-карту игрового поля	void setMap(boolean[] map)	-	-
paintComponent	Отрисовывает графические элементы игрового мира	public void paintComponent(Graphics graphics)	graphics	Контекст графики

4.11 Описание модуля MySlitherWebSocketClient

Модуль MySlitherWebSocketClient представляет собой клиент WebSocket для взаимодействия с сервером игры Slither.io. В нем реализованы методы для обработки различных типов сообщений, отправки данных на сервер и управления игровым процессом.

Таблица 4.11- Методы модуля MySlitherWebSocketClient

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
MySlitherWebSocketClient	Метод, запускающийся при создании объекта класса (конструктор)	public MySlitherWebSocketClient(URI serverUri, MySlitherJFrame view)	serverUri	URI сервера
			view	Вид игры
onOpen	Выполненяет действия при успешном подключении к серверу	public void onOpen(ServerHandshake handshake)	handshake	Рукопожатие сервера
onMessage	Обработывает текстовые сообщения от сервера	public void onMessage(String message)	message	Текст сообщения
onMessage	Обработывает бинарные данные от сервера	public void onMessage(ByteBuffer buffer)	buffer	Буфер байтов, содержащий данные сообщения от сервера

Продолжение таблицы 4.11

sendData	Отправляет данные серверу в соответствии с действиями игрока	void sendData(Player.Wish wish)	wish	Действие игрока
processKill	Обрабатывает сообщение о смерти игрока	private void processKill(int[] data)	data	Массив данных, содержащий информацию о сметри
processAddRemovePrey	Обрабатывает сообщения о добавлении или удалении добычи	private void processAddRemovePrey(int[] data)	data	Массив данных, содержащий информацию о добавлении или удалении добычи
onClose	Выполняет действия при отключении от серверу	public void onClose(int code, String reason, boolean remote)	code	Код закрытия
			reason	Причина закрытия
			remote	Флаг, указывающий, было ли закрытие инициировано удаленным хостом
onError	Выполняет действия при возникновении ошибка	public void onError(Exception ex)	ex	Объект типа Exception, представляющий возникшую ошибку

Продолжение таблицы 4.11

processUpdatePrey	Обрабатывает сообщение об обновлении состояния добычи	private void processUpdatePrey(int[] data)	Data	Массив данных, содержащий информацию об обновлении
processRemoveFood	Обрабатывает сообщение об удалении еды	private void processRemoveFood(int[] data)	data	Массив данных, содержащий информацию о координатах удаляемой еды
processAddFood	Обрабатывает сообщение о добавлении еды	private void processAddFood(int[] data, char cmd)	data	Массив данных, содержащий информацию о добавляемой еде
			cmd	Символ, указывающий на тип сообщения
processUpdateMinimap	Обрабатывает сообщение об обновлении миникарты	private void processUpdateMinimap(int[] data)	data	Массив данных, содержащий информацию о состоянии миникарты
processPong	Обрабатывает сообщение о получении ответа на пинг	private void processPong(int[] data)	data	Массив данных, содержащий информацию ответа сервера на пинг

Продолжение таблицы 4.11

processGlobalHighScore	Обрабатывает сообщение о глобальном рекорде дня	private void processGlobalHighScore(int[] data)	Data	Массив данных, содержащий информацию о рекорде
processRemoveSector	Обрабатывает сообщение об удалении сектора	private void processRemoveSector(int[] data)	data	Массив данных, содержащий информацию о координатах удаляемого сектора
processAddSector	Обрабатывает сообщение о добавлении сектора	private void processAddSector(int[] data)	data	Массив данных, содержащий информацию о координатах добавляемого сектора
processDead	Обрабатывает сообщение о смерти игрока	private void processDead(int[] data)	data	Массив данных, содержащий информацию о причине смерти
processLeaderboard	Обрабатывает сообщение о доске лидеров	private void processLeaderboard(int[] data)	data	Массив данных, содержащий информацию о лидерах

Продолжение таблицы 4.11

processRemoveSnakePart	Обрабатывает сообщение об удаление части тела змеи	private void processRemoveSnakePart(int[] data)	data	Массив данных, содержащий данные для обработки удаления
processUpdateFam	Обрабатывает сообщение об обновлении значение соотношение частей тела змеи	private void processUpdateFam (int[] data)	data	Массив данных, содержащий данные для обновления
processUpdateBodyparts	Обрабатывает сообщение об обновлении свойств частей тела змеи на основе заданной команды	private void processUpdateBodyparts(int[] data, char cmd)	data	Массив данных, содержащий данные для обновления частей тела
			cmd	Символьная команда, указывающая тип обновления
getNewAngle	преобразует угол в требуемый формат	private double getNewAngle(int angle)	angle	Угол
getNewSpeed	Преобразует скорость в требуемый формат	private double getNewSpeed(int speed)	speed	Скорость

Продолжение таблицы 4.11

processUpdateSnakePosition	Обрабатывает сообщение об обновлении положении змеи	private void processUpdateSnakePosition(int[] data, char cmd)	data	Массив данных, содержащий данные для обновления положения
			cmd	Символьная команда, указывающая тип обновления
processAddRemoveSnake	Обрабатывает сообщение об добавление или удаление змеи	private void processAddRemoveSnake(int[] data)	data	Массив данных, содержащий данные для добавления или удаления змеи
sendInitRequest	Отправляет запрос на инициализацию для установки имени пользователя и скина для змеи	void sendInitRequest(int snakeSkin, String nick)	snakeSkin	Идентификатор скина
			nick	Никнейм
processPreInitResponse	Обрабатывает ответ сервера на предварительную инициализацию	void processPreInitResponse(int[] data)	data	Массив данных, содержащий данные ответа
processInitResponse	Обрабатывает ответ сервера на инициализацию	void processInitResponse(int[] data)	data	Массив данных, содержащий данные ответа

Продолжение таблицы 4.11

decodeSecret	Декодирует секрет, предоставленный сервером	private static byte[] decodeSecret(int[] secret)	Secter	Массив данных, представляющий секрет
getServerList	получает список URI серверов	static URI[] getServerList()	-	-

4.12 Описание модуля MySlitherModel

Модуль MySlitherModel представляет собой представляет собой модель игры «Slither.io». Он хранит и управляет состоянием различных объектов в игре, а также содержит методы для манипуляции этими объектами и их атрибутами.

Таблица 4.12 - Методы модуля MySlitherModel

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
MySlitherWebSocketClient	Метод, запускающийся при создании объекта класса (конструктор)	public MySlitherModel(double spangdv, double nsp1, double nsp2, double nsp3, double mamu1, double mamu2, int gameRadius, int sectorSize, double cst, int mscps, MySlitherJFrame view)	spangdv	Скорость поворота змеи
			nsp1	Первый параметр скорости змеи
			nsp2	Второй параметр скорости змеи
			nsp3	Третий параметр скорости змеи

Продолжение таблицы 4.12

			mamu1	Базовая угловая скорость змеи
			mamu2	Угол на который может повернуться добыча
			gameRadius	Радиус игрового поля
			sectorSize	Размер сектора игрового поля
			cst	Соотношении скоростей хвоста змеи
			mscps	Максимальный размер длины тела змеи
			view	Вид игры
onOpen	Выполненяет действия при успешном подключении к серверу	public void removeSnake(int id)	handshake	Рукопожатие сервера
addSnake	Добавляет новую змею с заданными параметрами	public void addSnake(Snake snake)	snake	Объект типа Snake, который добавляется

Продолжение таблицы 4.12

getSnake	Возвращает змею с заданным идентификатором	Snake getSnake(int snakeID)	snakeId	Идентификатор змеи
getSnakeLength	Вычисляет длину змеи на основе длины тела и фактора заполнения.	int getSnakeLength(int bodyLength, double fillAmount)	bodyLength	Длина тела змеи
			fillAmount	Фактор заполнения змеи
getPrey	Возвращает добычу с заданным идентификатором	Prey getPrey(int id)	id	Идентификатор добычи
removePrey	Удаляет добычу с заданным идентификатором	void removePrey(int id)	id	Идентификатор добычи
addPrey	Добавляет новую добычу	void addPrey(int id, Prey prey){	id	Идентификатор добычи
			prey	Объект типа Prey, который добавляется
addSector	Добавляет сектор игрового поля	void addSector(int x, int y)	x	Координата x сектора
			y	Координата y сектора

Продолжение таблицы 4.12

removeSector	Удаляет сектор игрового поля и удаляет еду в этом секторе.	public void removeSector(int x, int y)	x	Координата x сектора
			y	Координата y сектора
addFood	Добавляет еду на игровое поле	public void addFood(int x, int y, double size, boolean isFastSpawn)	x	Координата x еды
			y	Координата y еды
			size	Размер еды
			isFastSpawn	Быстрое появление еды
removeFood	Удаляет еду с заданными координатами	public void removeFood(int x, int y)	x	Координата x еды
			y	Координата y еды

5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

В данном разделе я остановлюсь на тестировании программного средства. В таблицах 5.1, 5.2, 5.3 и 5.4 представлены результаты тестирования программного средства.

5.1 Тестирование и проверка работоспособности кнопок

Таблица 5.1 - Тестирование и проверка работоспособности кнопок

№	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Установка URI сервера для игры вручную	Запустить приложение, ввести URI сервера игры вручную	Ввод сервера в соответствующее текстовое поле	Тест пройден
2	Установка желаемо ника змеи для игры	Запустить приложение, ввести желаемы ник для змеи	Ввод ника в соответствующее текстовое поле	Тест пройден
3	Выбор цвета тела змеи из предоставленного списка	Запустить приложение, открыть выпадающий список с возможными цветами тела змеи	Выбор желаемого цвета тела змеи	Тест пройден
4	Установка флага для выбора случайного сервера	Запустить приложение, поставить флаг для выбора случайного сервера	Установка флага для выбора случайного сервера для игры	Тест пройден
5	Подключение к оригинальному игровому серверу	Запустить приложение, нажать на кнопку «Connect»	Подключение к оригинальному игровому серверу	Тест пройден

Продолжение таблицы 5.1

6	Отключение от оригинального игрового сервера	Запустить приложение, нажать на кнопку «Connect», после того как клиент подключиться к серверу нажать на кнопку «Disconnect»	Отключение от оригинального игрового сервера	Тест пройден

5.2 Тестирование управления игровым процессом

Таблица 5.2 - Тестирование управления игровым процессом

№	**Тестируемая функциональность**	**Последовательность действий**	**Ожидаемый результат**	**Полученный результат**
1	Следование змеи за курсором мыши	Запустить приложение, нажать кнопку «Connect», после подключения к серверу начать водить курсором мыши по игровому полю	Змея следует за курсором мыши	Тест пройден
2	Ускорение змеи при нажатии на левую кнопку мыши	Запустить приложение, нажать кнопку «Connect», после подключения к серверу нажать на левую кнопку мыши	Змея начала ускоряться	Тест пройден

5.3 Тестирование игровой логики

Таблица 5.3 - Тестирование игровой логики

№	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Появление змеи на экране, когда она попадает в активные секторы	Запустить приложение, нажать кнопку «Connect»	Змеи появляются на экране, когда попадают в активные секторы	Тест пройден
2	Появление еды на экране, когда она попадает в активные секторы	Запустить приложение, нажать кнопку «Connect»	Еда появляется на экране, когда попадает в активные секторы	Тест пройден
3	Появление добычи на экране, когда она попадает в активные секторы	Запустить приложение, нажать кнопку «Connect»	Добыча появляется на экране, когда попадает в активные секторы	Тест пройден
4	Удаление змеи с экрана, когда она выходит за активные секторы	Запустить приложение, нажать кнопку «Connect»	Змеи удаляется с экрана, когда выходит в активные секторы	Тест пройден
5	Удаление еды с экрана, когда она выходит за активные секторы	Запустить приложение, нажать кнопку «Connect»	Еда удаляется с экрана, когда выходит в активные секторы	Тест пройден
6	Удаление добычи с экрана, когда она выходит за активные секторы	Запустить приложение, нажать кнопку «Connect»	Добыча удаляется с экрана, когда выходит в активные секторы	Тест пройден

Продолжение таблицы 5.3

7	Заканчивание игры после того, как змея столкнулась с другой змеей	Запустить приложение, нажать кнопку «Connect», столкнуться с другой змеей	Игра оканчивается, змея игрока удаляется с экрана, на ее месте появляется еда	Тест пройден

5.4 Итоги тестирования

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

Для корректной работы программы необходимо распаковать архив в удобную вам папку, чтобы начать использовать программное средство, необходимо запустить файл MySlither.io.jar.

После открытия программы появляется окно с игровым полем (рис. 6.1)

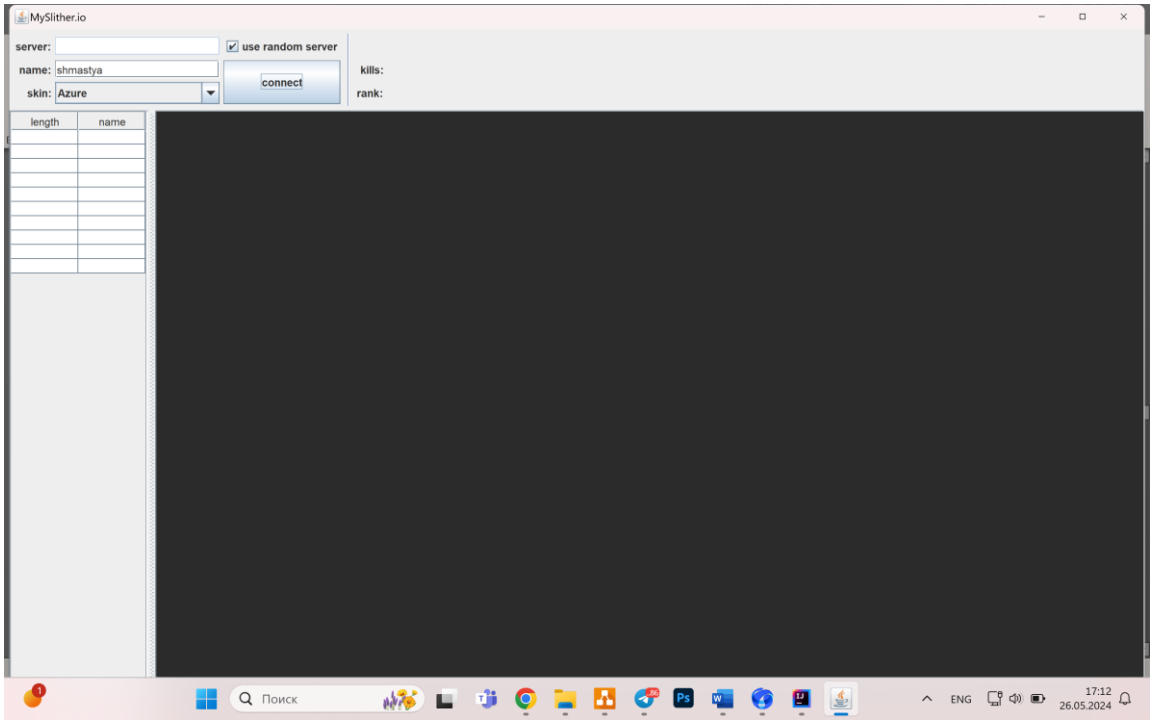

Рисунок 6.1 – Игровое поле

У пользователя появляется возможность самому ввести желаемый сервер, или поставить флаг для выбора случайного сервера, ник, который будет отображаться при игре, а также цвет тела змеи. Далее надо нажать на кнопку «Connect», после чего будет проведена попытка подключиться либо к введенному вручную серверу, либо к случайному. После удачного подключения на экране начнет отображаться игровое поле со змеей, которой будет управлять игрок, с змеями других игроков, едой и добычами (рис 6.2)

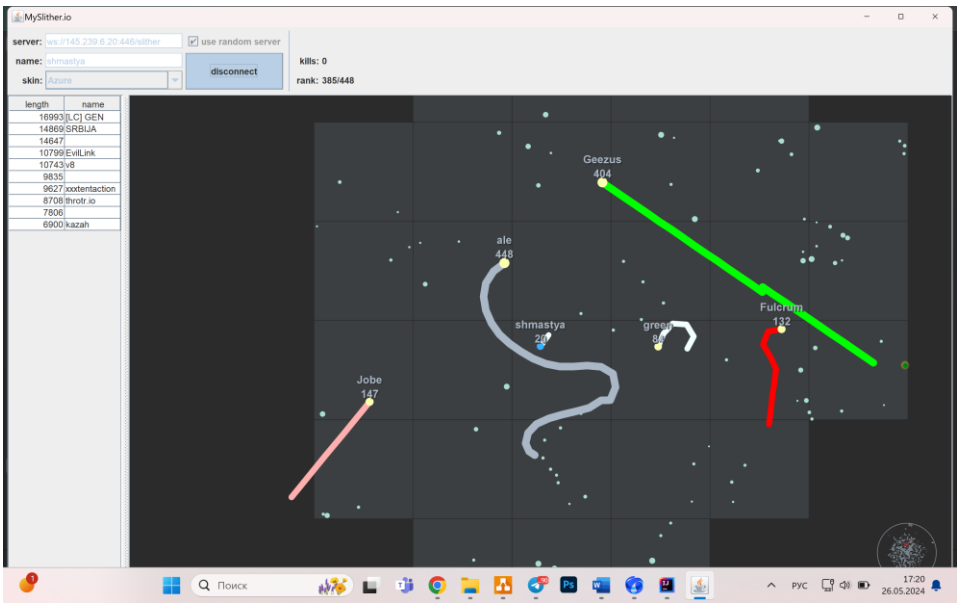

Рисунок 6.2 – Игровое поле

Теперь игрок может управлять змеёй с помощью курсора мыши, змея будет следовать за ним. Также можно ускорить змею при нажатии на левую кнопку мыши. Можно наблюдать за рейтингом игроков на сервере с помощью таблицы, которая находится слева от игрового поля, на ней отображается первые 10 самых длинных змей на сервере, если игрок будет в этом топе, то его ник будет подсвечиваться цветом. А сверху над игровым полем, игрок может наблюдать за своим рейтингом из всем игроков и количеству змей, которые разбились о тело змеи игрока.

Если игрок захочет окончить игру, то он может нажать на кнопку «Disconnect», после чего произойдет отключение от сервера (рис 6.3)

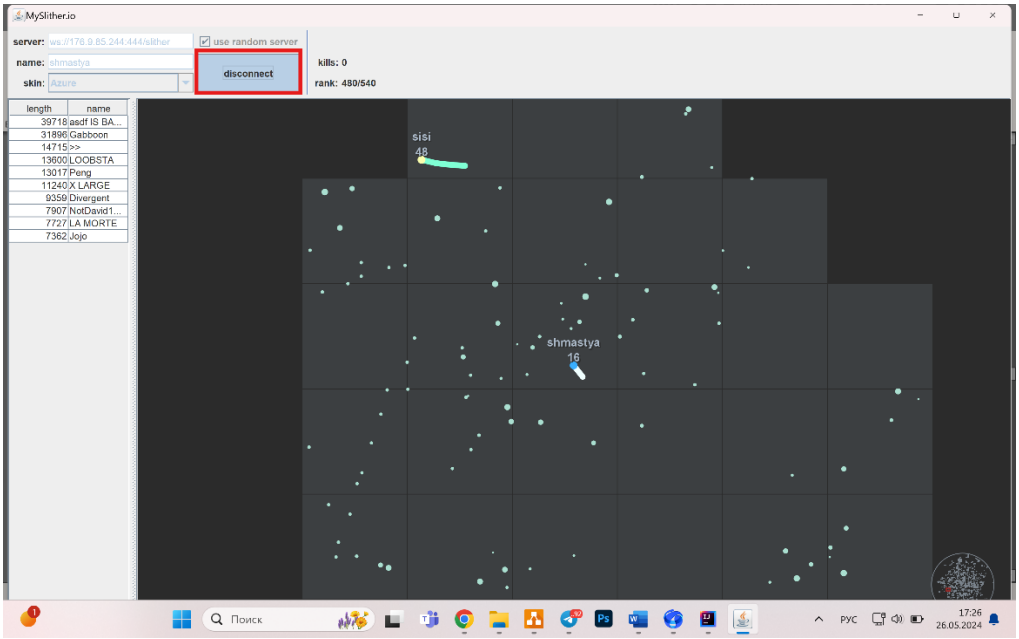

Рисунок 6.3 – Кнопка «Disconnect»

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги и выявлены их преимущества и недостатки. В качестве языка разработки использовался Java. На этапе проектирования были разработаны блок-схемы алгоритмов. В ходе работы был получен опыт работы со сторонними графическими библиотеками, в частности Swing.

В ходе данной работы был изучен протокол, по которому общается оригинальный сервер игры с клиентом, было создано программное средство «MySlither.io», которое предоставляет собой клиент для этой всемирно известной игры. И теперь пользователю предоставляется приятное времяпрепровождение, которое может помочь развивать мозг и реакцию, так как «MySlither.io», это тактическая игра, в которой также понадобиться навым быстрого реагирования в сложных ситуациях .

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список использованной литературы

[1] https://ru.wikipedia.org/wiki/Slither.io

[2] https://slitherio.fandom.com/wiki/Slither.io_Wiki

[3]https://ru.wikipedia.org/wiki/%D0%97%D0%BC%D0%B5%D0%B9%D0%BA%D0%B0_(%D0%B8%D0%B3%D1%80%D0%B0)

[4] https://github.com/ClitherProject/Slither.io-Protocol/tree/master

Приложение А

Код класса MySlitherWebSocketClient.java

```
package org.example;

import org.java_websocket.client.WebSocketClient;
import org.java_websocket.drafts.Draft_6455;
import org.java_websocket.handshake.ServerHandshake;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.ByteBuffer;
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;

import static org.example.MySlitherModel.PI2;

public class MySlitherWebSocketClient extends WebSocketClient {

    private static final byte[] DATA_PING = new byte[]{(byte) 251};
    private static final byte[] DATA_BOOST_START = new byte[]{(byte) 253};
    private static final byte[] DATA_BOOST_STOP = new byte[]{(byte) 254};
    private static final double ANGLE_CONSTANT = 16777215;
    private static final Map<String, String> HEADER = new LinkedHashMap<>();

    private final MySlitherJFrame view;
    private MySlitherModel model;

    private byte[] initRequest;
    private long lastAngleTime, lastPingTime;
    private byte lastAngleContent, angleToBeSent;
    private boolean lastBoostContent;
    private boolean waitingForPong;


    static {
        HEADER.put("Origin", "http://slither.io");
        HEADER.put("Pragma", "no-cache");
        HEADER.put("Cache-Control", "no-cache");
    }

    public MySlitherWebSocketClient(URI serverUri, MySlitherJFrame view) {
        super(serverUri, new Draft_6455(), HEADER);
        this.view = view;
    }

    @Override
    public void onOpen(ServerHandshake handshake) {
        view.log("connected: " + handshake.getHttpStatusMessage());
        view.onOpen();
    }
```

```
@Override
public void onMessage(String message) {
    view.log("message: " + message);
}

@Override
public void onMessage(ByteBuffer buffer) {
    byte[] b = buffer.array();
    if (b.length < 3) {
        view.log("too short");
        return;
    }
    int[] data = new int[b.length];
    for (int i = 0; i < b.length; i++) {
        data[i] = b[i] & 0xFF;
    }
    char cmd = (char) data[2];
    switch (cmd) {
        case '6' -> processPreInitResponse(data);
        case 'a' -> processInitResponse(data);
        case 'e', 'E', '3', '4', '5' -> processUpdateBodyparts(data, cmd);
        case 'h' -> processUpdateFam(data);
        case 'r' -> processRemoveSnakePart(data);
        case 'g', 'G', 'n', 'N' -> processUpdateSnakePosition(data, cmd);
        case 'l' -> processLeaderboard(data);
        case 'v' -> processDead(data);
        case 'w' -> processRemoveSector(data);
        case 'W' -> processAddSector(data);
        case 'm' -> processGlobalHighScore(data);
        case 'p' -> processPong(data);
        case 'u' -> processUpdateMinimap(data);
        case 's' -> processAddRemoveSnake(data);
        case 'F', 'b', 'f' -> processAddFood(data, cmd);
        case 'c' -> processRemoveFood(data);
        case 'j' -> processUpdatePrey(data);
        case 'y' -> processAddRemovePrey(data);
        case 'k' -> processKill(data);
    }
}

void sendData(Player.Wish wish){
    if (wish.angle != null){
        angleToBeSent = (byte) (wish.angle * 251 / PI2);
    }
    if (angleToBeSent != lastAngleContent && System.currentTimeMillis()-lastAngleTime>100){
        lastAngleTime = System.currentTimeMillis();
        lastAngleContent = angleToBeSent;
        send(new byte[]{angleToBeSent});
    }

    if (wish.boost != null && wish.boost != lastBoostContent){
        lastBoostContent = wish.boost;
        send(wish.boost ? DATA_BOOST_START : DATA_BOOST_STOP);
    }

    if (!waitingForPong && System.currentTimeMillis() - lastPingTime > 250){
        lastPingTime = System.currentTimeMillis();
        waitingForPong = true;
        send(DATA_PING);
```

```
    }
}

private void processKill(int[] data) {
    if (data.length != 8) {
        view.log("kill wrong length!");
        return;
    }

    int killerId = (data[3] << 8) | (data[4]);
    int kills = (data[5] << 16) |(data[6] << 8) | (data[7]);

    if (killerId == model.snake.id) {
        view.setKills(kills);
    } else {
        view.log("kill packet with invalid id: " + killerId);
    }
}

private void processAddRemovePrey(int[] data) {
    if (data.length == 5) {
        int preyId = (data[3] << 8) | (data[4]);
        model.removePrey(preyId);
    } else if (data.length == 7) {
        int preyId = (data[3] << 8) | (data[4]);
        int eatenId = (data[5] << 8) | (data[6]);
        model.removePrey(preyId);
    } else if (data.length==22){
        int preyId = (data[3] << 8) | (data[4]);
        int colorId = data[5];
        double x = ((data[6] << 16) |(data[7] << 8) | (data[8]))/5.0;
        double y = ((data[9] << 16) |(data[10] << 8) | (data[11]))/5.0;
        double size = data[12]/5.0;
        int direction = data[13]-48;
        double wang = ((data[14] << 16) |(data[15] << 8) | (data[16]))*PI2/ANGLE_CONSTANT;
        double ang = ((data[17] << 16) |(data[18] << 8) | (data[19]))*PI2/ANGLE_CONSTANT;
        double speed = ((data[20] << 8) | (data[21]))/1000.0;
        model.addPrey(preyId, new Prey(x, y, direction, wang, ang, speed, size));
    } else {
        view.log("add/remove prey wrong length!");
    }
}
@Override
public void onClose(int code, String reason, boolean remote) {
    view.log("closed: " + code + ", " + remote + ", " + reason);
    view.onClose();
}

@Override
public void onError(Exception ex) {
    view.log("ERROR: " + ex);
    ex.printStackTrace();
}

private void processUpdatePrey(int[] data) {
    if (data.length != 11 && data.length != 12 && data.length != 13 && data.length != 14 &&
            data.length != 15 && data.length != 16 && data.length != 18) {
        view.log("update prey wrong length!");
        return;
    }
```

```
    int preyId = (data[3] << 8) | data[4];
    int x = ((data[5] << 8) | data[6]) * 3 + 1;
    int y = ((data[7] << 8) | data[8]) * 3 + 1;

    synchronized (view.modelLock) {
        Prey prey = model.getPrey(preyId);
        prey.x = x;
        prey.y = y;

        switch (data.length) {
            case 11 -> prey.speed = ((data[9] << 8) | data[10]) / 1000.0;
            case 12 -> prey.ang = ((data[9] << 16) | (data[10] << 8) | data[11]) * PI2 / ANGLE_CONSTANT;
            case 13 -> {
                prey.dir = data[9] - 48;
                prey.wang = ((data[10] << 16) | (data[11] << 8) | data[12]) * PI2 / ANGLE_CONSTANT;
            }
            case 14 -> {
                prey.ang = ((data[9] << 16) | (data[10] << 8) | data[11]) * PI2 / ANGLE_CONSTANT;
                prey.speed = ((data[12] << 8) | data[13]) / 1000.0;
            }
            case 15 -> {
                prey.dir = data[9] - 48;
                prey.wang = ((data[10] << 16) | (data[11] << 8) | data[12]) * PI2 / ANGLE_CONSTANT;
                prey.speed = ((data[13] << 8) | data[14]) / 1000.0;
            }
            case 16 -> {
                prey.dir = data[9] - 48;
                prey.ang = ((data[10] << 16) | (data[11] << 8) | data[12]) * PI2 / ANGLE_CONSTANT;
                prey.wang = ((data[13] << 16) | (data[14] << 8) | data[15]) * PI2 / ANGLE_CONSTANT;
            }
            case 18 -> {
                prey.dir = data[9] - 48;
                prey.ang = ((data[10] << 16) | (data[11] << 8) | data[12]) * PI2 / ANGLE_CONSTANT;
                prey.wang = ((data[13] << 16) | (data[14] << 8) | data[15]) * PI2 / ANGLE_CONSTANT;
                prey.speed = ((data[16] << 8) | data[17]) / 1000.0;
            }
        }
    }
}

private void processRemoveFood(int[] data) {
    if (data.length != 7 && data.length != 9) {
        view.log("remove food wrong length!");
        return;
    }

    int x = (data[3] << 8) | data[4];
    int y = (data[5] << 8) | data[6];

    model.removeFood(x, y);
}

private void processAddFood(int[] data, char cmd) {
    boolean isAllowMultipleEntries = cmd == 'F';
    boolean isFastSpawn = cmd != 'f';

    if ((!isAllowMultipleEntries && data.length != 9)
            || (isAllowMultipleEntries && data.length < 9)
            || ((data.length - 9) % 6 != 0)) {
        view.log("add food wrong length!");
```

```
            return;
        }

        for (int i = 8; i < data.length; i += 6) {
            int x = (data[i - 4] << 8) | data[i - 3];
            int y = (data[i - 2] << 8) | data[i - 1];
            double size = data[i] / 5.0;
            model.addFood(x, y, size, isFastSpawn);
        }
    }

    private void processUpdateMinimap(int[] data) {
        boolean[] map = new boolean[80 * 80];
        int mapPos = 0;
        for (int dataPos = 3; dataPos < data.length; dataPos++) {
            int value = data[dataPos];
            if (value >= 128) {
                value -= 128;
                mapPos += value;
                if (mapPos >= map.length) {
                    break;
                }
            } else {
                for (int i = 0; i < 7; i++) {
                    if ((value & (1 << (6 - i))) != 0) {
                        map[mapPos] = true;
                    }
                    mapPos++;
                    if (mapPos >= map.length) {
                        break;
                    }
                }
            }
        }
        view.setMap(map);    }

    private void processPong(int[] data) {
        if (data.length != 3) {
            view.log("pong wrong length!");
            return;
        }

        waitingForPong = false;
    }

    private void processGlobalHighScore(int[] data) {
        if (data.length < 10) {
            view.log("add sector wrong length!");
            return;
        }

        int bodyLength = (data[3] << 16) | (data[4] << 8) | data[5];
        double fam = ((data[6] << 16) | (data[7] << 8) | data[8]) / ANGLE_CONSTANT;

        int nameLength = data[9];
        StringBuilder name = new StringBuilder(nameLength);
        for (int i = 0; i < nameLength; i++) {
            name.append((char) data[10 + i]);
        }
```

```
    StringBuilder message = new StringBuilder();
    for (int i = 0; i < data.length - 10 - nameLength; i++) {
        message.append((char) data[10 + nameLength + i]);
    }

    view.log("Received Highscore of the day: " + name + " (" +
            model.getSnakeLength(bodyLength, fam) + "): " + message);
}

private void processRemoveSector(int[] data) {
    if (data.length != 5) {
        view.log("add sector wrong length!");
        return;
    }
    int sectorX = data[3];
    int sectorY = data[4];
    model.removeSector(sectorX, sectorY);
}

private void processAddSector(int[] data) {
    if (data.length != 5) {
        view.log("remove sector wrong length!");
        return;
    }
    int sectorX = data[3];
    int sectorY = data[4];
    model.addSector(sectorX, sectorY);
}

private void processDead(int[] data) {
    if (data.length != 4) {
        view.log("dead wrong length!");
        return;
    }
    int deathReason = data[3];
    switch (deathReason) {
        case 0 -> view.log("You died.");
        case 1 -> view.log("You've achieved a new record!");
        case 2 -> view.log("Death reason 2, unknown");
        default -> view.log("invalid death reason: " + deathReason + "!");
    }
}

private void processLeaderboard(int[] data) {
    if (data.length < 8 + 10 * 7) {
        view.log("leaderboard wrong length!");
        return;
    }

    int ownRank = (data[4] << 8) | (data[5]);
    int playersCount = (data[6] << 8) | (data[7]);

    view.setRank(ownRank, playersCount);

    int rank = 0;
    int cursorPosition = 8;
    while (cursorPosition + 6 < data.length) {
        int bodyLength = (data[cursorPosition] << 8) | (data[cursorPosition + 1]);
        double bodyPartFam = ((data[cursorPosition + 2] << 16) | (data[cursorPosition + 3] << 8)
                | (data[cursorPosition + 4])) / ANGLE_CONSTANT;
```

```
            int nameLength = data[cursorPosition + 6];
            StringBuilder name = new StringBuilder(nameLength);
            for (int i = 0; i < nameLength && cursorPosition + 7 + i < data.length; i++) {
                name.append((char) data[cursorPosition + 7 + i]);
            }
            rank++;
            cursorPosition += nameLength + 7;
            view.setHighScoreData(rank-1, name.toString(), model.getSnakeLength(bodyLength, bodyPartFam),
                    ownRank == rank);
        }
        view.log("leaderboard is set");
    }

    private void processRemoveSnakePart(int[] data) {
        if (data.length != 8 && data.length != 5) {
            view.log("remove snake part wrong length!");
            return;
        }
        int snakeId = (data[3] << 8) | (data[4]);
        synchronized (view.modelLock) {
            Snake snake = model.getSnake(snakeId);
            if (data.length == 8) {
                snake.setFam(((data[5] << 16) | (data[6] << 8) | (data[7])) / ANGLE_CONSTANT);
            }
            snake.body.pollLast();
        }
    }

    private void processUpdateFam(int[] data) {
        if (data.length != 8) {
            view.log("update fam wrong length!");
            return;
        }
        int snakeId = (data[3] << 8) | (data[4]);
        synchronized (view.modelLock) {
            Snake snake = model.getSnake(snakeId);
            snake.setFam(((data[5] << 16) | (data[6] << 8) | (data[7])) / ANGLE_CONSTANT);
            view.log("newFam " + snake.getFam());
        }
    }

    private void processUpdateBodyparts(int[] data, char cmd) {
        if (data.length != 8 && data.length != 7 && data.length != 6) {
            view.log("update body-parts wrong length!");
            return;
        }

        int snakeId = (data[3] << 8) | (data[4]);
        int newDir = -1;
        double newAng = -1;
        double newWang = -1;
        double newSpeed = -1;

        if (data.length == 8) {
            newDir = cmd == 'e' ? 1 : 2;

            newAng = getNewAngle(data[5]);
            newWang = getNewAngle(data[6]);
            newSpeed = getNewSpeed(data[7]);
```

```
} else if (data.length == 7) {
   switch (cmd) {
      case 'e':
         newAng = getNewAngle(data[5]);
         newSpeed = getNewSpeed(data[6]);
         break;
      case 'E':
         newDir = 1;
         newWang = getNewAngle(data[5]);
         newSpeed = getNewSpeed(data[6]);
      case '3':
         newDir = 1;
         newAng = getNewAngle(data[5]);
         newWang = getNewAngle(data[6]);
      case '4':
         newDir = 2;
         newWang = getNewAngle(data[5]);
         newSpeed = getNewSpeed(data[6]);
      case '5':
         newDir = 2;
         newAng = getNewAngle(data[5]);
         newWang = getNewAngle(data[6]);
      default:
         view.log("update body-parts invalid cmd/length: " + cmd + ", " + data.length);
         return;
   }
} else if (data.length == 6) {
   switch (cmd) {
      case 'e':
         newAng = getNewAngle(data[5]);
         break;
      case 'E':
         newDir = 1;
         newWang = getNewAngle(data[5]);
      case '3':
         newSpeed = getNewSpeed(data[5]);
      case '4':
         newDir = 2;
         newWang = getNewAngle(data[5]);
      case '5':
         newSpeed = getNewSpeed(data[5]);
      default:
         view.log("update body-parts invalid cmd/length: " + cmd + ", " + data.length);
         return;
   }
}

synchronized (view.modelLock) {
   Snake snake = model.getSnake(snakeId);

   if (newDir != -1) {
      snake.dir = newDir;
   }
   if (newAng != -1) {
      snake.ang = newAng;
   }
   if (newWang != -1) {
      snake.wang = newWang;
   }
   if (newSpeed != -1) {
```

```
            snake.speed = newSpeed;
        }
    }
}

private double getNewAngle(int angle) {
    return angle * PI2 / 256;
}

private double getNewSpeed(int speed) {
    return speed / 18.0;
}

private void processUpdateSnakePosition(int[] data, char cmd) {
    boolean isAbsoluteCoordinates = (cmd == 'g' || cmd == 'n');
    boolean isNewBodyPart = (cmd == 'n' || cmd == 'N');

    if (data.length != 5 + (isAbsoluteCoordinates ? 4 : 2) + (isNewBodyPart ? 3 : 0)) {
        view.log("update snake body wrong length!");
        return;
    }

    int snakeId = (data[3] << 8) | (data[4]);

    synchronized (view.modelLock) {
        Snake snake = model.getSnake(snakeId);
        SnakeBodyPart head = snake.body.getFirst();

        double newX = isAbsoluteCoordinates ? ((data[5] << 8) | (data[6])) : (data[5] - 128 + head.x);
        double newY = isAbsoluteCoordinates ? ((data[7] << 8) | (data[8])) : (data[6] - 128 + head.y);

        if (isNewBodyPart) {
            snake.setFam(((data[isAbsoluteCoordinates ? 9 : 7] << 16)
                    | (data[isAbsoluteCoordinates ? 10 : 8] << 8)
                    | data[isAbsoluteCoordinates ? 11 : 9]) / ANGLE_CONSTANT);
        } else {
            snake.body.pollLast();
        }

        snake.body.addFirst(new SnakeBodyPart(newX, newY));

        snake.x = newX;
        snake.y = newY;

        if (isNewBodyPart)
            view.log("new bode part was add");
    }
}

private void processAddRemoveSnake(int[] data) {
    if (data.length == 6) {
        int id = (data[3] << 8) | (data[4]);
        model.removeSnake(id);
        view.log("add snake with id " + id);
    } else if (data.length >= 34) {
        int snakeId = (data[3] << 8) | (data[4]);

        double ang = ((data[5] << 16) | (data[6] >> 8) | data[7]) * PI2 / ANGLE_CONSTANT;
        double wang = ((data[9] << 16) | (data[10] >> 8) | data[11]) * PI2 / ANGLE_CONSTANT;
```

```
            double speed = ((data[12] << 8) | (data[13])) / 1000.0;
            //Snake last body part fullness
            double fam = ((data[14] << 16) | (data[15] >> 8) | data[16]) / ANGLE_CONSTANT;

            int skin = data[17];

            double x = ((data[18] << 16) | (data[19] << 8) | (data[20])) / 5.0;
            double y = ((data[21] << 16) | (data[22] << 8) | (data[23])) / 5.0;

            int nameLength = data[24];
            StringBuilder name = new StringBuilder(nameLength);
            for (int i = 0; i < nameLength; i++) {
               name.append((char) data[25 + i]);
            }

            int customSkinDataLength = data[25 + nameLength];
            StringBuilder customSkinDataName = new StringBuilder(customSkinDataLength);
            for (int i = 0; i < customSkinDataLength; i++) {
               customSkinDataName.append((char) data[25 + nameLength + i]);
            }

            double currentBodyPartX = ((data[26 + nameLength + customSkinDataLength] << 16)
                  | (data[27 + nameLength + customSkinDataLength] << 8)
                  | (data[28 + nameLength + customSkinDataLength])) / 5.0;
            double currentBodyPartY = ((data[29 + nameLength + customSkinDataLength] << 16)
                  | (data[30 + nameLength + customSkinDataLength] << 8)
                  | (data[21 + nameLength + customSkinDataLength])) / 5.0;

            Deque<SnakeBodyPart> body = new ArrayDeque<>();
            body.addFirst(new SnakeBodyPart(currentBodyPartX, currentBodyPartY));

            for (int nextSnakeBodyPart = 32 + nameLength + customSkinDataLength;
                 nextSnakeBodyPart + 1 < data.length; nextSnakeBodyPart += 2) {
               currentBodyPartX += (data[nextSnakeBodyPart] - 127) / 2.0;
               currentBodyPartY += (data[nextSnakeBodyPart + 1] - 127) / 2.0;
               body.addFirst(new SnakeBodyPart(currentBodyPartX, currentBodyPartY));
            }

            Snake snake = new Snake(snakeId, name.toString(),
view.SNAKES.get(skin),x,y,wang,ang,speed,fam,body,model);
            model.addSnake(snake);
         } else {
            view.log("add/remove snake wrong length!");
         }
      }

      void sendInitRequest(int snakeSkin, String nick) {
         // Set username and skin
         initRequest = new byte[4 + nick.length()];
         //packet-type (here 115 = 's')
         initRequest[0] = 115;
         //protocol version
         initRequest[1] = 10;
         // skin ID
         initRequest[2] = (byte) snakeSkin;
         // nick length
         initRequest[3] = (byte) nick.length();
         // nick bytes
         for (int i = 0; i < nick.length(); i++) {
            initRequest[i + 4] = (byte) nick.codePointAt(i);
```

```
        }

        //pre-init response
        view.log("sending pre-init request");
        //This is the first packet sent. The server will then respond with the Pre-init response.
        send(new byte[]{99});
    }

    void processPreInitResponse(int[] data) {
        view.log("sending decrypted, manipulated secret");
        send(decodeSecret(data));

        view.log("send init-request");
        send(initRequest);
    }

    void processInitResponse(int[] data) {
        if (data.length != 26) {
            view.log("init response wrong length!");
            return;
        }
        int gameRadius = (data[3] << 16) | (data[4] << 8) | (data[5]);
        //get mscps (maximum snake length in body parts units)
        int mscps = (data[6] << 8) | data[7];
        int sectorSize = (data[8] << 8) | data[9];
        // get spangdv (value / 10) (coef. to calculate angular speed change depending snake speed)
        double spangdv = data[12] / 10.0;
        // nsp1 (value / 100) (Maybe nsp stands for "node speed"?)
        double nsp1 = ((data[13] << 8) | data[14]) / 100.0;
        double nsp2 = ((data[15] << 8) | data[16]) / 100.0;
        double nsp3 = ((data[17] << 8) | data[18]) / 100.0;
        // get mamu (value / 1E3) (basic snake angular speed)
        double mamu1 = ((data[19] << 8) | data[20]) / 1000.0;
        //manu2 (value / 1E3) (angle in rad per 8ms at which prey can turn)
        double mamu2 = ((data[21] << 8) | data[22]) / 1000.0;
        //sct is a snake body parts count (length) taking values between [2 .. mscps].
        //fpsls[mscps] contains snake volume (score) to snake length in body parts units. 1/fmlts[mscps] contains
        // body part volume (score) to certain snake length.
        double cst = ((data[23] << 8) | data[24]) / 1000.0;
        int protocolVersion = data[25];

        if (protocolVersion != 11) {
            view.log("wrong protocol version (" + protocolVersion + ")");
            return;
        }

        model = new MySlitherModel(spangdv, nsp1, nsp2, nsp3, mamu1, mamu2, gameRadius, sectorSize, cst, mscps,
view);
        view.setModel(model);
        view.setKills(0);
    }

    private static byte[] decodeSecret(int[] secret) {

        byte[] result = new byte[24];

        int globalValue = 0;
        for (int i = 0; i < 24; i++) {
            int value1 = secret[17 + i * 2];
            if (value1 <= 96) {
```

```
            value1 += 32;
          }
          value1 = (value1 - 98 - i * 34) % 26;
          if (value1 < 0) {
            value1 += 26;
          }

          int value2 = secret[18 + i * 2];
          if (value2 <= 96) {
            value2 += 32;
          }
          value2 = (value2 - 115 - i * 34) % 26;
          if (value2 < 0) {
            value2 += 26;
          }

          int interimResult = (value1 << 4) | value2;
          int offset = interimResult >= 97 ? 97 : 65;
          interimResult -= offset;
          if (i == 0) {
            globalValue = 2 + interimResult;
          }
          result[i] = (byte) ((interimResult + globalValue) % 26 + offset);
          globalValue += 3 + interimResult;
        }

        return result;
      }

      static URI[] getServerList() {
        String i33628_String;
        try {
          HttpURLConnection i49526_HttpURLConnection = (HttpURLConnection)
                new URI("http://slither.io/i33628.txt").toURL().openConnection();
          i49526_HttpURLConnection.setRequestProperty("User-Agent", "java/1.8.0_72");
          InputStream i49526_InputStream = i49526_HttpURLConnection.getInputStream();
          BufferedReader i49526_BufferedReader = new BufferedReader(new
InputStreamReader(i49526_InputStream));
          i33628_String = i49526_BufferedReader.lines().collect(Collectors.joining("\n"));
        } catch (IOException | URISyntaxException ex) {
          throw new Error("Error reading server-list!");
        }

        int[] data = new int[(i33628_String.length() - 1) / 2];
        for (int i = 0; i < data.length; i++) {
          int u1 = (i33628_String.codePointAt(i * 2 + 1) - 97 - 14 * i) % 26;
          if (u1 < 0) {
            u1 += 26;
          }
          int u2 = (i33628_String.codePointAt(i * 2 + 2) - 104 - 14 * i) % 26;
          if (u2 < 0) {
            u2 += 26;
          }
          data[i] = (u1 << 4) + u2;
        }

        URI[] serverList = new URI[(i33628_String.length() - 1) / 22];
        for (int i = 0; i < serverList.length; i++) {
          try {
            serverList[i] = new URI("ws://"
```

```
                    + data[i * 11 + 0] + "."
                    + data[i * 11 + 1] + "."
                    + data[i * 11 + 2] + "."
                    + data[i * 11 + 3] + ":"
                    + ((data[i * 11 + 4] << 16) + (data[i * 11 + 5] << 8) + data[i * 11 + 6])
                    + "/slither");
            } catch (URISyntaxException ex) {
                throw new Error("Error building server-address!");
            }
        }
        return serverList;
    }
}
```

Код класса MySlitherModel.java

```
package org.example;

import java.awt.*;
import java.util.Deque;
import java.util.LinkedHashMap;
import java.util.Map;

public class MySlitherModel {
    static final double PI2 = Math.PI * 2;

    final double spangdv;
    final double nsp1, nsp2, nsp3;
    private final double mamu1, mamu2;
    final int gameRadius;
    final int sectorSize;
    private final double cst;
    private final int mscps;
    private final double[] fpsls, fmlts;

    final boolean[][] sectors;
    private long lastUpdateTime;
    private final MySlitherJFrame view;

    Snake snake;

    Map<Integer, Snake> snakes = new LinkedHashMap<>();
    Map<Integer, Prey> preys = new LinkedHashMap<>();
    Map<Integer, Food> foods = new LinkedHashMap<>();

    public MySlitherModel(double spangdv, double nsp1, double nsp2, double nsp3, double mamu1, double mamu2,
                          int gameRadius, int sectorSize, double cst, int mscps, MySlitherJFrame view) {
        this.gameRadius = gameRadius;
        this.sectorSize = sectorSize;

        this.spangdv = spangdv;
        this.nsp1 = nsp1;
        this.nsp2 = nsp2;
        this.nsp3 = nsp3;
        this.mamu1 = mamu1;
        this.mamu2 = mamu2;
        this.cst = cst;
        this.mscps = mscps;
        this.view = view;

        sectors = new boolean[gameRadius * 2 / sectorSize][gameRadius * 2 / sectorSize];
```

```
    fmlts = new double[mscps + 1];
    fpsls = new double[mscps + 1];
    for (int i = 0; i < mscps; i++) {
        double base = (double) (mscps - i) / mscps;
        fmlts[i] = 1 / (base * base * Math.sqrt(Math.sqrt(base)));
        fpsls[i + 1] = fpsls[i] + fmlts[i];
    }

    lastUpdateTime = System.currentTimeMillis();

}

public void removeSnake(int id) {
    synchronized (view.modelLock) {
        snakes.remove(id);
    }
}

public void addSnake(Snake snake) {
    synchronized (view.modelLock) {
        Snake newSnake = snake;
        if (this.snake == null) {
            this.snake = newSnake;
        }
        snakes.put(newSnake.id, newSnake);
    }
}

Snake getSnake(int snakeID) {
    return snakes.get(snakeID);
}

int getSnakeLength(int bodyLength, double fillAmount) {
    bodyLength = Math.min(bodyLength, mscps);
    return (int) (15 * (fpsls[bodyLength] + fillAmount * fmlts[bodyLength]) - 20);
}
Prey getPrey(int id) {
    return preys.get(id);
}

void removePrey(int id){
    synchronized (view.modelLock){
        preys.remove(id);
    }
}

void addPrey(int id, Prey prey){
    synchronized (view.modelLock){
        preys.put(id, prey);
    }
}

void addSector(int x, int y) {
    synchronized (view.modelLock) {
        sectors[y][x] = true;
    }
}

public void removeSector(int x, int y) {
```

```
        synchronized (view.modelLock) {
            sectors[y][x] = false;
            foods.values().removeIf(food -> food.x / sectorSize == x && food.y / sectorSize == y);
        }
    }

    public void addFood(int x, int y, double size, boolean isFastSpawn) {
        synchronized (view.modelLock) {
            foods.put((y * gameRadius * 3) + x, new Food(x, y, size, isFastSpawn));
        }
    }

    public void removeFood(int x, int y){
        synchronized (view.modelLock) {
            foods.remove((y * gameRadius * 3) + x);
        }
    }
}
```