Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Базовая кафедра «Интеллектуальные системы управления»

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
« _____» _____ 20 ___ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

27.04.03 «Системный анализ и управление»
27.04.03.02 «Системный анализ данных и технологий принятия решений»

«Сохранение игрового баланса при помощи идентификации лавинообразных процессов»

Научный руководитель	_____	_____	А.А.Даничев
	подпись, дата	должность, ученая степень	
Выпускник	_____		Г.В. Никониров
	подпись, дата		
Рецензент	_____	_____	_____
	подпись, дата	должность, ученая степень	инициалы, фамилия
Нормоконтролер	_____	_____	_____
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2019

РЕФЕРАТ

Выпускная квалификационная работа по теме «Сохранение игрового баланса при помощи идентификации лавинообразных процессов» содержит 66 страниц текстового документа, 19 используемых источников, 71 иллюстрацию, 7 формул, 1 таблицу.

ГРАФ, ИГРА, СЮЖЕТ, БАЛАНС, ЛАВИНА, ПРОЦЕСС, НЕПАРАМЕТРИЧЕСКАЯ РЕГРЕССИЯ, АЛГОРИТМ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ИДЕНТИФИКАЦИЯ.

Целью данной работы является обеспечение сохранения игрового баланса в автоматическом режиме.

Актуальность данной работы проявляется в экономической выгоде разработчиков игр от сохранения баланса в игре.

Объектом исследования является модель игрового мира.

Предметом исследования является баланс в игровом мире.

Задачами исследования являются:

- разработать правила игрового мира, в рамках которого будет производится тестирование;
- сформулировать критерии оценки игрового баланса;
- разработать алгоритмы сохранения игрового баланса на основе: порогового значения, генетического алгоритма, идентификации лавинных процессов;
- разработать программное обеспечение (ПО) для моделирования;
- сравнить различные подходы для решения проблемы игрового баланса.

Апробация системы для моделирования была произведена при помощи использования разработанной системы в рамках анализа стабильности работы сейсмических станций в КГБУ «ЦРМПиООС».

В результате проведения данной работы была изучена литература по данной теме, проведены исследования, выбран и модифицирован алгоритм.

По итогу алгоритм показал хорошую работоспособность и применимость для балансировки игрового мира.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире есть огромное количество способов провести свое время, как с пользой, так и без нее. И одним из самых древних форм времяпрепровождения являются игры [1]. Еще в древние времена люди играли в различные игры, за счет этого они постигали мир и получали знания от других поколений (в те времена игры сводились к воображаемой охоте на добычу).

Сейчас существует множество других способов передачи знаний, поэтому игры чаще всего являются средством времяпрепровождения для развлечений, конечно, не стоит забывать о всяких играх для развития детей. Однако, более взрослые люди предпочитают играть в игры не ради саморазвития, а ради забавы.

Поэтому сейчас активно развиваются компьютерные игры [2]. Эта область, кажущаяся детской на первый взгляд, на деле является хорошо развитым направлением бизнеса, сейчас никого не удивляет купить виртуальный ключ от игры за 2000 рублей. Однако, существуют и другие способы получения дохода от индустрии компьютерных игр. Одним из них являются платные подписки — это такой способ предоставления доступа к игре, в котором игрок платит за время, проведенное в игре (как правило, абонентская подписка на определенный срок, чаще всего кратный месяцу). Еще один не менее популярный среди производителей игр — это внутриигровые покупки или пожертвования. Они позволяют купить за реальные деньги виртуальные ценности, например, деньги или одежду для игрового аватара (персонаж, которым управляет игрок).

Однако, не важно как хороша компьютерная игра, в ней, как правило, есть недостатки. Почти все игры имеют один общий — генерация игрового мира. В основном проблема проявляется в однообразии игровых локаций (мест, которые посещает игровой персонаж) или их несбалансированности (локация может содержать только слабых противников или, наоборот, только сильных). Для решения первой проблемы есть специальное направление в индустрии «геймдизайн» [3] — это направление, которое занимается созданием игрового контента по пожеланиям игроков. Вторую проблему обычно решает тоже направление, но чем больше расширяется сообщество игры, тем сложнее становится поддерживать баланс. Основные проблемы начинаются при резком увеличении или уменьшении количества игроков. **Целью** данной работы является обеспечение сохранения игрового баланса в автоматическом режиме.

Для этого необходимо выполнить следующие задачи:

- разработать правила игрового мира, в рамках которого будет производится тестирование;
- сформулировать критерии оценки игрового баланса;
- разработать алгоритмы сохранения игрового баланса на основе: порогового значения, генетического алгоритма, идентификации лавинных процессов;

- разработать программное обеспечение (ПО) для моделирования;
- сравнить различные подходы для решения проблемы игрового баланса.

Для реализации ПО планируется использовать язык программирования Go [4], который является кроссплатформенным и, который позволяет запускать программы, написанные на нем, почти на любых устройствах. Для работы с исходными текстами программы планируется применять систему контроля версий Git [5] — это программа для контроля изменений в программном коде. Для хранения данных экспериментов планируется использовать реляционную базу данных PostgreSQL [6], которая является проектом с открытым программным кодом, и поддерживаемая большим сообществом разработчиков. Для написания исходных текстов программы была выбрана среда разработки (текстовый редактор, предназначенный для написания программ) Atom [7].

Апробация среды моделирования была произведена в КГБУ «ЦРМПиООС» в рамках расчета стабильности и эффективности работы сейсмических станций.

В рамках исследования была написана и опубликована статья [8]. В статье была рассмотрена проблема игрового баланса, и способ ее устранения при помощи идентификации лавинных процессов.

1 Среда моделирования

Для каждой модели или алгоритма лучший способ убедиться в его работоспособности — это испытать в реальных условиях. Для алгоритмов, представляющих возможности к численному моделированию, испытания можно производить на реальных данных, передавая их алгоритму. В нашем случае, поскольку алгоритм будет численным, возможны его многократные тесты на сгенерированных данных. Для этой работы будет использоваться выдуманная игра (подробнее в разделе 1.1), которая будет обладать всеми интересующими нас свойствами — лавинность, инертность. Поскольку игра будет выдуманной, получить большие объемы данных для моделирования не составит труда. Однако, перед тем, как приступать к моделированию, необходимо разработать правила игры и программу для моделирования.

1.1 Разработка игровых правил

Изначально планировалось создать правила для игроков, монстров и деревень (поселений с союзными для игроков неигровыми персонажами — NPC — not player character). При этом планировалось 4 вида монстров, 4 типа персонажа, которыми можно управлять. Планировалось, что монстры будут голодать и, чтобы насытиться, им необходимо будет поедать персонажей игроков (героев) или уничтожать деревни. В свою очередь героям необходимы деревни, чтобы восстанавливать свои силы (здоровье и т.д.). Герои и монстры стремились бы к развитию — уничтожению представителей противоположной фракции, это бы повышало их характеристики.

Однако, многообразие видов монстров и типов героев не позволяет эффективно оценивать работу алгоритма, вследствие чего монстры и герои больше не имеют типов. Также деревни и механика сытости избыточно усложняет алгоритмы генерации монстров, поэтому из-за необходимости в простой тестовой модели от механики сытости пришлось отказаться, а деревни были заменены на крепости (крепость не исцеляет героев, а только атакует монстров). Также для сохранения баланса была изменена механика регенерации здоровья, теперь регенерация происходит постоянно. Это позволит снизить размерность входных параметров модели.

В итоге игровые правила приняли следующий вид. Игра будет представлена сильно упрощенным фэнтезийным миром. В этом мире будет всего лишь 3 вида сущностей: герои, монстры и крепости.

У героев есть следующие скрытые характеристики (находятся внутри модели, не подвержены изменениям извне):

- максимальное здоровье (натуральное число);
- текущее здоровье (натуральное число);
- восстановление здоровья за ход (натуральное число);
- урон (сколько здоровья герой отнимает у противника за ход, натуральное число);

- радиус атаки (насколько далеко герой может атаковать, натуральное число);

- подвижность (какое расстояние герой может преодолеть за ход, натуральное число).

У монстров есть следующие скрытые характеристики:

- максимальное здоровье (натуральное число);

- текущее здоровье (натуральное число);

- восстановление здоровья за ход (натуральное число);

- урон (сколько здоровья монстр отнимает у противника за ход, натуральное число);

- радиус атаки (насколько далеко монстр может атаковать, натуральное число);

- подвижность (какое расстояние монстр может преодолеть за ход, натуральное число).

У крепостей есть следующие скрытые характеристики:

- урон (сколько здоровья крепость снимает монстру за ход, натуральное число);

- радиус атаки (насколько далеко крепость может атаковать, натуральное число);

- время существования (сколько ходов стоит крепость, целое число).

Действия сущностей будут описываться следующим списком (за ход сущность может совершить только одно действие из списка, чем выше действие, тем больший приоритет оно имеет), где *CONST1*, *CONST2*, *CONST3* задаются в начале моделирования и являются натуральными числами:

а) герой, если у него меньше *CONST1* % здоровья, ищет ближайшую крепость и идет к ней;

б) монстр, если у него меньше *CONST2* % здоровья, направляется в противоположную сторону от ближайшего сильного героя;

в) герой каждый ход выбирает случайного монстра в радиусе атаки и наносит ему урон;

г) если герой убивает монстра, то его характеристики изменяются следующим образом:

1) текущее и максимальное здоровье увеличиваются на величину МАКСИМАЛЬНОЕ_ЗДОРОВЬЕ_МОНСТРА * *CONST3*;

2) регенерация здоровья увеличиваются на величину РЕГЕНЕРАЦИЯ_ЗДОРОВЬЯ_МОНСТРА * *CONST3*;

3) урон увеличивается на величину УРОН_МОНСТРА * *CONST3*;

д) крепость каждый ход выбирает случайного монстра в радиусе атаки и наносит ему урон;

е) если крепость убивает монстра, то характеристики ближайшего героя изменяются следующим образом:

1) текущее и максимальное здоровье увеличиваются на величину МАКСИМАЛЬНОЕ_ЗДОРОВЬЕ_МОНСТРА * *CONST3*;

2) регенерация здоровья увеличиваются на величину

РЕГЕНЕРАЦИЯ_ЗДОРОВЬЯ_МОНСТРА * *CONST3*;

3) урон увеличивается на величину УРОН_МОНСТРА * *CONST3*;

ж) монстр каждый ход выбирает случайного героя в радиусе атаки и наносит ему урон;

з) если монстр убивает героя, то его характеристики изменяются следующим образом:

1) текущее и максимальное здоровье увеличиваются на величину МАКСИМАЛЬНОЕ_ЗДОРОВЬЕ_ГЕРОЯ * *CONST3*;

2) регенерация здоровья увеличиваются на величину РЕГЕНЕРАЦИЯ_ЗДОРОВЬЯ_ГЕРОЯ * *CONST3*;

3) урон увеличивается на величину УРОН_ГЕРОЯ * *CONST3*;

и) герой, если у него больше *CONST1* % здоровья, ищет ближайшего монстра, которого он может одолеть, и идет к нему;

к) монстр, если у него больше *CONST2* % здоровья, ищет ближайшего героя, которого он может победить, и идет к нему.

При перемещениях сущности не учитывают прочие факторы кроме своей цели, таким образом, возможны ситуации, где герой при движении в сторону крепости наткнется на сильного монстра, или монстр при движении от героя наткнется на крепость. Также герои при движении в сторону крепости не учитывают сколько ходов она простоит, таким образом, они могут прийти к крепости, которая разрушится на следующий ход.

1.2 Прототип программного обеспечения

Для моделирования необходимо ПО обладающее следующими возможностями:

- работа с произвольными структурами данных;
- сохранение результатов моделирования;
- перемоделировать эксперимент с произвольного момента;
- визуализация результатов экспериментов, как серий, так и отдельных;
- динамический сбор метрик, возможность их пересчета, возможность добавления произвольных метрик.

Поскольку среди доступного ПО не было найдено ни одного удовлетворяющего требованиям, было принято решение о разработке специализированного ПО.

ПО для моделирования в данном случае представляет собой систему, состоящую из следующих элементов:

- граф с данными — там хранятся данные модели;
- обработчик событий — он производит моделирование и выполняет операции над данными модели;
- обработчик патчей — он выполняет изменение данных, а также их сохранение для возможности воспроизведения поведения модели.

Для начала опишем само хранение данных. Данные хранятся в графе, при этом граф имеет произвольную степень связанности. Узлы графа хранят в себе

данные, при этом есть как обязательные данные (присутствуют во всех узлах), так и произвольные. Обязательные данные — это номер узла (его уникальный идентификатор) и тип узла (тип может быть произвольным, например, территория или человек, а также тип может быть специальным: событие, процесс). Дуги в графе хранят данные о связи, имеют в себе следующие данные: тип, уникальный идентификатор.

Существуют два специальных типа узла: событие и процесс. Тип процесс отвечает за хранение данных некого действия, что производится в модели (действие должно быть атомарным), например, выпадение осадков — это процесс. У процесса есть время начала его исполнения, которое измеряется в единицах атомарного времени (например, в ходах). Тип событие отвечает за хранение данных некоторых продолжительных действий, которые могут быть разделены на процессы, например, событие дождь сопровождается процессами выпадения осадков.

Для обработки данных используется обработчик событий. Он производит операции над данными с помощью заранее заданных функций для выбора того, над какими данными производить операцию, используются специальные узлы с типом процесса. Они описывают какие параметры должна использовать функция, в том числе и то, какие данные использовать (другие узлы и дуги). Результатом вызова функции обработчиком является патч (каждая функция создает свой). Патч представляет из себя набор данных об изменениях модели (создание, удаление узла дуги, изменение данных в узле, а также создание завершения новых процессов). При этом патч — это технический элемент (не является частью модели) и в графе он не хранится.

Уточним что такое патч — это некая совокупность изменений модели за некоторый период. Патчи создаются в результате выполнения процессов в модели, при этом, до того как патч будет применен на модель, данные в ней не будут изменены.

Поскольку патчи производятся процессами, то в каждую единицу атомарного времени происходит выполнение всех процессов, что начинаются в это время, с них собираются патчи, после чего они соединяются в общий патч за эту единицу времени. После чего общий патч сохраняется в историю (для воспроизводимости действий), а потом применяется на данные модели.

Также хотелось бы упомянуть преимущества, которые дает система патчей:

- воспроизводимость — после моделирования мы можем воспроизвести состояние модели в конкретные моменты времени;
- легкий анализ изменений — поскольку патч, это совокупность изменений, то возможно использовать информацию из патчей без полного воспроизведения модели для построения различных оценок и проведения анализа;
- проверка предположений — при помощи патчей можно достаточно легко проверить высказывания вида «что, если сделать это (тут имеется в виду некоторое изменение модели) не так», поскольку мы может получить модель в

любой момент времени и провести моделирование из этого состояние другим методом (совокупностью действий, которые могут происходить в модели).

1.3 Модификация программного обеспечения

После реализации прототипа программы [9], были выявлены следующие недостатки: избыточность данных для дуг, усложнение системы из-за раздельной работы с процессами и событиями.

Первая проблема проявлялась в неудобстве работы с дугами, как отдельной структурой данных:

- структура для хранения дуг (имеется в виду совокупность структуры контроллера дуг и самих объектов дуг) занимала 17% оперативной памяти, выделенной приложению (для сравнения, узлы занимали только 59%, при этом количество данных (переменных) в узле в 4-7 раз больше, чем в дуге);
- для получения связанных узлов (не обязательно одной дугой) требовалось выполнять дорогостоящие по процессорному времени операции (алгоритмы обхода графа требовали частых запросов к оперативной памяти, что создавало дополнительную нагрузку на процессор);
- неудобный для разработки интерфейс объекта дуги, имея всего-лишь несколько полей, он нуждался в сложных функциях и дополнительном контроллере (это приводило к трате дополнительных человеко-часов на разработку ПО);
- редкое использование объектов дуг — терялось их основное теоретическое преимущество — работа с однообразными по структуре (программной) связями в графе;
- нецелесообразное использование объектов дуг в ПО, дуги заменялись более удобными полями в объектах узлов графа.

Для ее решения были рассмотрены следующие варианты:

- переход от отдельных объектов для дуг на общий объект-контроллер;
- перенос данных о дугах в объекты узлов.

Первый вариант решил проблемы 1 и 2, но проблемы 3-5 остались нерешенными. Второй вариант [10] позволил решить все проблемы. Однако, следует заметить, что второй вариант не отказывается от использования дуг, просто теперь данные о дуге будут храниться в объектах узла, которые она связывает.

Вторую проблему — усложнение системы из-за раздельной работы с процессами и событиями, было решено устранить отказом от типа события и введением периодических процессов [11]. То есть, теперь процессы и события объединены и представляют собой некоторую инструкцию (алгоритм), которая может выполняться неоднократно, при этом инструкция должна выполняться за атомарную единицу времени (за 1 ход). Если требуется выполнить сложную инструкцию с неатомарным временем, то она разбивается на цепь инструкций с условными переходами. Таким образом, процесс выполняется за атомарную

единицу времени, может породить дочерний процесс (цепь инструкций), а также запускаться циклично.

1.4 Возможности приложения

Было реализовано приложение, позволяющее моделировать и анализировать процесс размещения сущностей при управлении игровым миром (Рисунок 1.1).

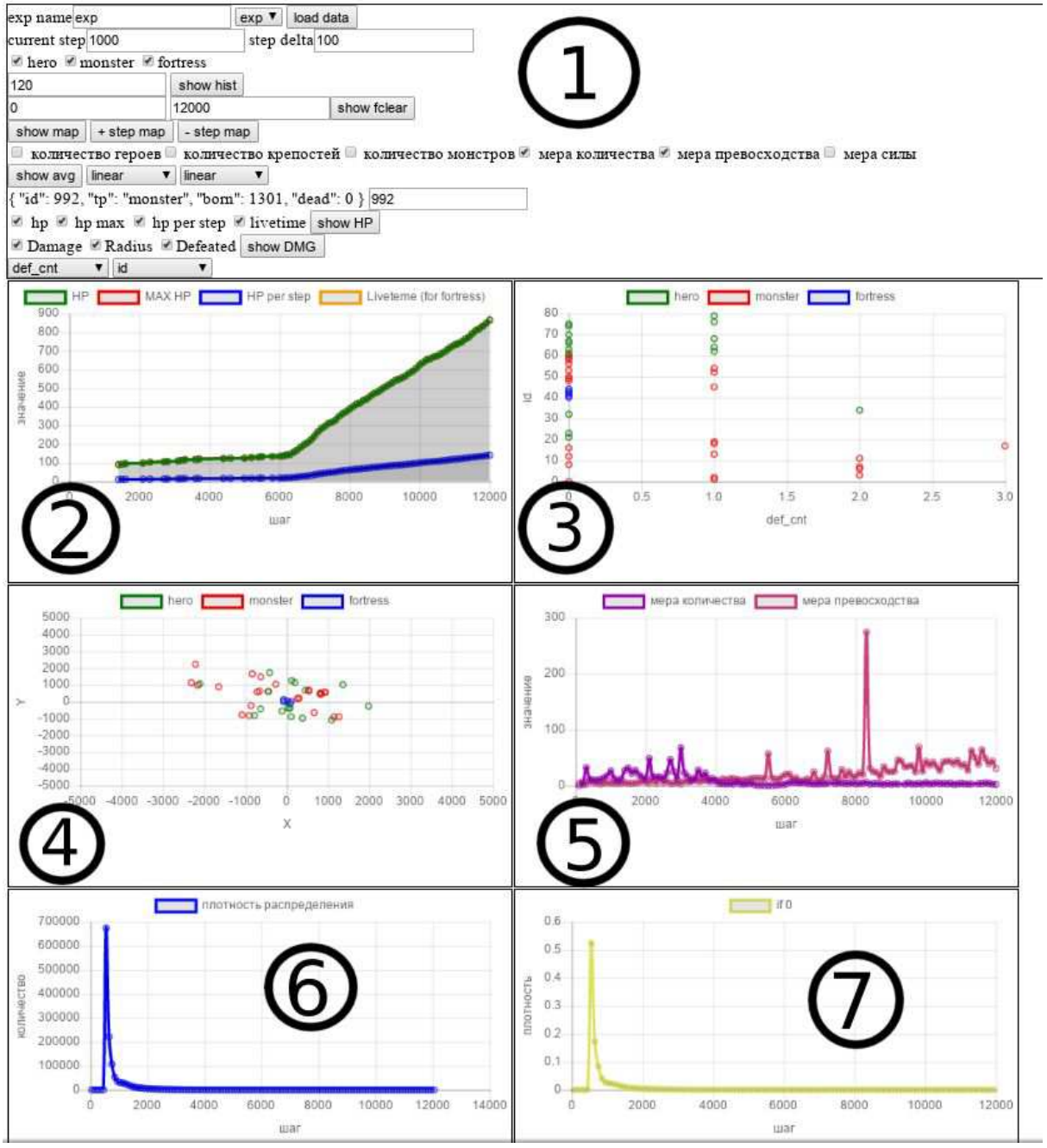

Рисунок 1.1 — Приложение

На данный момент в приложении возможно:

а) провести моделирование данных со следующими параметрами:

1) количество шагов моделирования;
2) тип моделирования: случайный (параметры задаются случайно), фиксированный (параметры заданы заранее);

б) провести визуализацию метрик игрового мира (выбор настроек обозначен цифрой 1 на рисунке 1.1):

1) снапшот игровой карты с расположением сущностей (обозначен цифрой 4 на рисунке 1.1);
2) корреляционное поле характеристик сущностей (обозначено цифрой 3 на рисунке 1.1);
3) график изменений характеристики (например, здоровье или атака) сущности (обозначен цифрой 2 на рисунке 1.1);
4) график изменений средних и количественных данных для всего игрового мира (среднее здоровье героев и т.д., обозначен цифрой 5 на рисунке 1.1);
5) распределение количества неудачных экспериментов текущей серии (обозначена цифрой 6 на рисунке 1.1);
6) плотности неудачных экспериментов по сериям (обозначено цифрой 7 на рисунке 1.1).

1.5 Использование приложения в практической задаче

Само приложение, благодаря его гибкости, позволило с небольшими изменениями выполнить практическую задачу, отличную от моделирования игрового мира.

Была поставлена следующая задача: при помощи приложения узнать коэффициенты стабильности работы сейсмостанций Красноярского края, а также эффективность их использования.

Для исходных данных были взяты каталоги с зарегистрированными землетрясениями, и каталоги технических сбоев на станции. На основе вероятностного распределения землетрясений (координаты, время, интенсивность) и частотного распределения сбоев (периодичность, продолжительность) была настроена среда моделирования.

В результате тестирования среда моделирования показала незначительные отличия от реальных данных, которые являются допустимыми. Также системой был составлен расчет стабильности и эффективности работы сейсмостанций на 2019 год.

2 Модель

После описания среды для моделирования, необходимо описать модель, при помощи которой будет произведен анализ и управление средой.

2.1 Описание модели

Представим объект управления в виде черного ящика (рисунок 2.1)

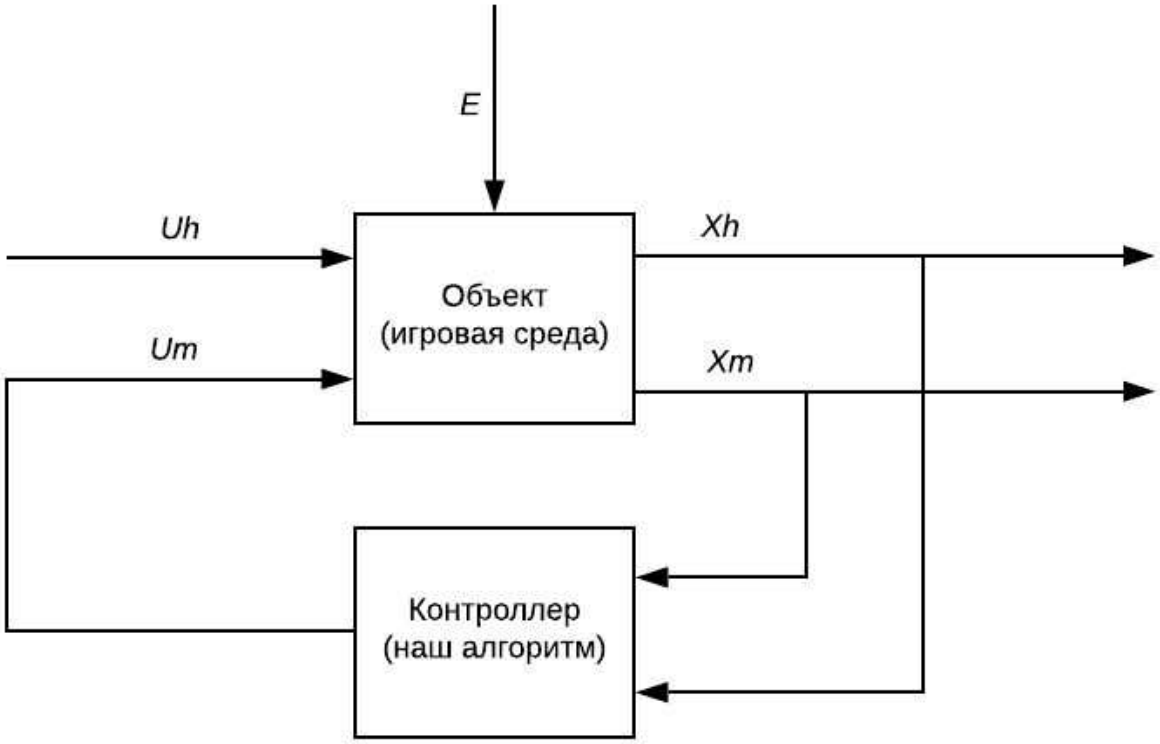

Рисунок 2.1 — Объект управления

На рисунке приняты следующие обозначения. *Uh* — вектор управляющего воздействия героев, где Uh_{count} количество генерируемых героев за раз, Uh_{period} период генерации. *Um* — вектор управляющего воздействия монстров, где Um_{count} количество генерируемых монстров за раз, Um_{period} период генерации. *E* — вектор помехи, где E_{count} количество генерируемых крепостей за раз, E_{period} период генерации (помеха распределена по нормальному закону). *Xh* — матрица выходных данных (герои), где i номер героя, j его характеристика (hp — максимальное здоровье, dmg — текущий урон). Например, $Xh_{1,hp}$ максимальное здоровье 1 героя. *Xm* — матрица выходных данных (монстры), где i номер монстра, j его характеристика (hp — максимальное здоровье, dmg — текущий урон).

2.2 Критерии игрового баланса

Основная проблема — наличие избыточной силы у одной из сторон конфликта (игровой дисбаланс). Дисбаланс может выражаться по разному, например:

- количеством — у одной из сторон больше численность;
- силой — у одной из сторон более сильные войска;
- превосходством — у одной из сторон слабейшие силы, намного слабее сильнейших сил другой стороны.

В рамках данного исследования мы возьмем упрощенную модель, в которой мы сможем управлять появлением только 1 фракции.

Для определения дисбаланса мы будем использовать следующие формулы:

$$Fc = \frac{N_{Xh}}{N_{Xm}} + \frac{N_{Xm}}{N_{Xh}} - 2\,, \tag{2.1}$$

где Fc — метрика количества;

N_{Xh} — количество строк в матрице Xh;

N_{Xm} — количество строк в матрице Xm.

$$Fp = \frac{M[Xh_{hp}]}{M[Xm_{hp}]} + \frac{M[Xm_{hp}]}{M[Xh_{hp}]} + \frac{M[Xh_{dmg}]}{M[Xm_{dmg}]} + \frac{M[Xm_{dmg}]}{M[Xh_{dmg}]} - 4\,, \tag{2.2}$$

где Fp — метрика силы;

$M[]$ — оценка математического ожидания.

$$Fd = \frac{Max[Xh_{hp}]}{Min[Xm_{hp}]} + \frac{Max[Xm_{hp}]}{Min[Xh_{hp}]} + \frac{Max[Xh_{dmg}]}{Min[Xm_{dmg}]} + \frac{Max[Xm_{dmg}]}{Min[Xh_{dmg}]} - 4\,, \tag{2.3}$$

где Fd — метрика превосходства.

Для метрик по оси Y откладывается перевес в % (1-100%), при этом следует учитывать следующие:

а) для метрики превосходства параметр указывает 2-х кратный перевес, т.е. 2 - 100%;

б) для метрики силы параметр указывает 2-х кратный перевес, т.е. 2 — 100%;

в) метрики силы и превосходства являются условно оценочными, т.е.

1) их перевес не строго 2-х кратный (он варьируется);

2) они показывают более детальное состояние мира, т.е. по ним можно понять, что происходит в мире.

Для метрики количества критическое превышение — это 300% (происходит отсечение генерации — остановка эксперимента), когда количество одной фракции больше количества другой более чем в 3 раза.

Для метрики превосходства нет критического превышения (остановки не будет вне зависимости от размера превышения), но при этом рекомендованное значение не более 10 для простой генерации, и 100 для генерации с поглощением.

Для метрики силы нет критического превышения (остановки не будет вне зависимости от размера превышения), но при этом рекомендованное значение не более 2 для простой генерации, и 10 для генерации с поглощением.

2.3 Блок управления с генетическим алгоритмом

Для блока управления будем использовать генетический алгоритм [12]. Однако, в отличии от большинства генетический алгоритмов [13], оценка эффективности особи будет производиться по результатам поведения в основной системе, а также при определенных условиях — критическом состоянии системы, будет производиться смена особи. Критическое состояние системы, это когда в системе в течении нескольких шагов метрика количества превышает значение 2 (200% превышение численности фракции).

В самом алгоритме для упрощения мы будет использовать только два гена для особи: плотность генерации монстров и интенсивность генерации монстров.

Для начала опишем особенности популяции. У всех особей есть время жизни, после того как была выбрана особь для управления системой, время ее жизни продлевается, а, если точнее, после того, как система перестала управляться особью, начинается отсчет времени до смерти особи, если система опять использовала эту особь, то таймер сбросится.

Опишем подробнее процесс размножения и выбора особи для управления системой. Периодически алгоритм порождает одну новую особь (селекция, мутация и кроссинговер будут описаны позднее), затем эта особь начинает управлять системой. При этом особь, которая управляла до этого, активирует свой таймер до смерти. Также, для особи, которая использовалась, считается ее эффективность (насколько система хорошо работала при этой особи), и время ее жизни рассчитывается из этого параметра.

Селекция [14] производится очень просто, выбираются 2 лучшие по эффективности особи.

Кроссинговер [14] тоже достаточно прост, по каждому гену особи считается среднее арифметическое ее родителей.

Мутация [14] представляет собой изменение одного случайного гена. Изменение происходит по следующей формуле:

$$G' = G * (0.75 + R), \tag{2.4}$$

где G — первоначальное значение гена;

G' — новое значение гена;

R — случайное нормальное число в диапазоне от 0 до 0,5.

Отдельно стоит описать случай, когда система находится в критическом состоянии. Если окажется, что система находится в критическом состоянии, тогда производится мутация текущей заполняющей особи (если особь уже мутировала до этого, производится повторная мутация), если на момент следующей проверки (через 10 ходов — примерно 8 часов реального времени) система не улучшает свое состояние (снижение значения оценки количества, не обязательно до не критичного уровня), текущая особь умерщвляется, а на ее место приходит особь с лучшей оценкой работы из популяции. Если после этих действий критическое состояние только ухудшается, то действия повторяются (текущая особь мутирует и тд).

2.4 Лавинный процесс

Лавинный процесс — это резкое изменение выходного значения системы при малом изменении входного значения системы [15].

В случае нашей системы, это может быть резкое увеличение или снижение популяции монстров при малом оттоке или притоке игроков.

Хоть лавинный процесс и является неочевидной реакцией на входное воздействие, он поддается идентификации благодаря внутренним параметрам, а также выходным параметрам, которые не затрагивает процесс, при условии наличия у системы более одного выхода.

2.5 Блок управления с пороговым алгоритмом

Для блока управления будем использовать пороговый алгоритм с условным оператором. Во время своей работы, при достижении определенного значения формулы (2.5), интенсивность генерации новых особей, пересчитывается по формуле (2.6).

$$Sc_i = \left| \frac{\Delta F c_i / \Delta t}{\Delta F c_{i-1} / \Delta t} - 1 \right| = \left| \frac{\Delta F c_i}{\Delta F c_{i-1}} - 1 \right| , \quad (2.5)$$

где $\Delta F c_i$ — прирост метрики количества на i шаге;

$\Delta F c_{i-1}$ — прирост метрики количества на i-1 шаге;

Δt — длина шага.

$$Mi' = Mi * (a * \frac{Hc}{Mc} + b) , \quad (2.6)$$

где Mc — количество монстров на текущий момент;

Hc — количество героев на текущий момент;

Mi — интенсивность генерации монстров на текущий момент;

a, b — коэффициенты инертности (вычисляются эмпирическим путем, в данном случае были взяты a=0.8, b=0.2).

3 Практическая часть

После составления модели проведем серию экспериментов для выяснения способа обнаруживать и устранять лавины.

Для каждого метода будет проводиться 2 серии экспериментов, с поглощением и без него. Поглощение — это когда существо при своей победе получает развитие в зависимости от сил противника. Соответственно, при генерации с поглощением игровые сущности могут развиваться и становиться сильнее, без поглощения не могут, их уровень силы не меняется. Поглощение усиливает системную инертность, что приближает систему к реальной массовой игре (онлайн от 100 000 игроков на сервере одновременно). Отсутствие поглощения уменьшает инертность, при этом усиливается влияние выбросов, что приближает систему к играм с малым онлайном (онлайн до 100 000, в среднем 1 000 - 10 000 игроков на сервере одновременно).

Эксперимент считается неудавшимся, если оценка количества превысит значение 3 (300% превышение количества одной фракции). Один эксперимент будет состоять из генерации игровых событий количеством 12 000 шагов, что эквивалентно реальному году нахождения игры на рынке. Проверка на неудачность начинается с 500 шага из-за влияния холодного старта на показатель количества.

3.1 Генерация без блока управления

Для начала проведем генерацию с целью — выяснить, возможно ли достижения баланса без блока управления.

В результате генерации без поглощения было произведено 2921693 итераций, из них 0 успешно.

Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

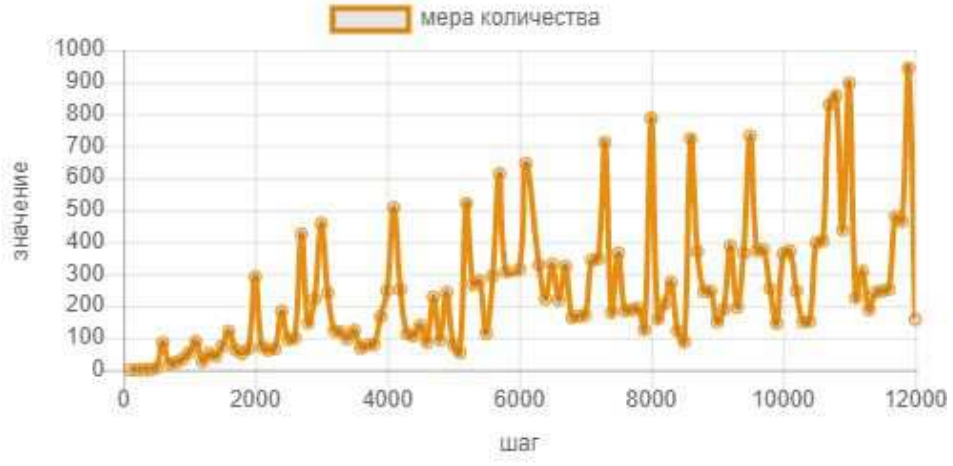

Рисунок 3.1 — Мера количества

На рисунке видно, что с увеличением шага мера количества начинает резко колебаться и достигать чрезмерно больших значений.

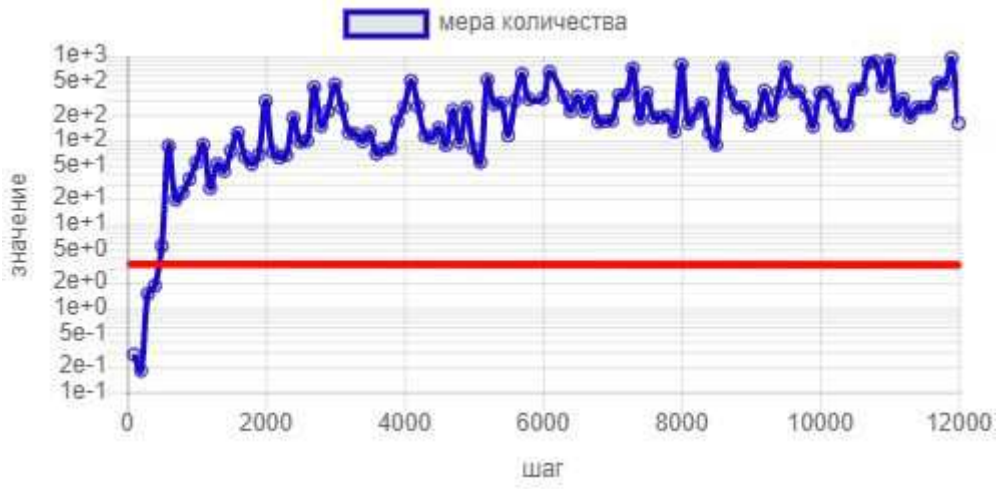

Рисунок 3.2 — Мера количества, логарифмический масштаб

В логарифмическом масштабе более наглядно видно как произошел скачок. Приблизительно, на 500 шаге, мера начала резко расти и пересекла допустимый предел (красная черта на рисунке). Это лавинный процесс — на плавное изменение входных параметров система отреагировала резким изменение данных. Далее, до конца эксперимента, среднее значение меры количества зафиксировалось в районе 200 (превышение 20 000%).

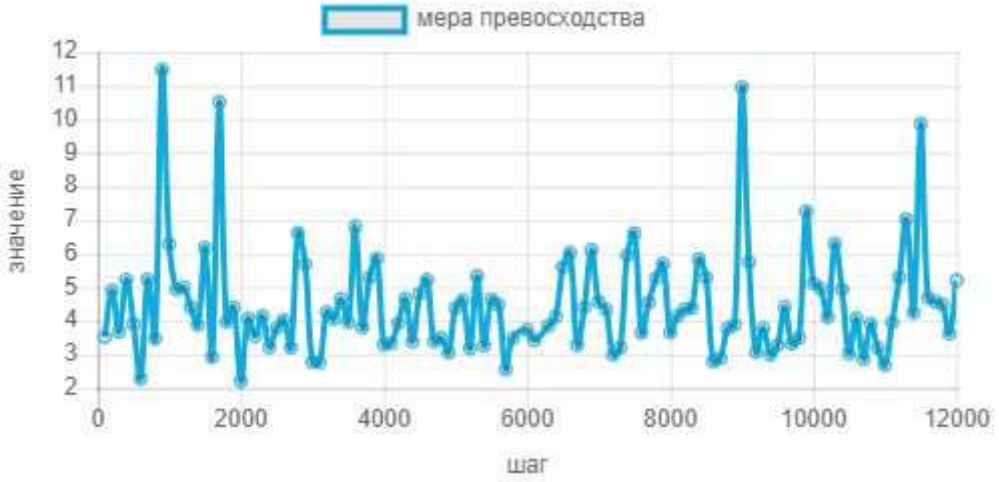

Рисунок 3.3 — Мера превосходства

На рисунке мера превосходства имеет среднее значение порядка 4.5, что является вполне приемлемым, и показывает на то, что любой член фракции сильнее любого члена другой фракции не более, чем в 4 раза.

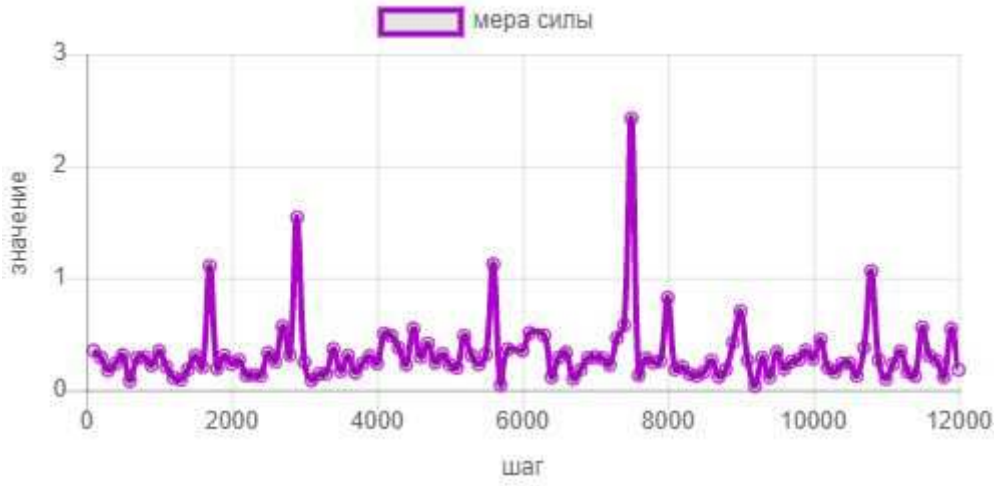

Рисунок 3.4 — Мера силы

Мера силы имеет среднее значение 0.5, что указывает на то, что перевес по средней силе члена фракций по сравнению с членом другой фракции составляет не более 50%.

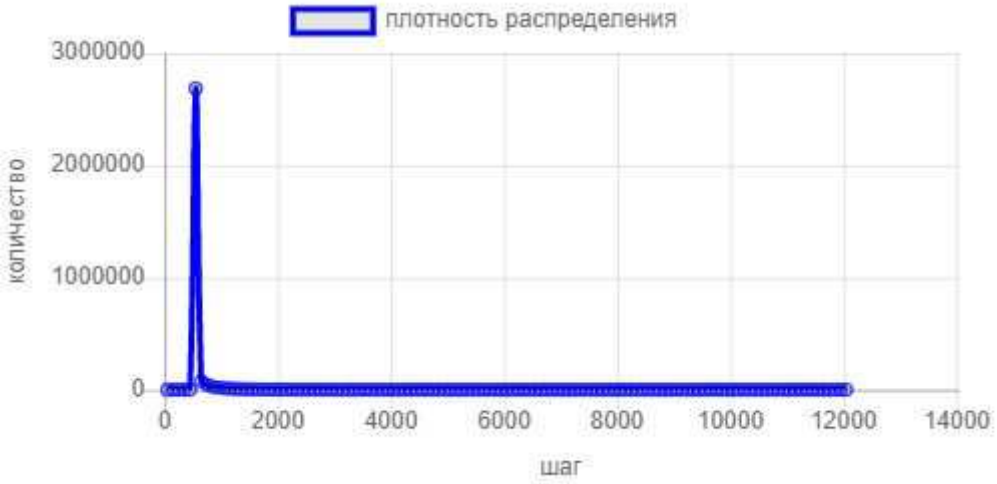

Рисунок 3.5 — Количество провалов экспериментов в шаг экспериментов

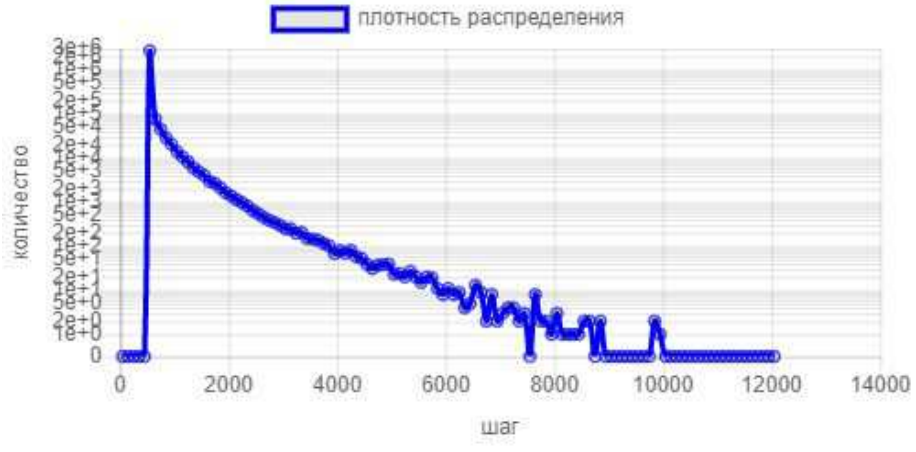

Рисунок 3.6 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- количество одной из фракций в среднем превышает допустимый предел не менее, чем на один порядок (больше в 50 раз, когда допустимо не более, чем в 3, отмечено красной линией);
- мера превосходства находится в допустимых пределах;
- мера силы находится в допустимых пределах;
- распределения плотности указывают на то, что большая часть экспериментов проваливается при первой проверке, и лишь малая часть продолжается дальше.

В результате генерации с поглощением было произведено 11497697 итераций, из них 0 успешно.

Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

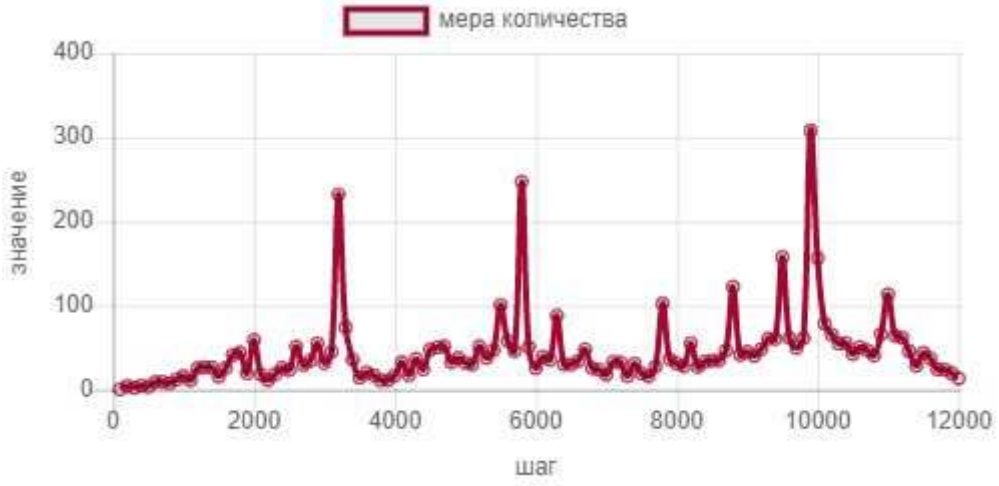

Рисунок 3.7 — Мера количества

На рисунке видно, что среднее значение меры количества находится в районе 50. Также видны резкие всплески, которые не являются выбросами — это лавинные процессы.

Рисунок 3.8 — Мера количества, логарифмический масштаб

На рисунке видно, как мера количества превышает допустимое значение (красная линия на графике). Также видно 4 лавинных процесса — на 500, 3000,5500, 8000 шаге. Первый из них привел к превышению, последующие вызвали колебания отношения количества.

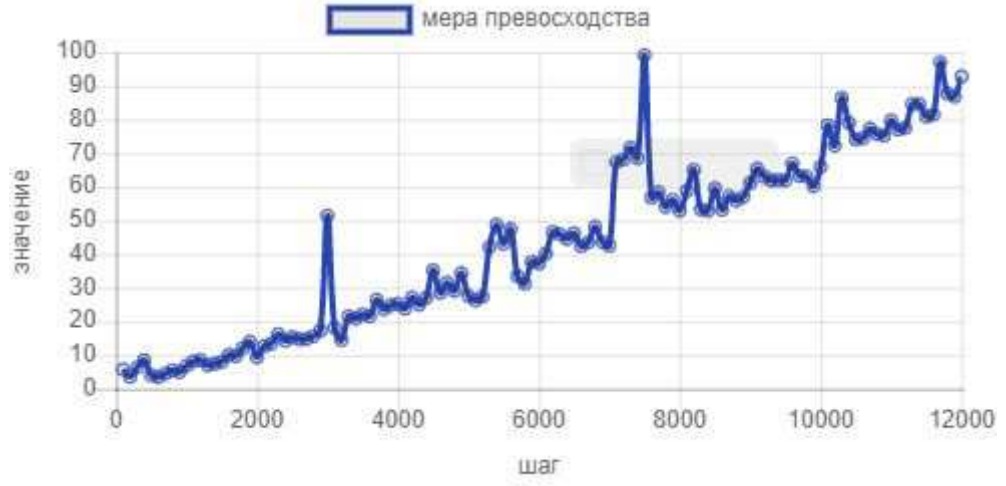

Рисунок 3.9 — Мера превосходства

На рисунке видно возрастающую последовательность. Это объясняется тем, что в мире появилось существо, которое побеждает всех своих противников (босс, очень сильное игровое существо, которое, как правило, не победить в одиночку). Этот рост ожидаем для генераций с поглощением, и его не избежать.

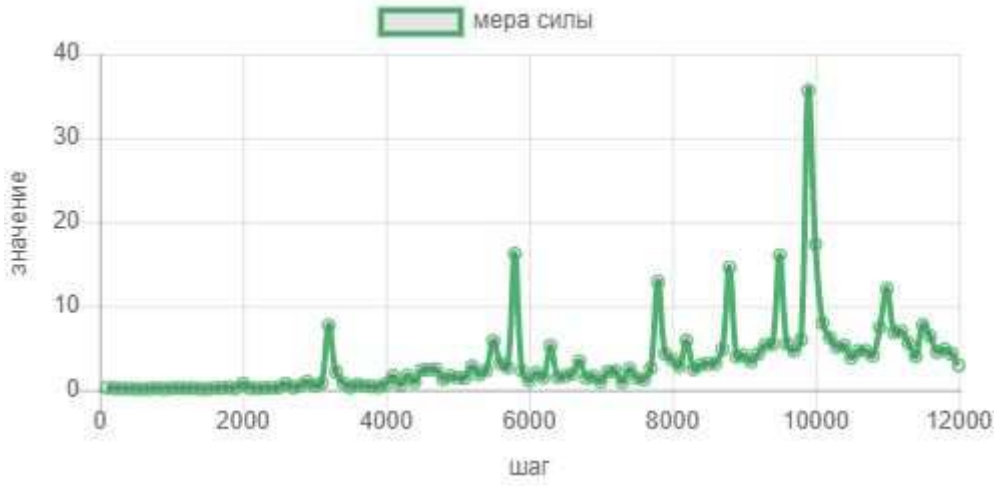

Рисунок 3.10 — Мера силы

Хоть в среднем значение силы находится около 6, на графике наблюдаются резкие всплески (пики). Они обусловлены сокращением числа фракции с сильной особью, поскольку метрика силы чувствительнее реагирует на большие значения характеристик при малом количестве особей, из-за неробастности математического ожидания.

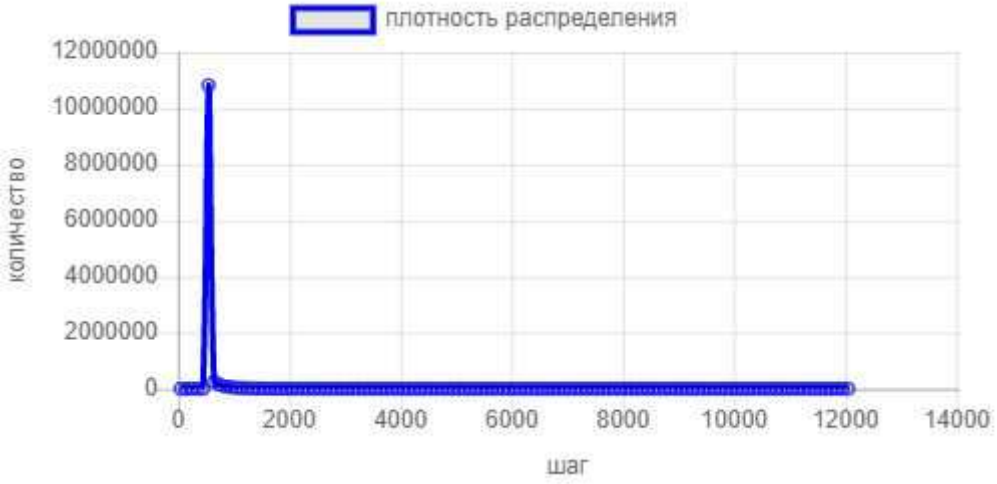

Рисунок 3.11 — Количество провалов экспериментов в определенный шаг

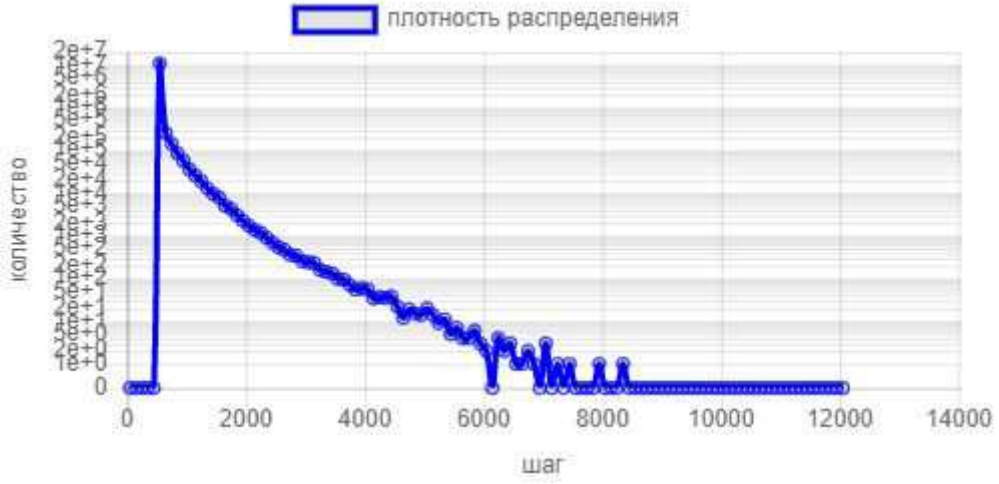

Рисунок 3.12 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- количество одной из фракций в среднем превышает допустимый предел не менее, чем на один порядок (больше в 30 раз, когда допустимо не более, чем в 3, отмечено красной линией);
- мера превосходства возрастает (ожидаемый результат при поглощении). Это не свойственно генерациям с поглощением, и указывает на развитие сущностей в игровом мире (прокачка персонажа, появление босса);
- мера силы большую часть времени находится в допустимых пределах;
- распределения плотности указывают на то, что большая часть экспериментов проваливается при первой проверке, и лишь малая часть продолжается дольше.

Сравнение генераций:

- при генерации с поглощением лучше стабильность количества (превышение почти в 2 раза ниже 30 против 50);
- при генерации с поглощением могут возникнуть чрезмерно сильные члены фракций — это можно заметить по мере превосходства (ее стабильное увеличение указывает на особь с постоянным развитием), также они влияют на меру силы при низком количестве членов противоположной фракции.

Общий вывод — большая часть неудавшихся экспериментов находится при первой проверке (500 шаг), это обусловлено проблемой холодного старта (которая не будет рассмотрена в рамках данной работы).

3.2 Генерация при помощи генетического алгоритма

В результате генерации без поглощения было произведено 695887 итераций, из них 0 успешно. Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

Рисунок 3.13 — Мера количества

На рисунке 3.13 видно 1 лавинный процесс, который был подавлен. Однако, при этом эксперимент является неуспешным, так как было превышение допустимой нормы метрики.

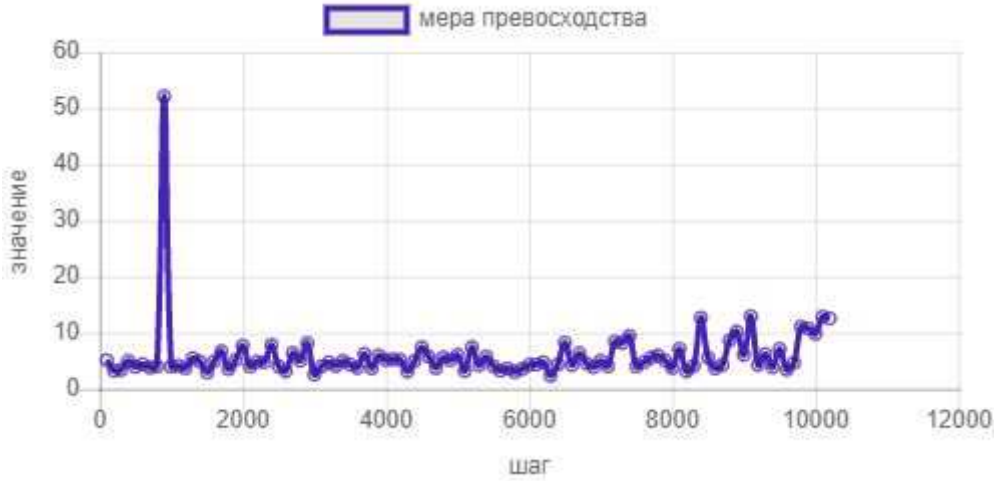

Рисунок 3.14 — Мера превосходства

В среднем мера превосходства находится около 7, однако, в самом начале есть резкий всплеск и ближе к концу наблюдаются колебания, это обусловлено большой численностью одной из фракций, а также высокой смертностью ее членов. При высокой численности вероятность появления особи с характеристиками, близкими к минимальным, растет, когда она появляется, происходит всплеск, при ее смерти происходит уменьшение этого всплеска (колебания).

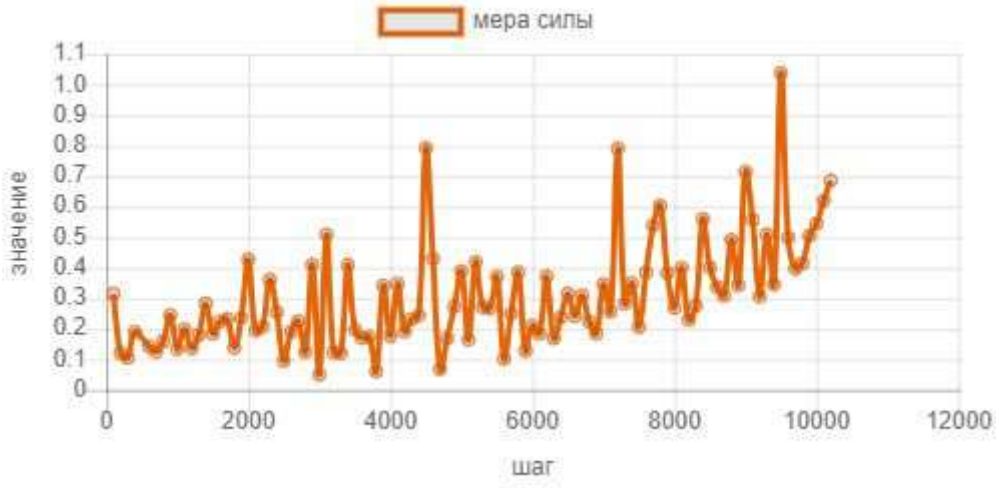

Рисунок 3.15 — Мера силы

В среднем мера силы имеет значение около 0.4, что находится в рекомендуемых пределах, и говорит о силовой сбалансированности фракций. Также на графике видны всплески, они обусловлены резкими перепадами численности одной из фракций.

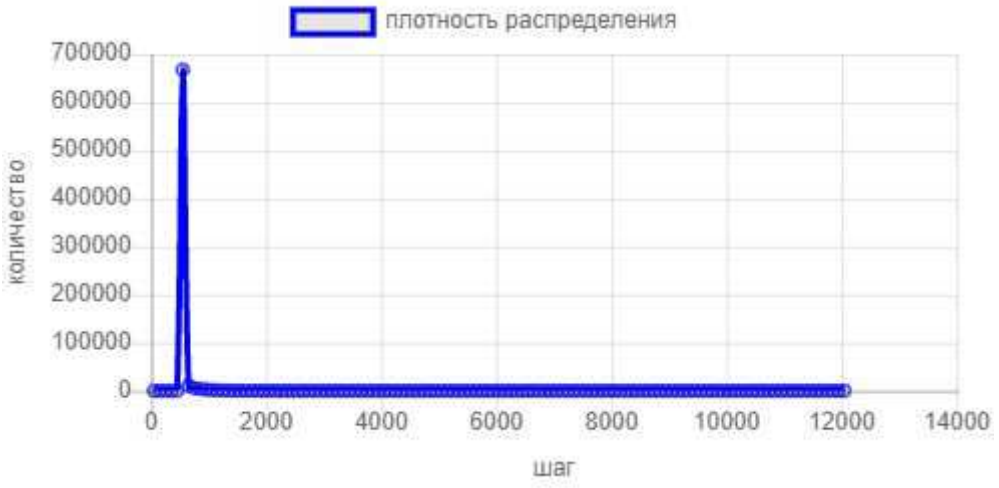

Рисунок 3.16 — Количество провалов экспериментов в определенный шаг

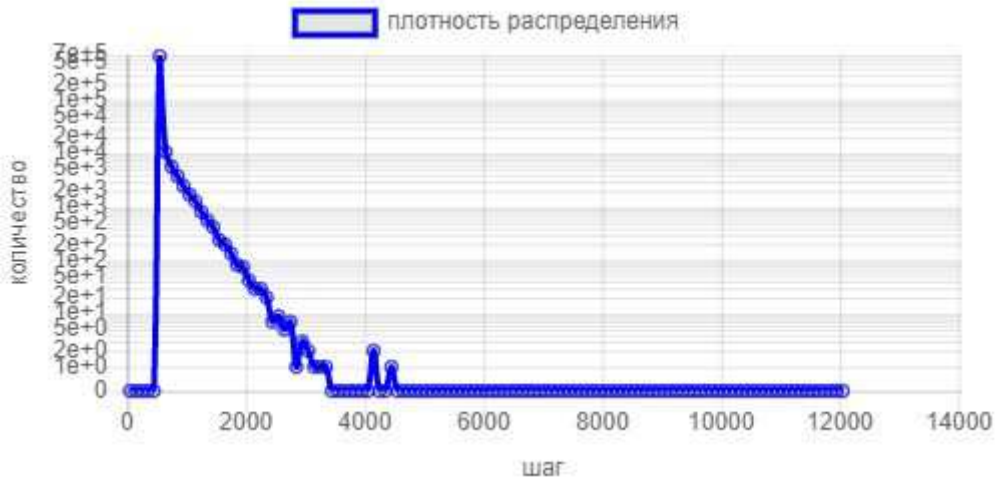

Рисунок 3.17 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- по мере количества генетический алгоритм показывает неплохой результат, но все равно не может справиться с задачей;
- мера превосходства находится в рекомендуемых границах;
- мера силы находится в рекомендуемых границах;
- плотности распределения показывают, что после начала проверок отсеивается большая часть экспериментов, а также, что дальше 5000 шагов ни один эксперимент не прошел.

В результате генерации с поглощением было произведено 2282835 итераций, из них 2 успешно. Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

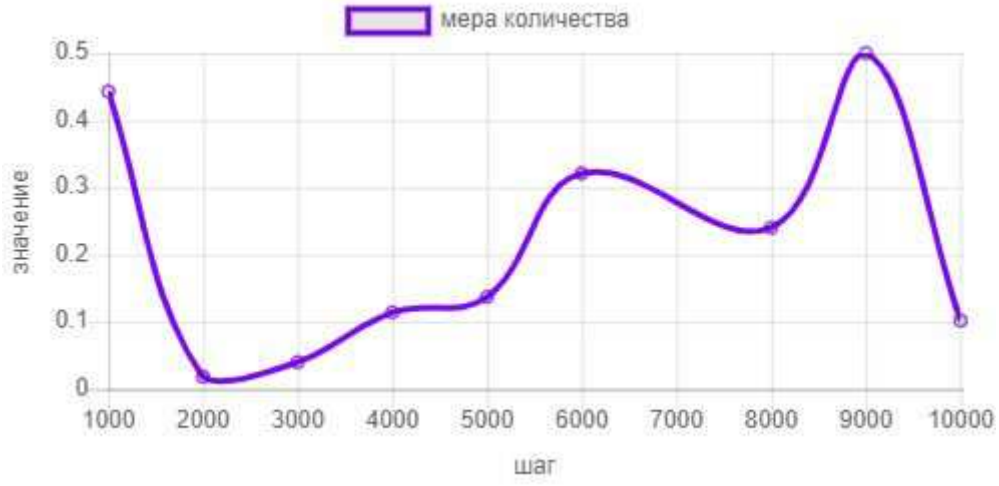

Рисунок 3.18 — Мера количества

На рисунке видно, что значение меры количества не превышало 0.5, а в среднем было на уровне 0.2, что говорит о том, что разница по количеству между фракциями была около 20%. Это очень хороший показатель.

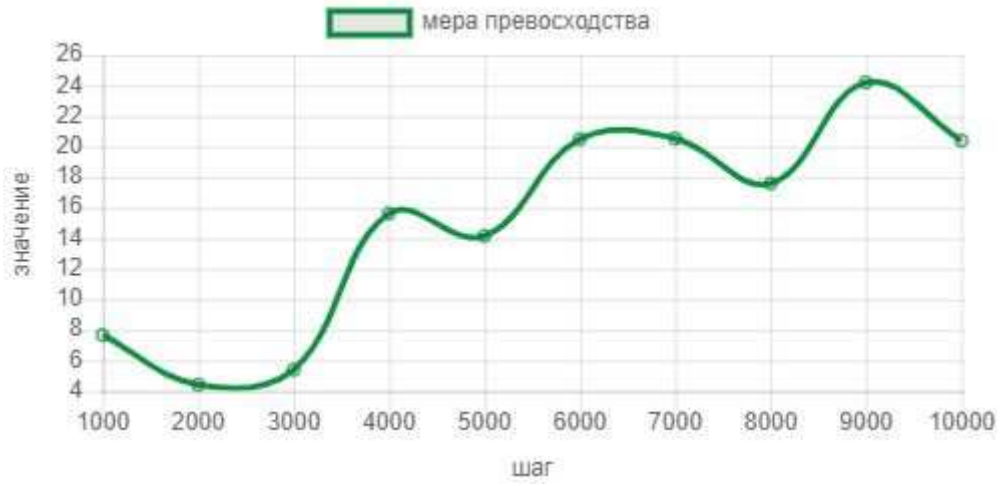

Рисунок 3.19 — Мера превосходства

Мера превосходства в среднем имеет значение около 18, это хороший показатель, так как в данном случае есть поглощение, и многократные увеличения метрики не являются редким явлением.

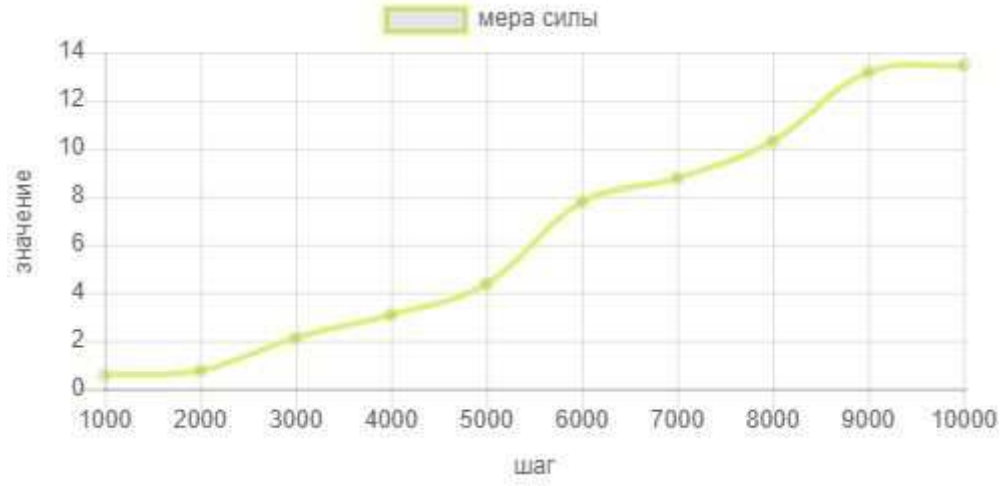

Рисунок 3.20 — Мера силы

Мера силы в среднем имеет значение около 8, это хороший показатель с учетом поглощения и появления босса.

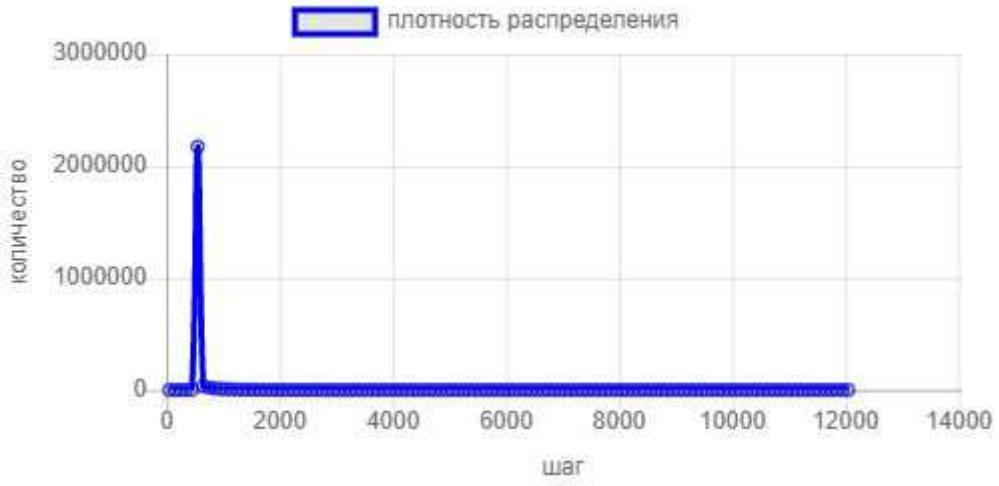

Рисунок 3.21 — Количество провалов экспериментов в определенный шаг

Рисунок 3.22 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- по мере количества генетический алгоритм показывает хороший результат;
- мера превосходства находится в допустимых границах;
- мера силы находится в допустимых границах;
- плотности распределения показывают, что после начала проверок отсеивается большая часть экспериментов, а также, что дальше 5000 шагов прошло всего несколько экспериментов.

Выводы по генерации:

- генетический алгоритм показывает хорошие результаты, в случае с поглощением были даже успешные итерации;
- генетический алгоритм из-за сложности модели не может хорошо адаптироваться под нее.

3.3 Генерация при помощи порогового алгоритма

В результате генерации без поглощения было произведено 1699686 итераций, из них 20 успешно. Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

Рисунок 3.23 — Мера количества

На рисунке значения не превышают 0.6 и среднее значение около 0.2. Также видны пики — часть из них является началом лавинного процесса. Но, из-за алгоритма, он предотвращается, поддерживая состояние системы в желаемом диапазоне.

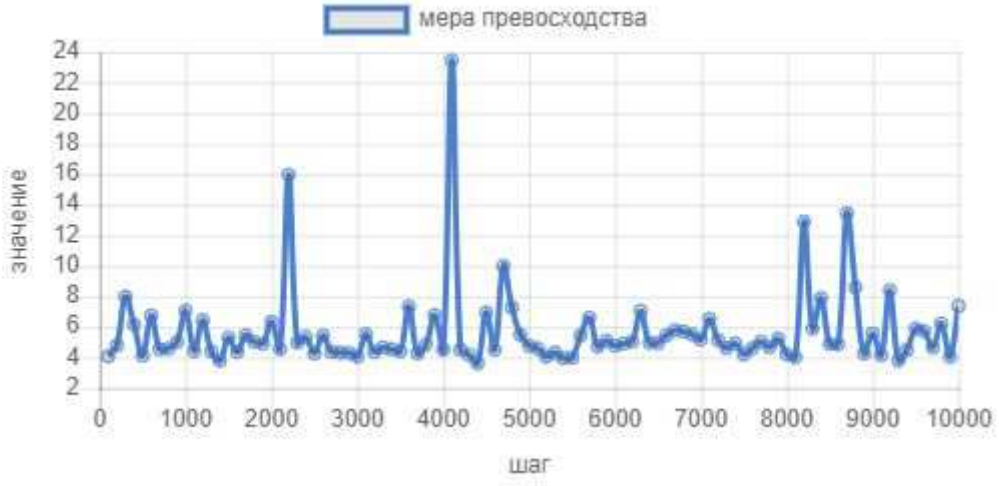

Рисунок 3.24 — Мера превосходства

Мера превосходства имеет в среднем значение около 6, что является нормальным показателем. Также видны пики, которые являются сигналами о предстоящей лавине из-за снижения численности фракции.

Рисунок 3.25 — Мера силы

Среднее значение метрики силы около 0.2, что указывает на прекрасную сбалансированность фракции — при возможной разнице в генерации до 50%, разница составляет в среднем всего 20%.

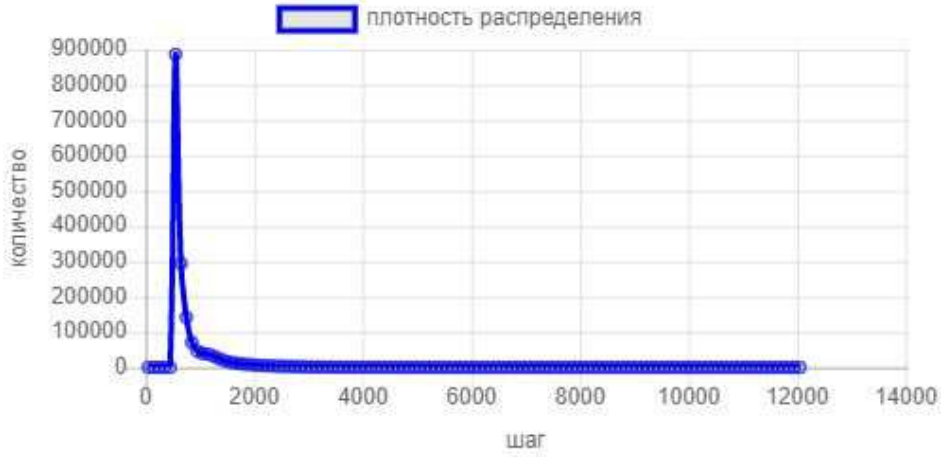

Рисунок 3.26 — Количество провалов экспериментов в определенный шаг

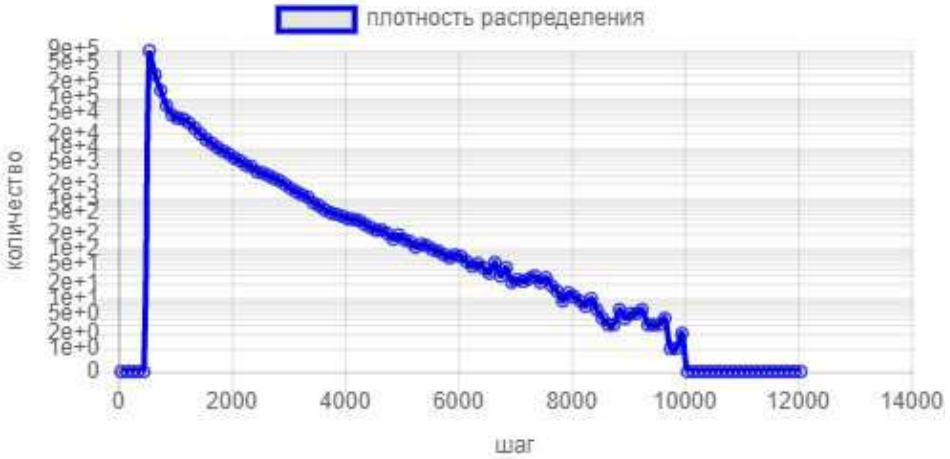

Рисунок 3.27 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- по мере количества пороговый алгоритм показывает хороший результат;
- мера превосходства находится в допустимых границах;
- мера силы находится в допустимых границах;
- плотности распределения показывают, что после начала проверок отсеивается большая часть экспериментов, но при этом до конца доходит больше экспериментов, чем в генетическом алгоритме и обычном моделировании.

В результате генерации с поглощением было произведено 173282 итераций, из них 3043 успешно. Выбрав из результатов генерации итерацию с наименьшим дисбалансом по количеству, ее оценки выглядят следующим образом:

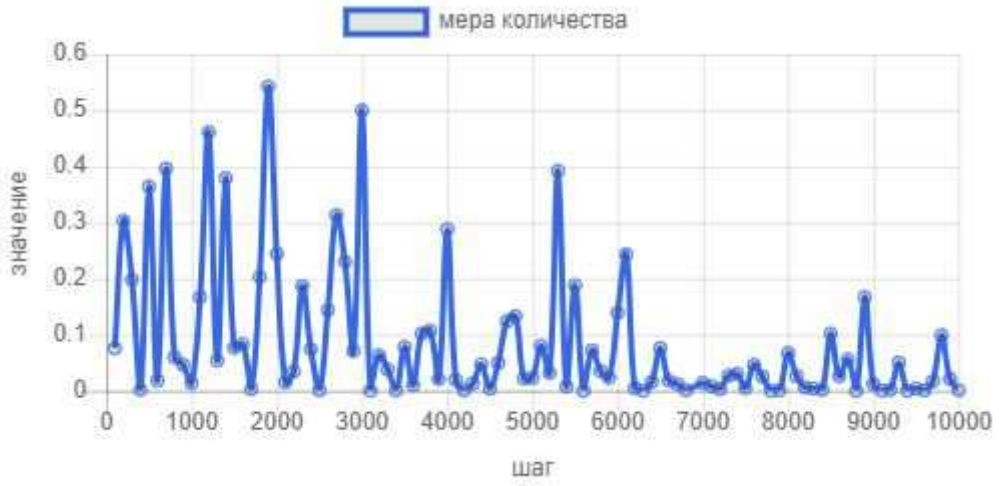

Рисунок 3.28 — Мера количества

На рисунке среднее значение около 0.15, что является превосходным результатом и показывает средний дисбаланс 15%. Также видны всплески в начале и середине — часть из них, это начало лавинных процессов, которые успешно предотвратил алгоритм.

Рисунок 3.29 — Мера превосходства

На рисунке мера превосходства около 10, что является нормальным показателем для эксперимента с поглощением. Также виден всплеск, он обусловлен появлением и смертью босса, при этом лавины из-за этого не произошло.

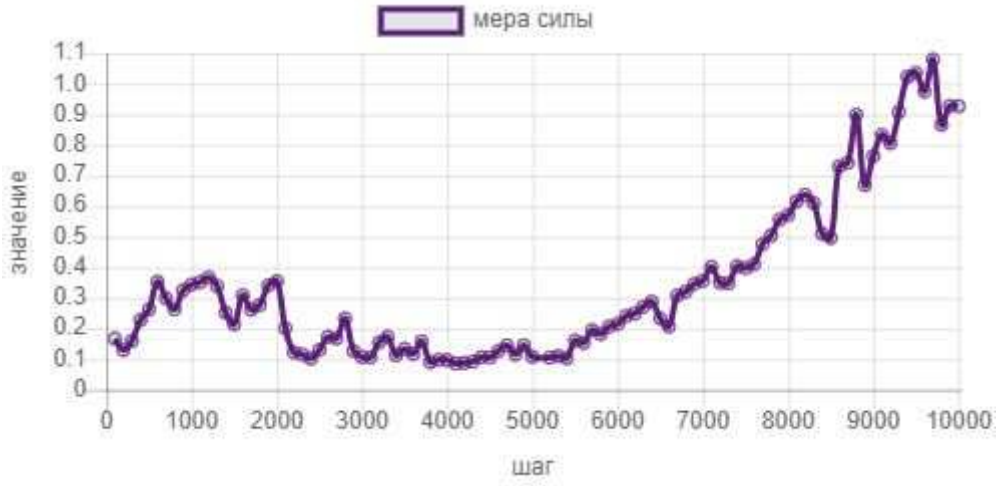

Рисунок 3.30 — Мера силы

На рисунке видно, что метрика сначала стремится к 0 (стабилизируется), но потом начинает расти, что свидетельствует о наборе силы одной фракцией (равномерные победы над другой фракцией без появления босса в результате).

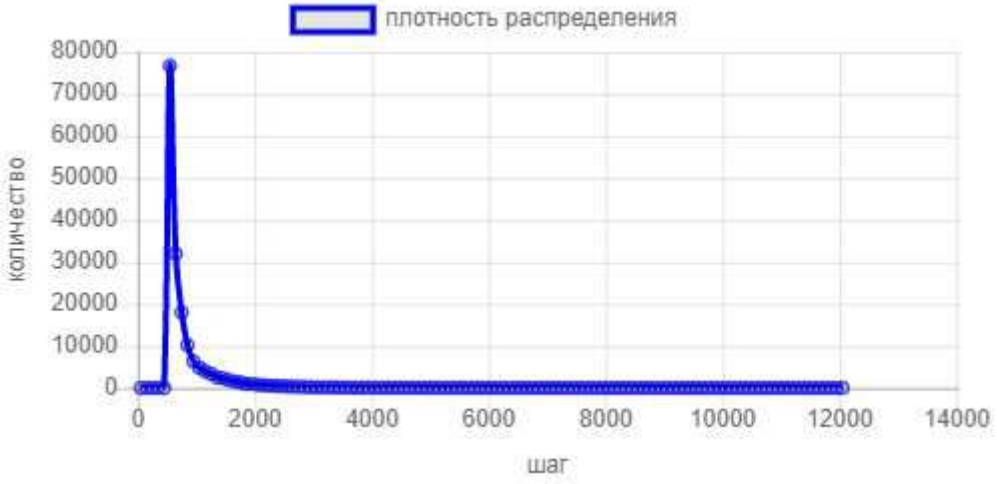

Рисунок 3.31 — Количество провалов экспериментов в определенный шаг

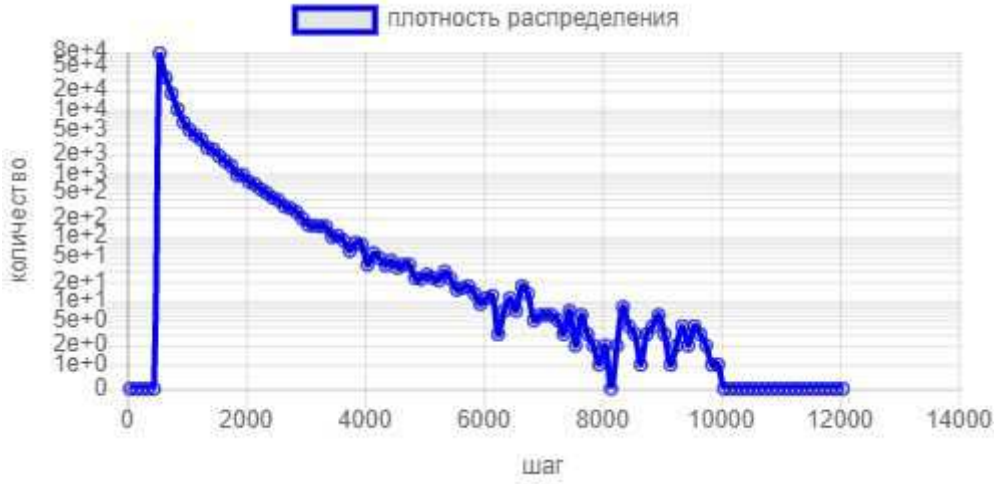

Рисунок 3.32 — Количество провалов экспериментов в определенный шаг в логарифмическом масштабе

Выводы по рисункам:

- по мере количества пороговый алгоритм показывает хороший результат;
- мера превосходства находится в допустимых границах;
- мера силы находится в допустимых границах;
- плотности распределения показывают, что после начала проверок отсеивается большая часть экспериментов, но при этом до конца доходит больше экспериментов, чем в генетическом алгоритме и обычном моделировании.

Выводы по генерации:

- пороговый алгоритм показал хорошие результаты по сравнению с генетическим алгоритмом;
- алгоритм показывает высокую устойчивость при генерации.

3.4 Анализ плотностей распределения

Для начала взглянем на общую картину плотностей распределения.

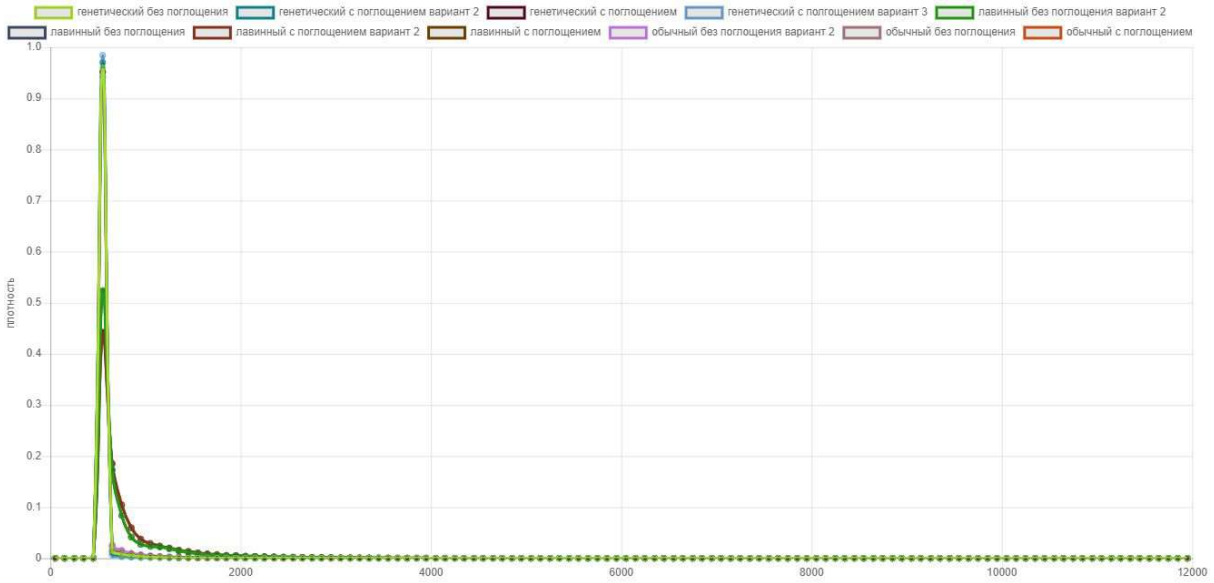

Рисунок 3.33 — Все плотности распределения

На этом рисунке и далее по оси абсцисс откладывается номер шага, по оси ординат плотность неуспешных экспериментов по выбранному диапазону. На рисунке можно заметить, что основная часть экспериментов обрывается до 2000 хода — это связано с тем, что в момент запуска игры происходят резкие пики спады активности игроков (проблема холодного старта), это приводит к серьезной нестабильности. Ее устранение может увеличить число игроков, кто будет играть в эту игру — т. е. повысить прибыль.

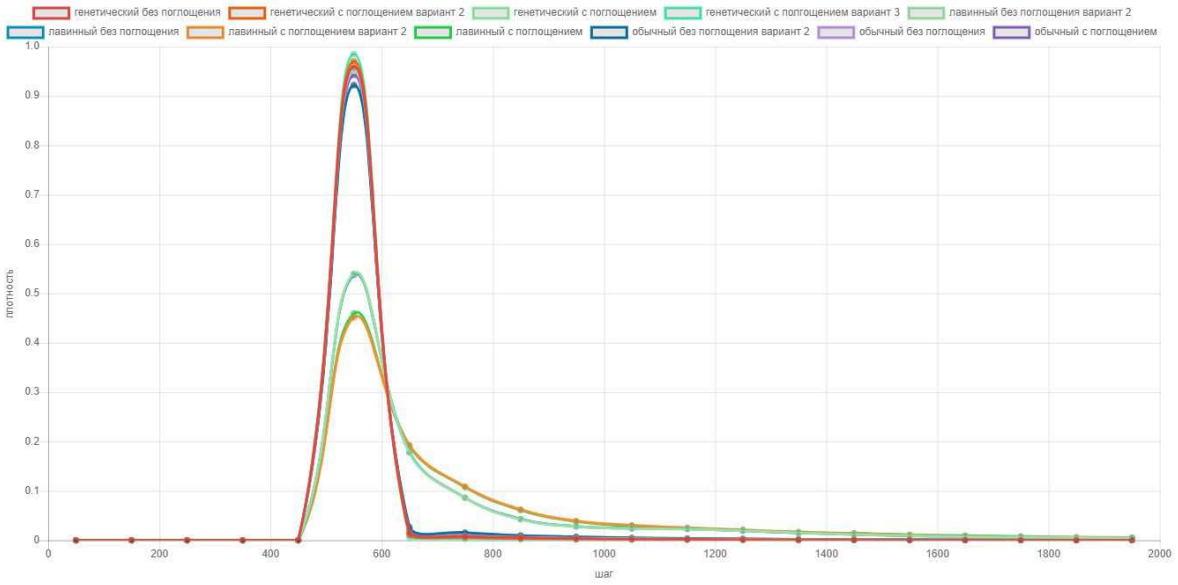

Рисунок 3.34 — Плотность распределения 0 - 2000

На рисунке плотность пересчитана в диапазоне от 0 до 2000 шагов (2 месяца реального времени). На нем видно, что пороговый алгоритм имеет более плавную форму и равномернее распределен по плотности, в тоже время генетический алгоритм и эксперименты без блока управления имеют более крутой пик. Этот пик обусловлен началом проверок на отсечение при проведении эксперимента, при этом сами алгоритмы работают с самого начала ходов. Взглянем на этом отрезке на плотности генетического алгоритма и без блока управления более детально.

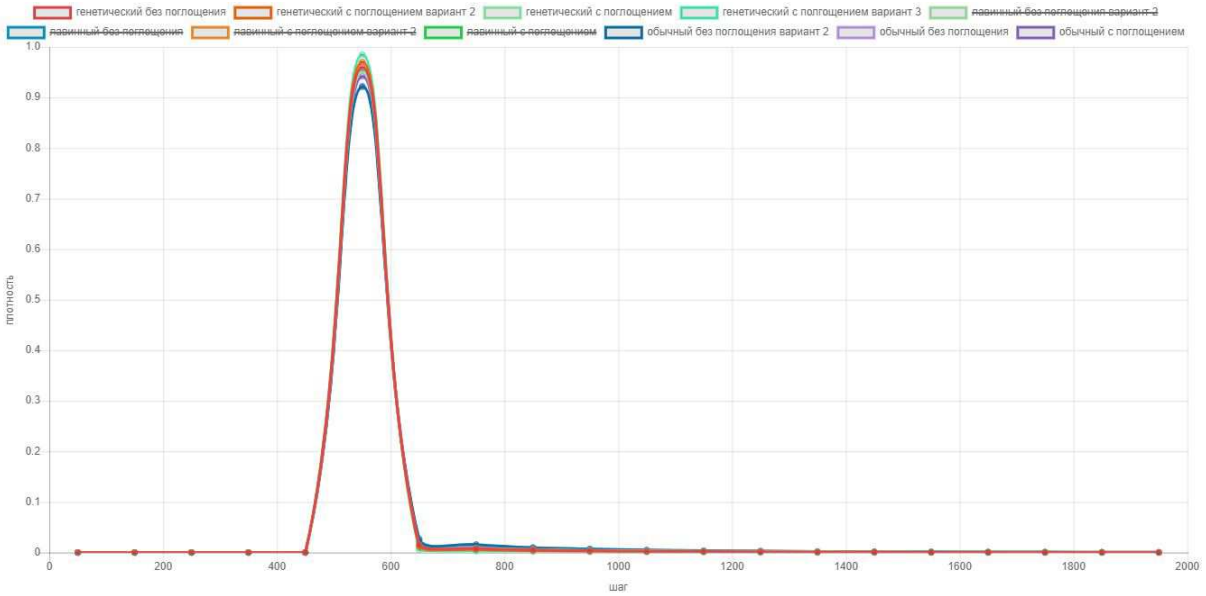

Рисунок 3.35 — Плотность распределения 0 - 2000

На рисунке хорошо видно наличие резкого пика. Его наличие говорит о двух вещах:

- генетический алгоритм или при отсутствии блока управления, приводят к резким пикам в плотности из-за того, что они не могут справиться с активностью игроков в самом начале;

- в дальнейшем генетический алгоритм справляется с задачей, и количество неудачных экспериментов стремится к нулю.

Теперь рассмотрим пороговый алгоритм.

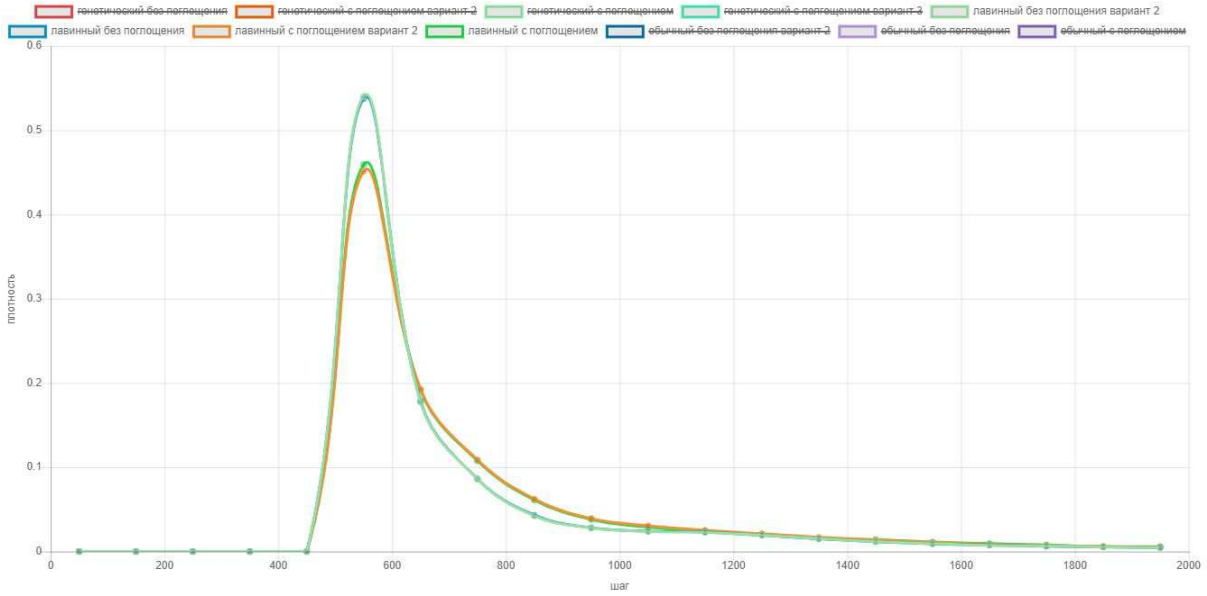

Рисунок 3.36 — Плотность распределения 0 - 2000 пороговый алгоритм

На этом рисунке тоже видно резкий пик, однако его склон более пологий, что указывает на то, что алгоритм лучше справляется с поставленной задачей. Это обуславливается тем, что неудачные эксперименты отбрасываются на более позднем шаге, что позволяет системе дольше находиться в стабильном состоянии. Также на рисунке видно, как поглощение улучшает результаты (делает график более пологим), это объясняется тем, что поглощение добавляет больше инертности, это в свою очередь приводит к тому, что система менее резко реагирует на изменения, и у алгоритма появляется больше возможностей стабилизировать систему.

Предположим, что первые 1000 ходов (1 месяц реального времени) система стремится к стабильному состоянию. Поэтому рассмотрим хвост пика из предыдущих рисунков более детально.

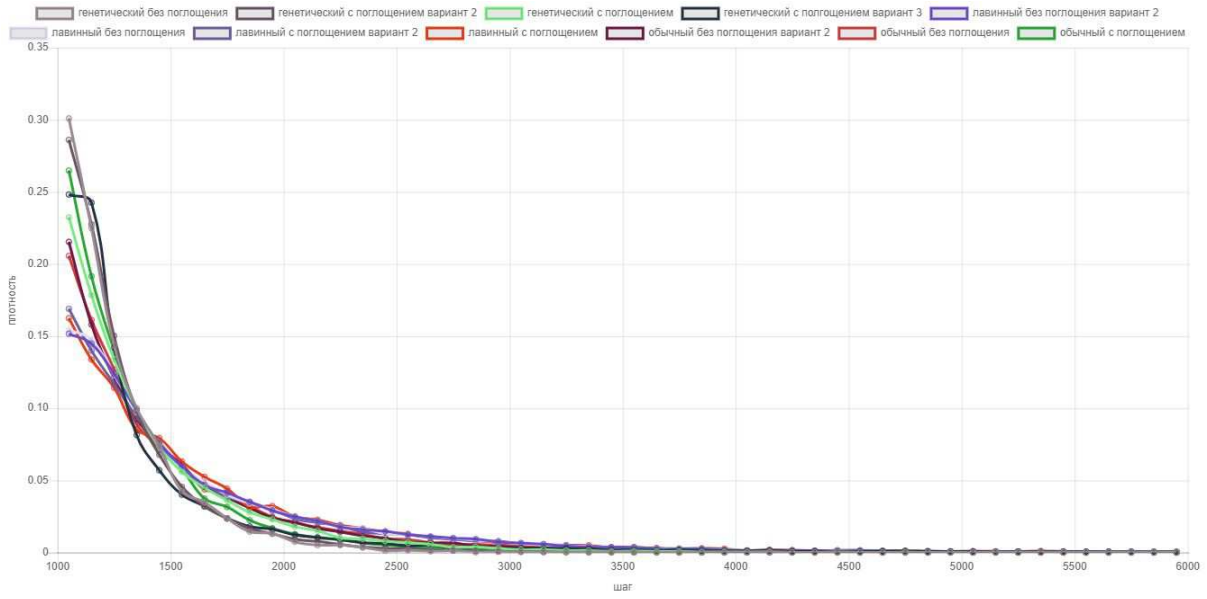

Рисунок 3.37 — Плотность распределения 1000 - 6000

На этом рисунке можно заметить, что после 4000 ходов все плотности стремятся к нулю, поэтому рассмотрим более детально промежуток 1000 - 4000 ходов.

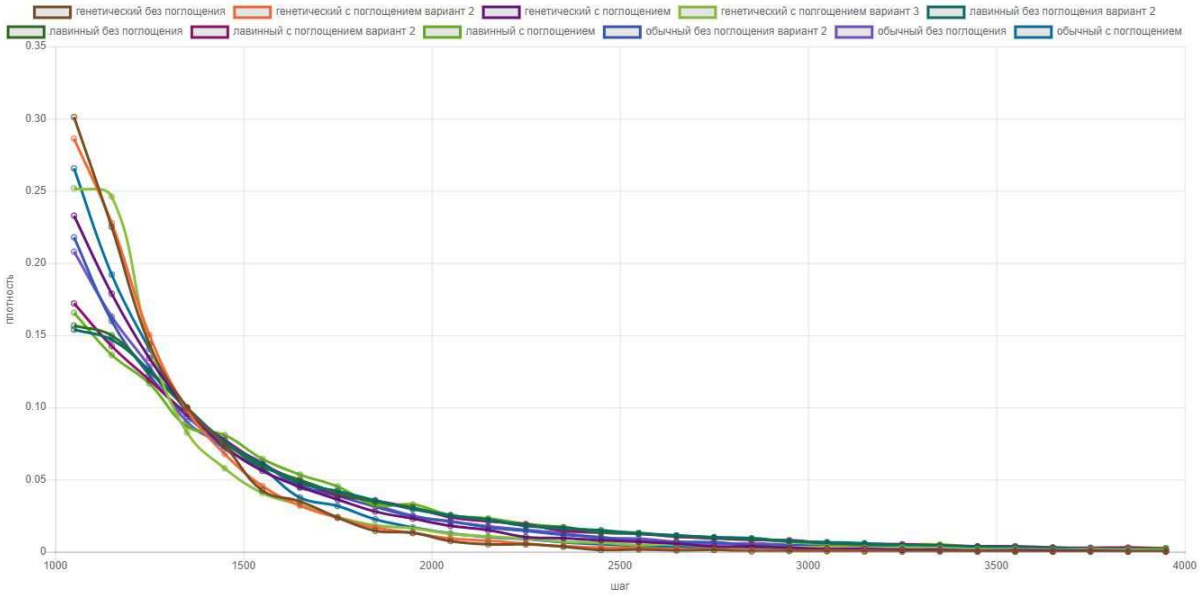

Рисунок 3.38 — Плотность распределения 1000 - 4000

На рисунке можно заметить, что все графики генераций с поглощением являются более пологими. Это можно более детально увидеть на следующих рисунках:

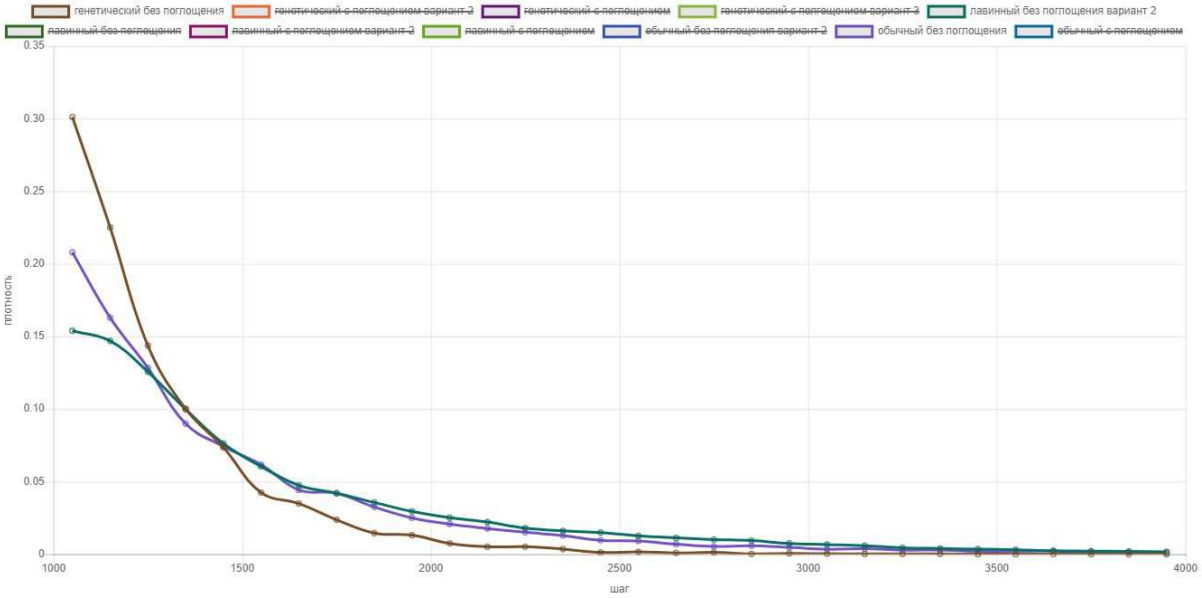

Рисунок 3.39 — Плотность распределения 1000 - 4000 без поглощения

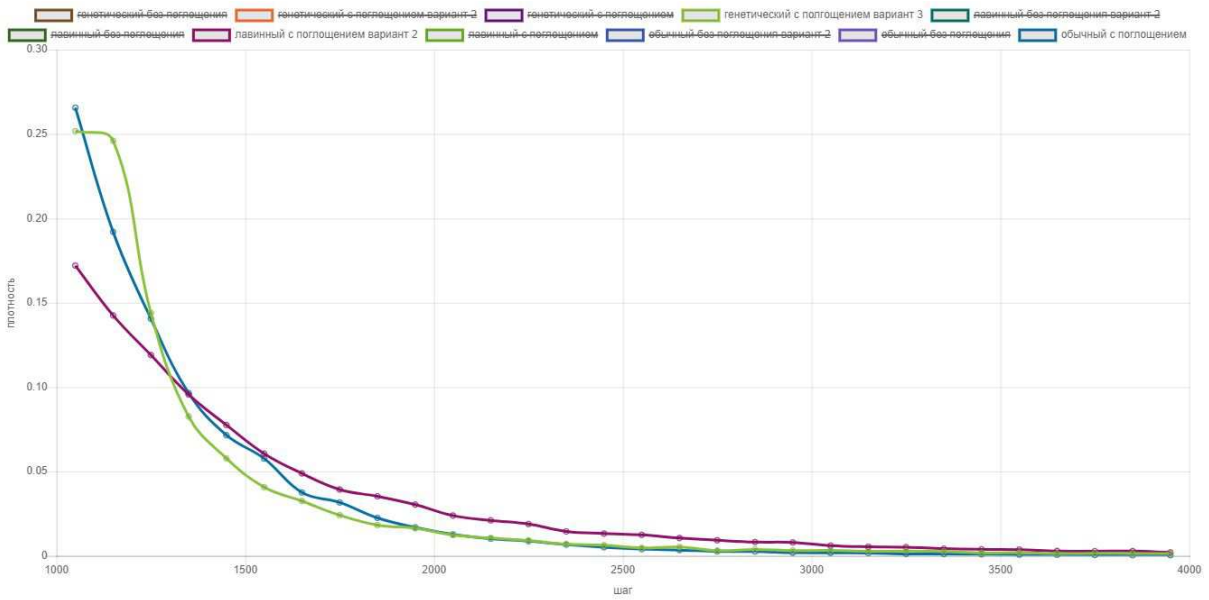

Рисунок 3.40 — Плотность распределения 1000 - 4000 с поглощением

Также на рисунках видно, что пороговый алгоритм, который показывает наилучший результат по сравнению с другими алгоритмами, стремится к более пологой форме.

3.5 Сравнение алгоритмов

В результате генераций и изучения данных по ним была составлена следующая таблица:

Таблица 3.1 — Успешность экспериментов

Алгоритм	Без поглощения			С поглощением		
	Всего	Успешно	Процент	Всего	Успешно	Процент
Без блока управления	2921693	0	0%	11497697	0	0%
Генетический	695887	0	0%	2282835	2	0.0000876%
Пороговый	1699686	20	0.001176%	173282	3043	1.756%

По таблице можно сделать следующие выводы:

- без блока управления система не может устранить или подавить лавинные процессы;
- генетический алгоритм не успевает адаптироваться в системе с низкой инертностью (система прекращает свое существование раньше, чем алгоритм приспособится);
- пороговый алгоритм может остановить часть лавин.

По результатам исследования плотностей распределения неудачных экспериментов (глава 3.4) можно сделать следующие выводы:

- большинство экспериментов прекращается при первой проверке (500 шаг), это обусловлено проблемой холодного старта;
- без блока управления система не способна справиться с лавинными процессами, это заметно из-за крутизны пика для распределения неудачных экспериментов;
- генетический алгоритм по плотности распределения неудачных экспериментов показывает результат хуже, чем без блока управления, это обусловлено плохой адаптируемостью к быстро изменяющимся в системе параметрам;
- пороговый алгоритм показывает наилучший результат по плотностям измерения.

Теперь подведем итог по каждому алгоритму.

Без блока управления система не может достигнуть состояния баланса. Это обуславливается наличием стохастичности в модели появления игроков (нельзя точно предсказать интенсивность появления игроков наперед). Периодические всплески интенсивности выбивают систему из устойчивого состояния, либо поддерживают и/или усиливают нестабильное состояние. Это проявляется в виде лавин, когда приход или уход новой группы игроков может обрушить всю систему (баланс в ней), при этом группа может быть минимальна или вовсе состоять из 1 человека.

При управлении с помощью генетического алгоритма система не может достигнуть стабильного состояния без высокой инерции. Т.е., в малых по количеству игроков системах, алгоритм не успевает адаптироваться к изменяющимся условиям. Напротив, в огромных по количеству системах, алгоритм успевает адаптироваться к изменениям в системе, но при этом качество адаптации плохое и не позволяет поддерживать систему в стабильном состоянии постоянно. Это объясняется тем, что генетический алгоритм ориентирован на управление системой в большей степени, чем подавление лавин из-за обучения.

При управлении с помощью порогового алгоритма система может достигнуть стабильного состояния или вернуться в него. Это обуславливается тем, что основная причина потери стабильности это лавины, которые возникают при изменении численности игроков, а алгоритм, в свою очередь, ориентирован на их подавление. Однако, алгоритм не может прогнозировать лавину заранее и противодействовать ей из-за своей примитивности.

Подводя общий итог, можно сказать, что в рамках данной задачи наиболее эффективен анти-лавинный алгоритм. Но при этом требуется, чтобы он был способен прогнозировать лавину и предотвращать ее.

3.6 Возникшие проблемы

Во время выполнения работы возник ряд проблем и они были успешно решены. Далее представлено детальное описание основных проблем и их решения.

3.6.1 Проблема со слишком большим объемом данных

После тестирования прототипа ПО была обнаружена проблема [16] передачи на вэб-интерфейс большого объема данных — от 100 мегабайт до нескольких гигабайт (рисунок 3.41).

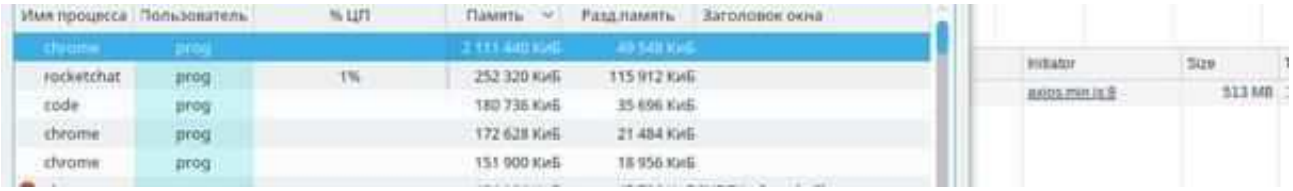

Рисунок 3.41 — Потребляемые ресурсы

Такой объем передачи многократно превосходит стандартные объемы передачи данных в JSON формате (стандартный объем данных составляет от 1 Мб до 15-20 Мб). Как видно из рисунка, в браузер поступает 0,5 Гб данных, после чего браузеру требуется 2 Гб оперативной памяти, чтобы обработать эти данные. Это приводит к аварийному завершению работы браузера даже на мощных ПЭВМ, таким образом, проводить анализ большого объема данных (более чем за 10 000 ходов) не представляется возможным.

Чтобы решить эту проблему, было решено сократить количество данных, отдаваемых для анализа [17]. Передача только одного хода, а не всех, а также улучшение на стороне браузера обработки данных позволило решить проблему большого объема передаваемых данных.

3.6.2 Проблема долгой обработки данных

Суть проблемы заключается в долгой обработке данных для визуализации при анализе (на отрисовку графика на 10 000 точек тратится 60-350 секунд, при этом на сам процесс отрисовки тратится менее 1 секунды). При этом проблемы, долгой по времени генерации данных для анализа, не наблюдается. Эта проблема проявилась после устранения проблемы с большими объемами данных. Чтобы ее решить, было применено профилирование программного кода [18].

По результатам профилирования было выяснено, что:

- использование реляционной базы данных замедляет получение данных (16% процессорного времени тратится на функцию получения данных, в то

время, как этот показатель для приложений по анализу данных должен быть не выше 5%);

- текущее использование патчей позволяет экономить ресурсы ПЭВМ, однако, сильно снижает скорость последующего анализа (около 43% процессорного времени тратится на воспроизведение данных (их подготовка), и только 32% на их анализ, при том, что время, затраченное на анализ, должно составлять более 60% от затраченного времени процессора).

Для устранения первой проблемы наилучшим решением стал отказ от использования реляционной базы данных [19]. Теперь данные сохраняются непосредственно на жесткий диск и контроль за их сохранением производится приложением. Если быть точнее, то была реализована очень простая база данных с использованием файловой системы. Поскольку в ней отсутствуют основные механизмы управления транзакциями, это позволило снизить расходы на ресурсы ПЭВМ (до 3% процессорного времени), при этом жертвуя надежностью (механизмы управления транзакциями позволяют восстановить данные в случае аварийных завершений приложения, а также не позволяют повредить данные в иных случаях).

Для решения второй проблемы было принято решение проводить анализ данных непосредственно во время моделирования [20]. Таким образом, в каждый игровой ход снимались показания и пересчитывались метрики, которые использовались в последующем анализе. Такое решение позволило сократить время на подготовку данных для анализа почти до 0 (остались временные затраты на передачу данных в вэб-приложение для визуализации). Также было настроено сохранение метрик действий (эти метрики показывают различные действия, например, появление нового существа на карте или уничтожения монстра героем).

3.6.3 Проблема большого объема данных с метриками

Внедрение сбора метрик создало проблему больших объемов данных на жестком диске ПЭВМ. Данные 1 эксперимента занимали более 10% от объема жесткого диска, в то время, как планировалось проводить серии из тысяч экспериментов, т.е. занимаемый объем 1-м экспериментом должен быть меньше 0.01%. Сбор метрик требует сохранения их в виде файлов, что в свою очередь приводит к чрезмерному увеличению занимаемого места (данные после моделирования ходов занимают 112 Гб). Для решения этой проблемы были протестированы следующие варианты:

- использовать архивирование и сжатие данных [21];
- применить систему патчей к метрикам;
- сократить количество метрик или количество данных, производимое метрикой (делать замеры реже).

Первый вариант позволил сжать размер данных, примерно, в 4 раза. Но, он оказался неприменим для непрерывной генерации данных и их анализа. Это происходит в силу алгоритма сжатия, который в первую очередь заменяет

дубли (не обязательно полных данных, дублем может быть часть числа или строки) на ссылки. Из-за такого алгоритма применить сжатия к системе патчей не представляется возможным, т. к. для хорошего сжатия необходимо, чтобы поток данных был непрерывен, а система патчей генерирует данные кусками, что мешает эффективному сжатию. На практике это означает, что при возможности сжатия данных в 4 раза, при непосредственном применении к системе патчей, эффективное сжатие составляет 1,2-1,5 раза, при существенном увеличении нагрузки на ПЭВМ (дополнительные расходы процессорного времени 10-20%). Однако, несмотря на плохие показатели при непрерывном сжатии данных, сжатие данных после моделирования для их дальнейшего хранения является крайне эффективным способом экономии места (эффективное сжатие до 6 раз).

Второй вариант позволяет на уровне приложения пропускать повторное значение метрик. После его реализации [22] были получены хорошие показатели занимаемого места метриками (сокращение объема метрик в 100 и более раз для моделирования более 100 000 ходов). Также это снизило нагрузку на визуализацию данных, теперь не требуется отображать на графике 100 000 точек для одной метрики, а необходимо отобразить 1000 точек или меньше.

Третий вариант был реализован [23] в виде системы снапшотов (snaphost — снимок/слепок состояния). Система сохраняет самые объемные метрики, а также метрики, не представляющие большого интереса для анализа, не каждый ход, а раз в несколько ходов (на данный момент раз в 1000). Это позволяет существенно сократить занимаемое место на жестком диске (до 60-70 Мб на 100 000 шагов моделирования).

В итоге были реализованы все три варианта — первый для длительного хранения и архивирования данных, второй и третий для непрерывного моделирования.

4 Анти-лавинный алгоритм

В рамках данной главы мы рассмотрим способы повышения качества работоспособности нашего порогового алгоритма.

Рассмотрим более детально значение метрики количества при генерации без блока управления и ее первых двух производных по шагу. Для начала эксперимент без поглощения.

Рисунок 4.1 — Метрика количества

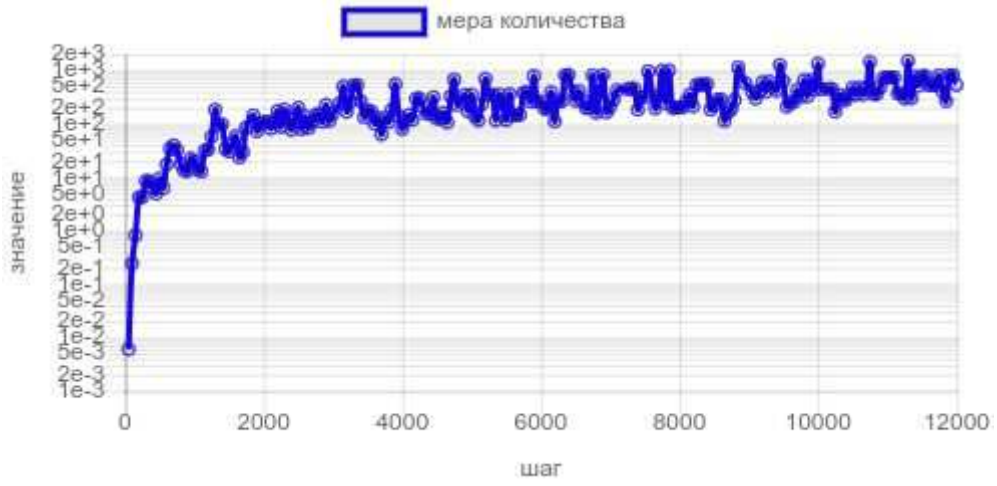

Рисунок 4.2 — Метрика количества логарифмический масштаб

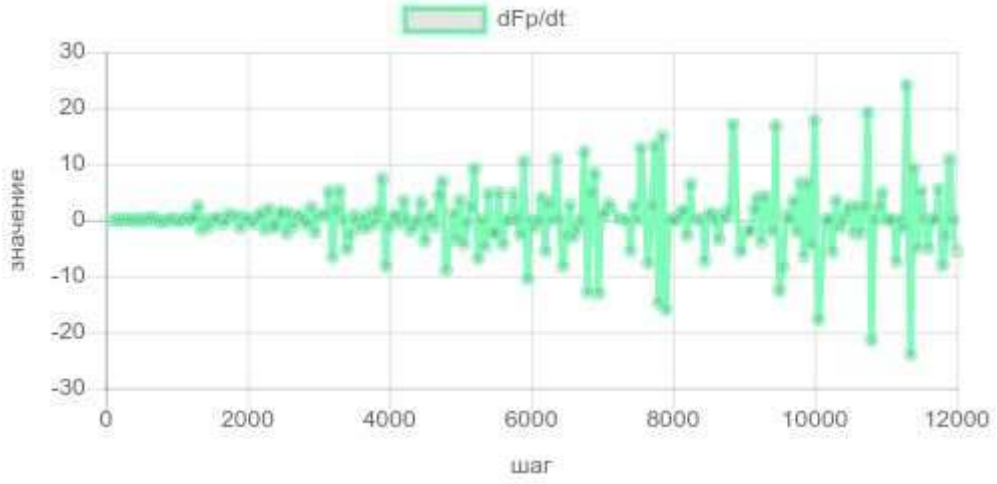

Рисунок 4.3 — Метрика количества первая производная

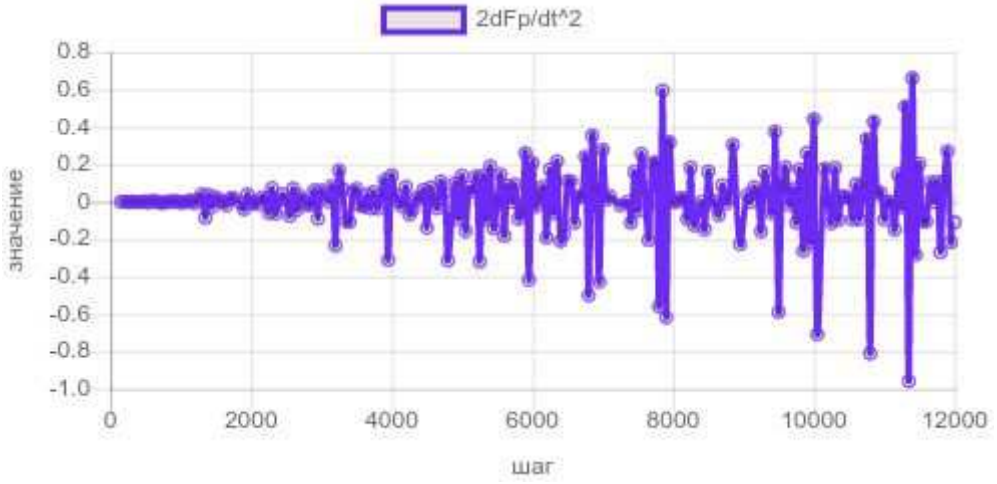

Рисунок 4.4 — Метрика количества вторая производная

В данном случае это не особо заметно, но когда первая и вторая производная имеют положительное значение больше определенной величины, происходит лавинный процесс. Когда они имеют отрицательное значение, то происходит спад лавинного процесса.

Это более наглядно можно увидеть при генерации с поглощением.

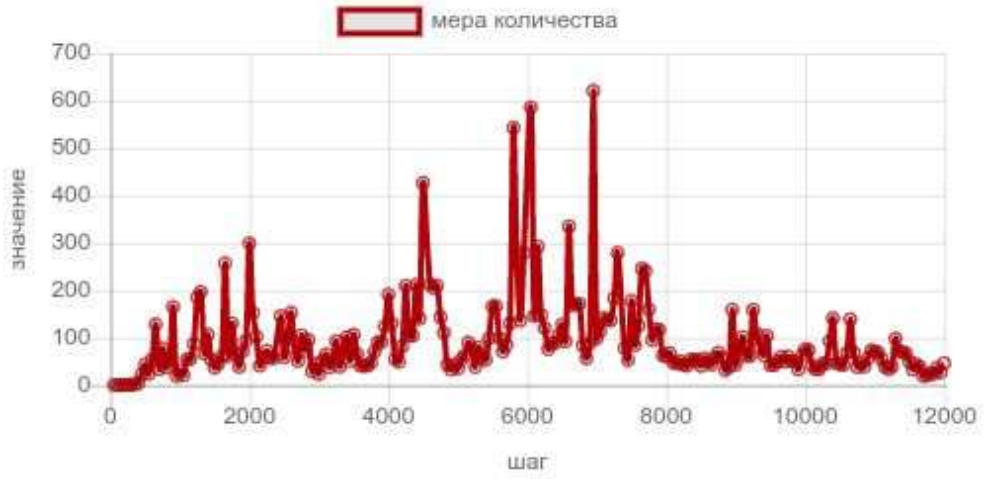

Рисунок 4.5 — Метрика количества

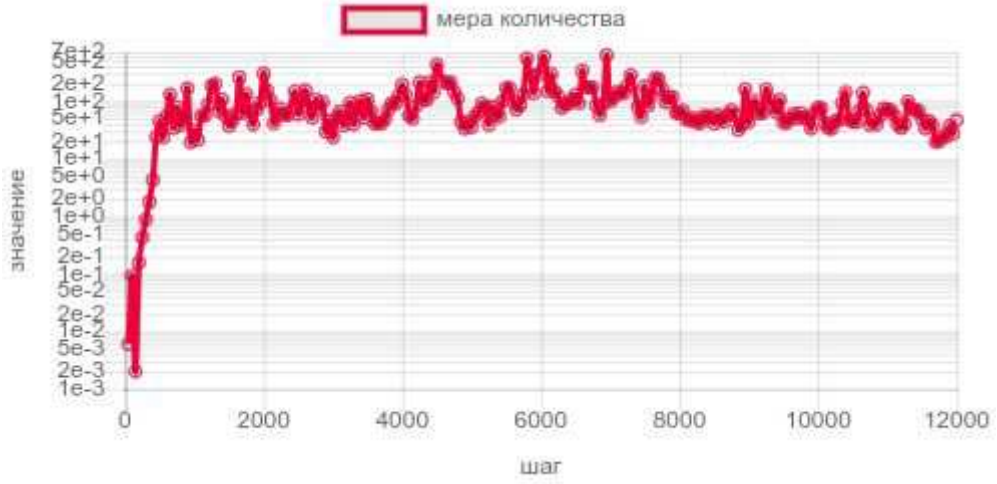

Рисунок 4.6 — Метрика количества логарифмический масштаб

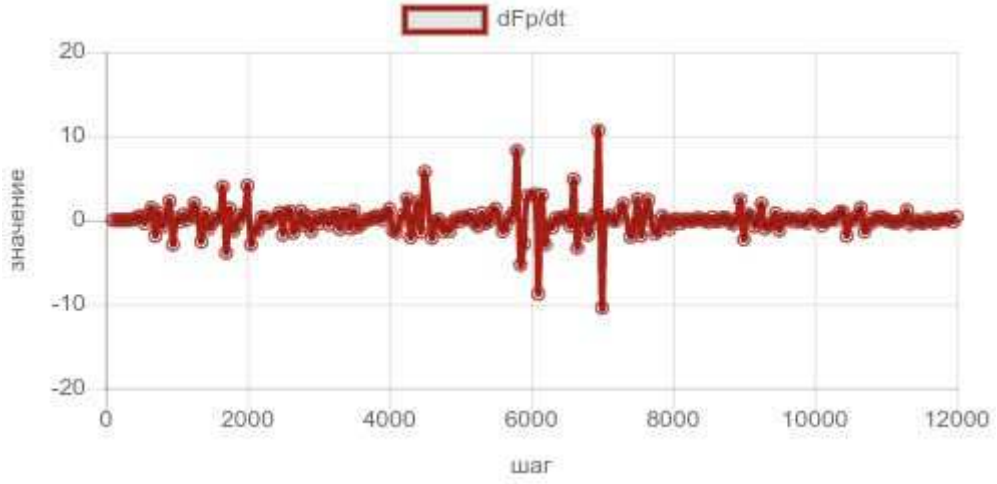

Рисунок 4.7 — Метрика количества первая производная

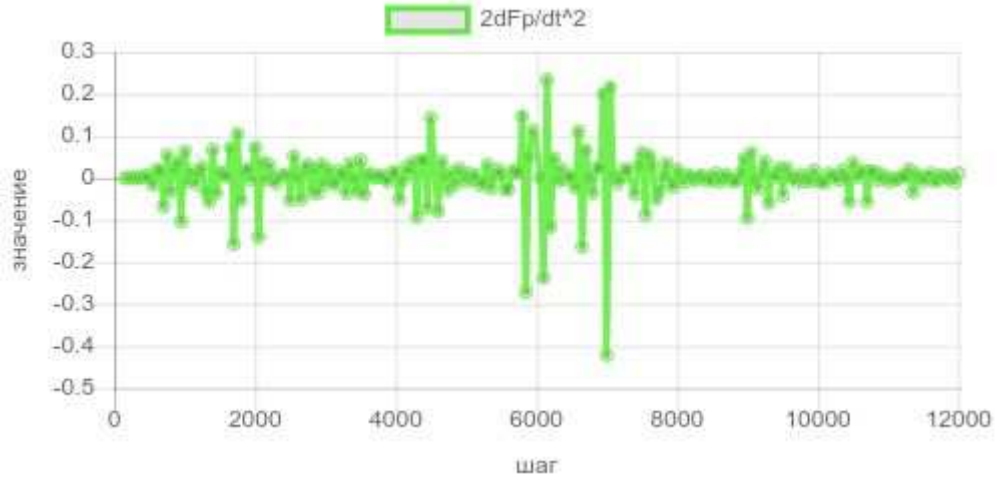

Рисунок 4.8 — Метрика количества вторая производная

На данном эксперименте можно явно увидеть лавинные процессы, для наглядности сопоставим графики на одном рисунке:

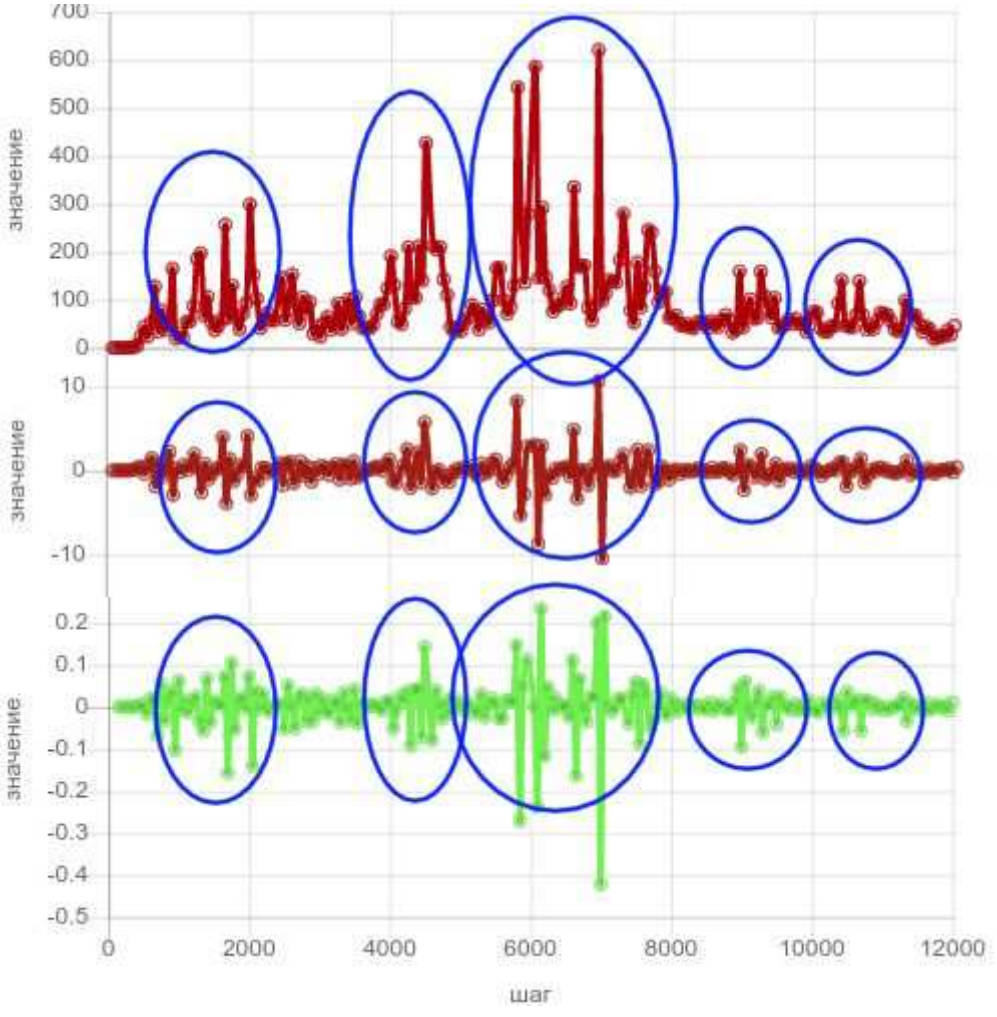

Рисунок 4.9 — Объединенный график

На рисунке 4.9 синим отмечены лавинные процессы. Как можно заметить, здесь есть 5 лавинных процессов различной формы. Для того, чтобы точно

идентифицировать процесс, будем использовать непараметрическую оценку регрессии Надарая-Уотсона.

4.1 Непараметрическая оценка регрессии

Цель регрессионного анализа [24] состоит в аппроксимации неизвестной функции по известным ее точкам.

Простейшим методом регрессионного анализа является ядерное сглаживание. Ядерная регрессия — непараметрический статистический метод, позволяющий оценить условное математическое ожидание случайной величины. Его смысл заключается в поиске нелинейного отношения между парой случайных величин X и Y [25].

В любой непараметрической регрессии условное математическое ожидание величины Y относительно величины X можно записать так: $E(Y|X) = m(X)$, где m — некая неизвестная функция.

Надарая [26] и Уотсон [27] одновременно (в 1964 году) предложили оценивать m как локально взвешенное среднее, где веса определялись бы ядром. Оценка Надарая-Уотсона:

$$\widehat{m_h}(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) \cdot y_i}{\sum_{i=1}^{n} K_h(x - x_i)}, \tag{4.1}$$

где K — ядро с шириной окна h.

Далее мы будем использовать эту оценку для сглаживания, причем для первичного сглаживания мы примем ширину окна равной 500 (10 значений исходной выборки в одном диапазоне), для вторичного сглаживания 300 (6 значений в одном диапазоне).

4.2 Идентификация лавинного процесса

Для начала взглянем на нормированный график метрики и ее производных.

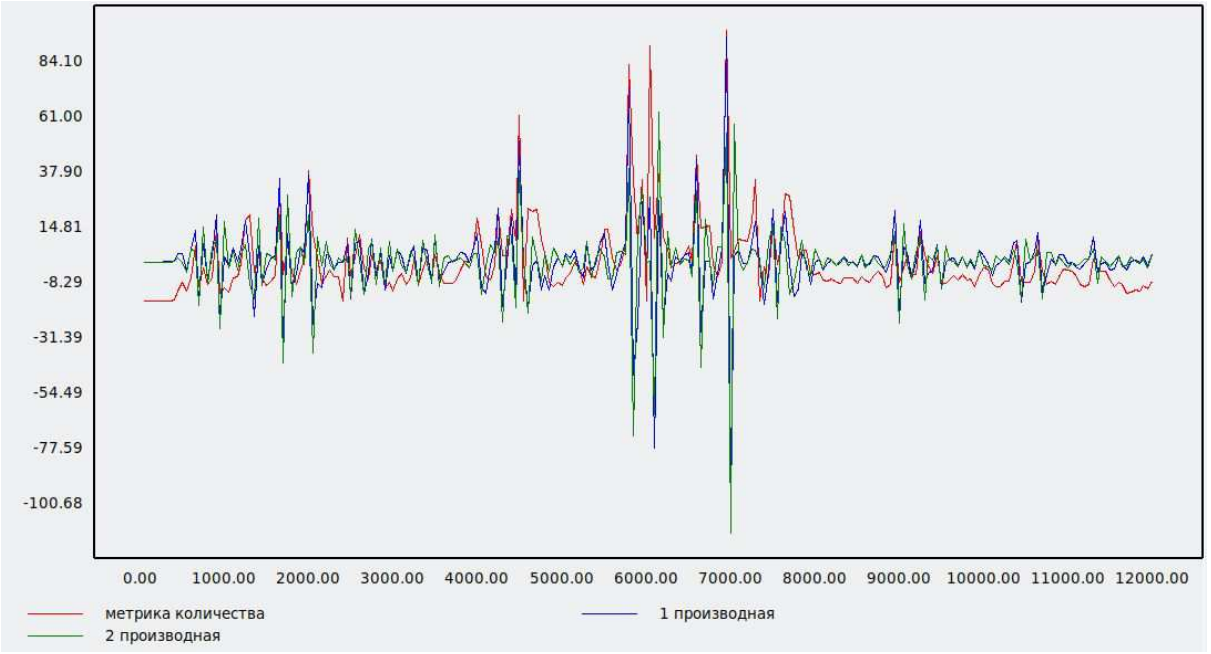

Рисунок 4.10 — Нормированный график

Поскольку отрицательные значение производных показывают спад лавинного процесса, то их нормированные отрицательные значения также будут показывать этот спад. Для значения метрики количества свойственны только положительные значения, однако, при нормировании появились отрицательные. Отрицательные являются показателем метрики в момент отсутствия лавинного процесса или его спада. Вследствие этого взглянем на график, где представлены только положительные нормированные значения.

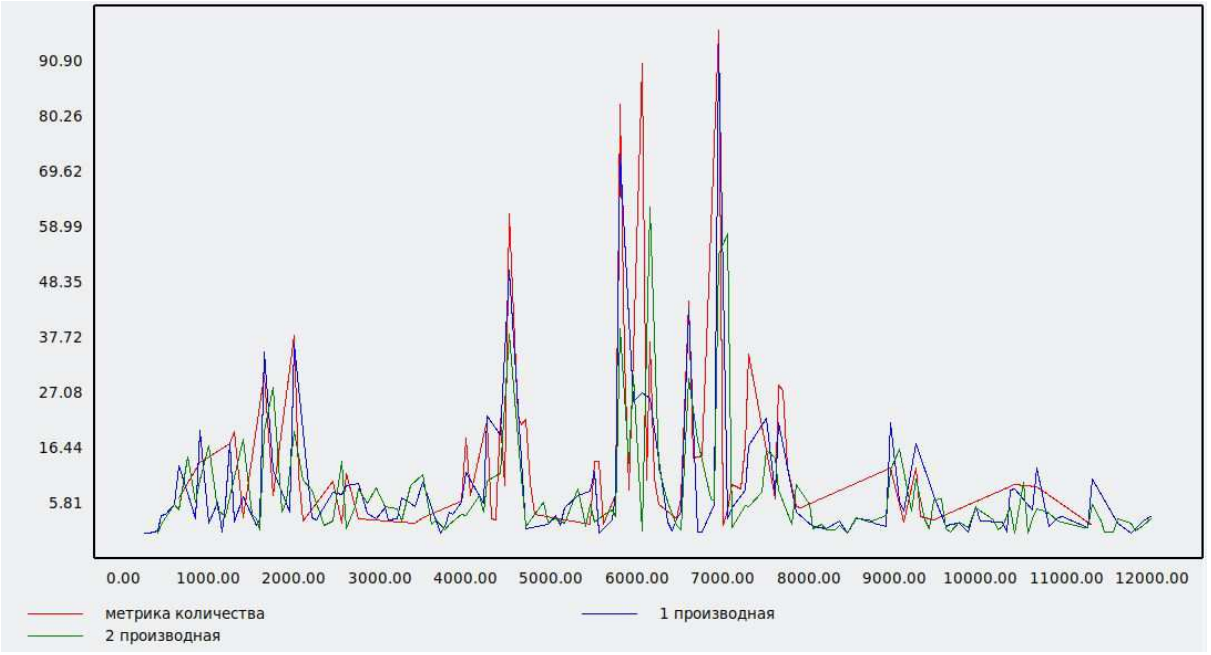

Рисунок 4.11 — Положительная часть нормированного графика

Как можно увидеть из рисунка, лавинному процессу соответствует увеличение производных как до самого процесса, так и во время процесса.

Основываясь на этом, можно произвести идентификацию лавинного процесса посредством оценки 1 и 2 производной.

Для использования непараметрической оценки регрессии Розенблатта-Парзена нужно определиться с входом, выходом модели, а также получить обучающую выборку. На вход будем подавать первую и вторую производную метрики количества. Выходом будет оценка лавины — это значение в диапазоне от 0 до 1, которое показывает близость и опасность лавины. Для того, чтобы получить обучающую выборку, произведем сглаживание значений метрики количества при помощи непараметрической регрессии. Это поможет получить сглаженный график, на котором лучше видно лавинный процесс, и вследствии чего можно получить обучающую выборку [28; 29].

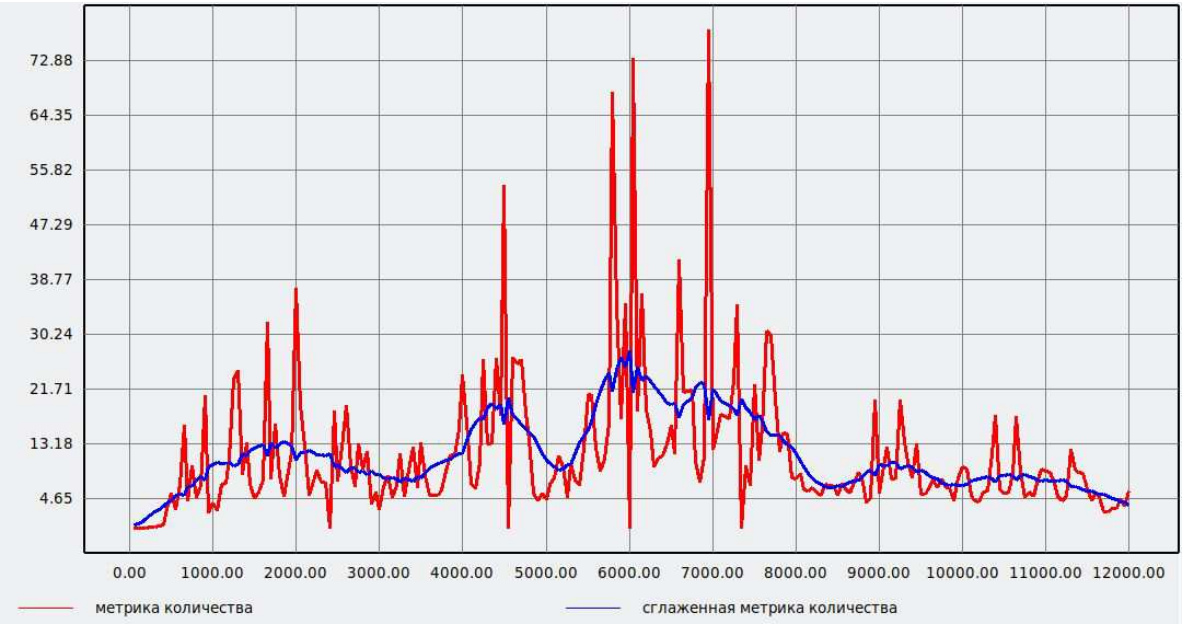

Рисунок 4.12 — Сглаженная метрика количества

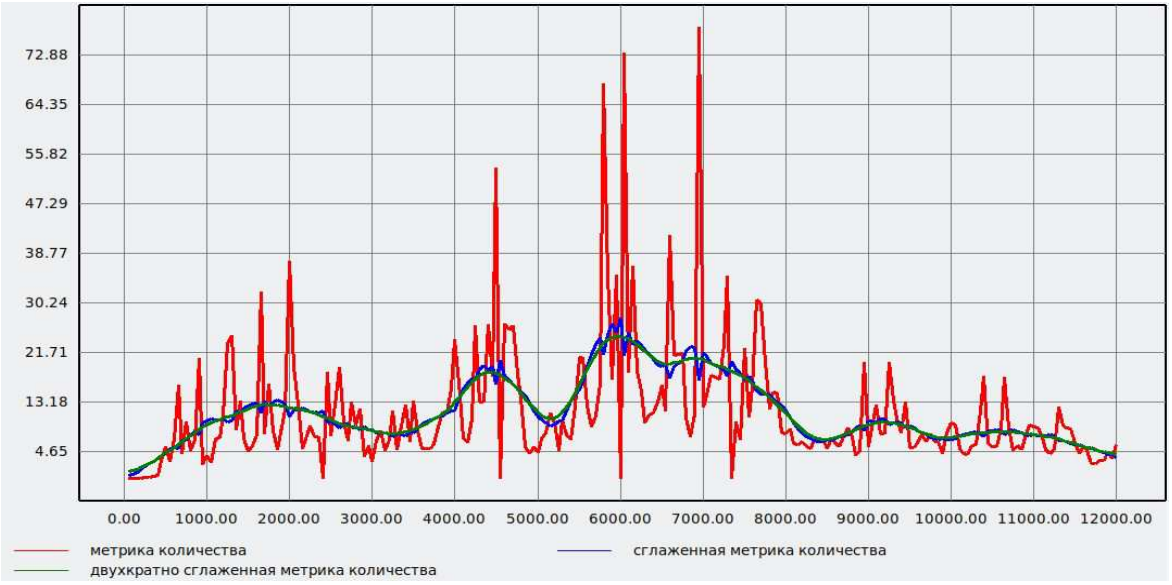

Рисунок 4.13 — Сглаженная и повторно сглаженная метрика количества

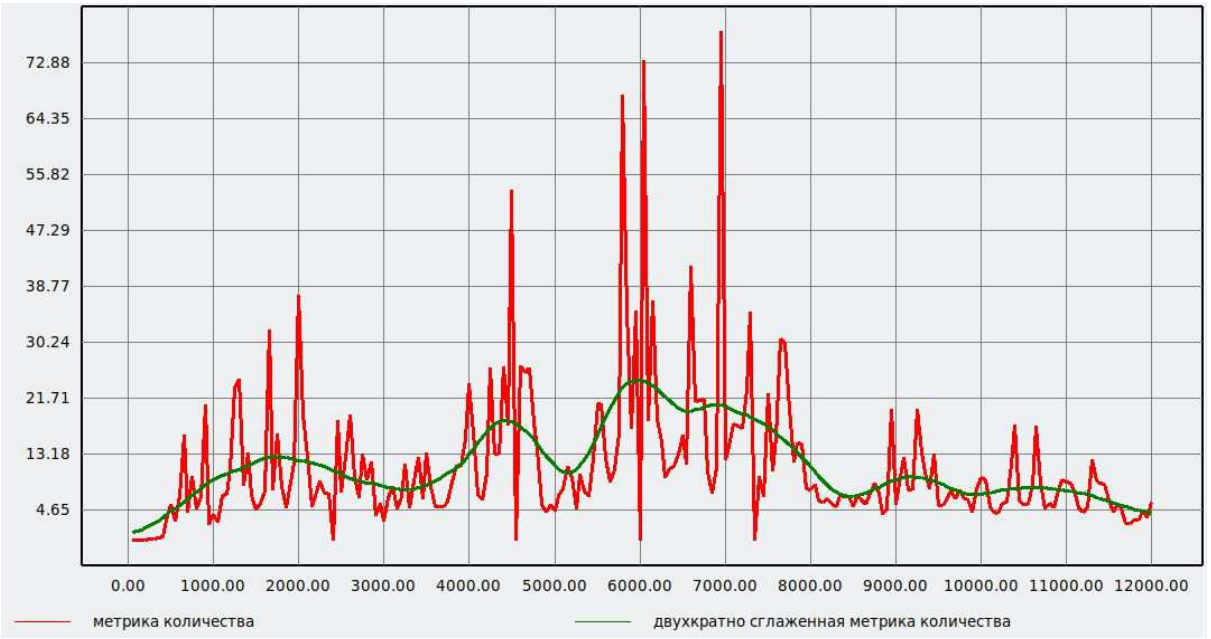

Рисунок 4.14 — Двукратно сглаженная метрика количества

Как видно по рисункам, сглаженная метрика, в особенности двукратно, наглядно демонстрирует лавинные процессы. На основании этой сглаженной метрики можно создать обучающую выборку, где начало выпуклости соответствует началу лавинного процесса, а вершина его максимальной силе. На основании этого в оценочной функции значение начала будет построено как 0, а пик как 1, значения между ними будут распределены согласно скорости возрастания лавинного процесса.

4.3 Результаты идентификации лавинных процессов

Для построения оценочной функции возьмем производную от двукратно сглаженной метрики количества.

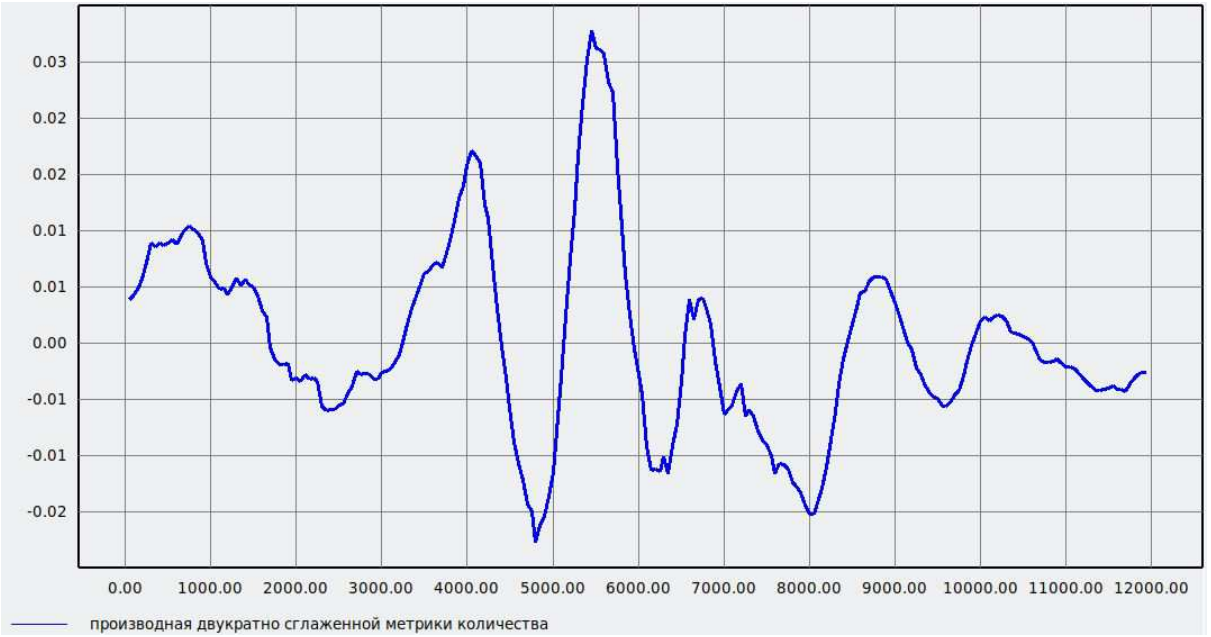

Рисунок 4.15 — Производная двукратно сглаженной метрики количества

Для улучшения оценки произведем ее сглаживание при помощи непараметрической регрессии.

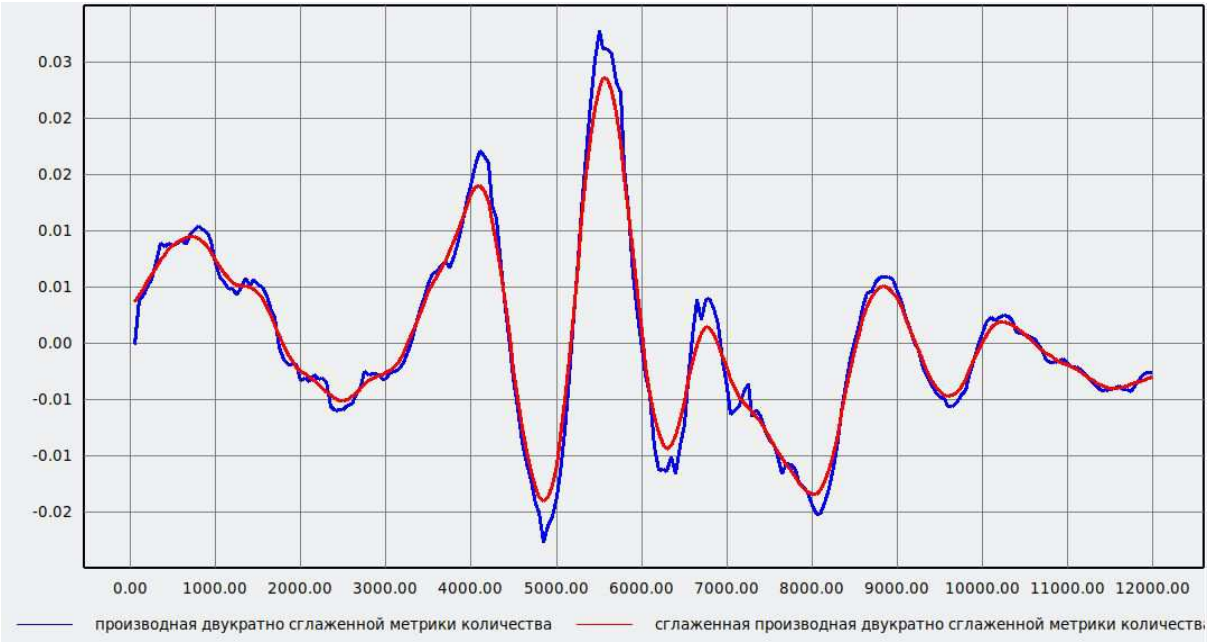

Рисунок 4.16 — Сглаженная производная двукратно сглаженной метрики количества

Поскольку нам необходимо знать лишь о наступлении лавины, а не ее спаде, возьмем только положительную половину, а также только области с возрастанием функции (положительная производная).

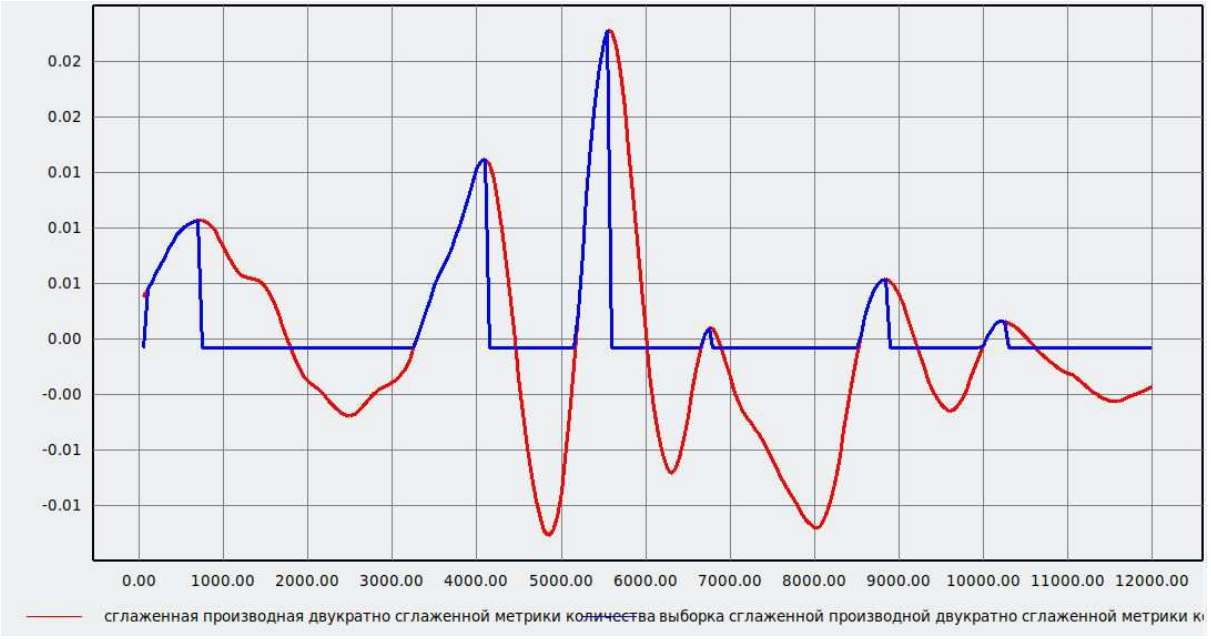

Рисунок 4.17 — Выборка сглаженной производной двукратно сглаженной функции

После нормирования получим выход для обучающей выборки.

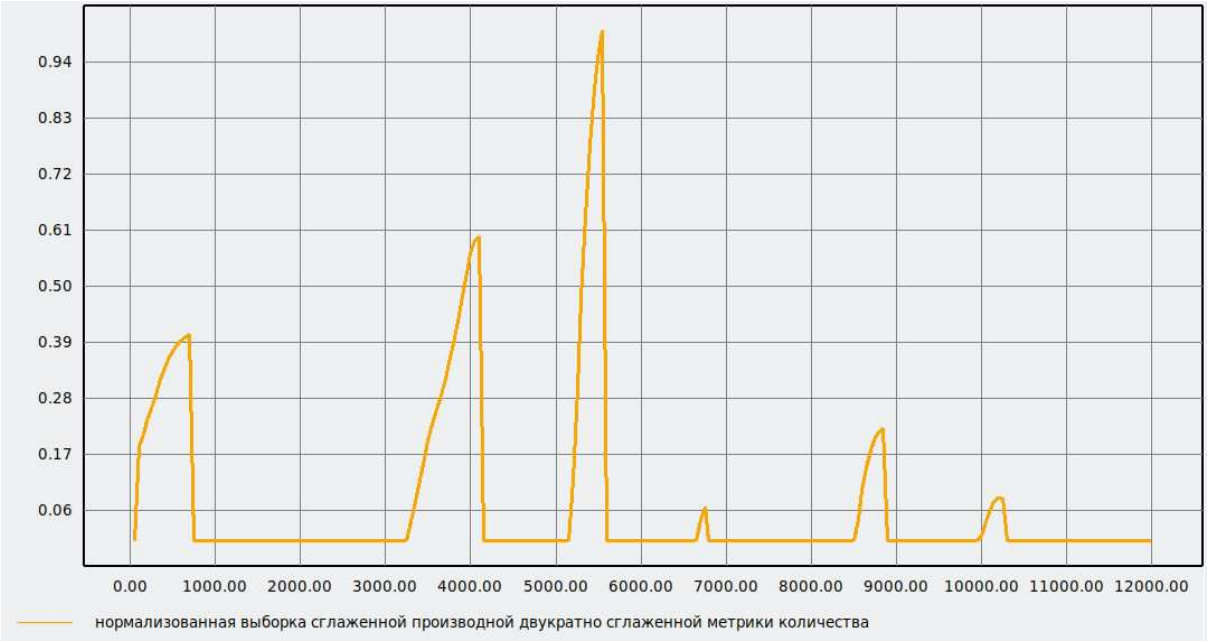

Рисунок 4.18 — Выход обучающей выборки

После того как мы получили обучающую выборку, обучим и протестируем алгоритм идентификации лавин [30].

После проведения 1376113 тестов, алгоритм успешно идентифицировал 4382602 лавин из 4382673, при этом все ошибки идентификации были пропусками лавин, ложных срабатываний не было зафиксировано. Таким образом, точность идентификации составила 99,998%. На рисунках ниже приведен пример идентификации лавин в одном из экспериментов.

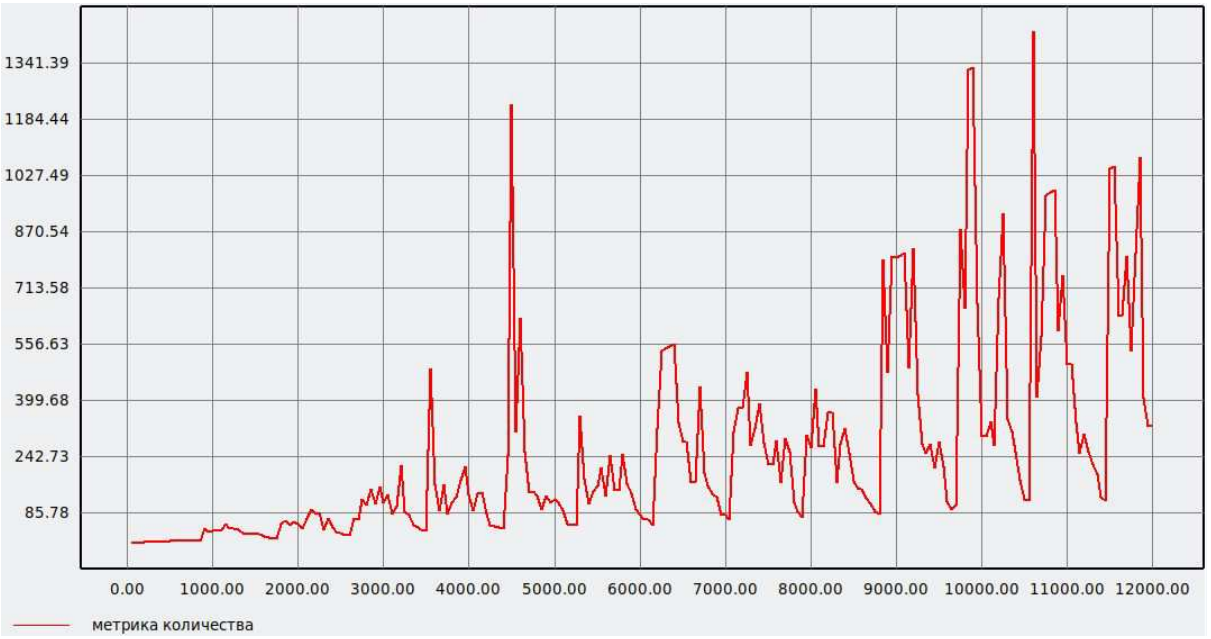

Рисунок 4.19 — Метрика количества

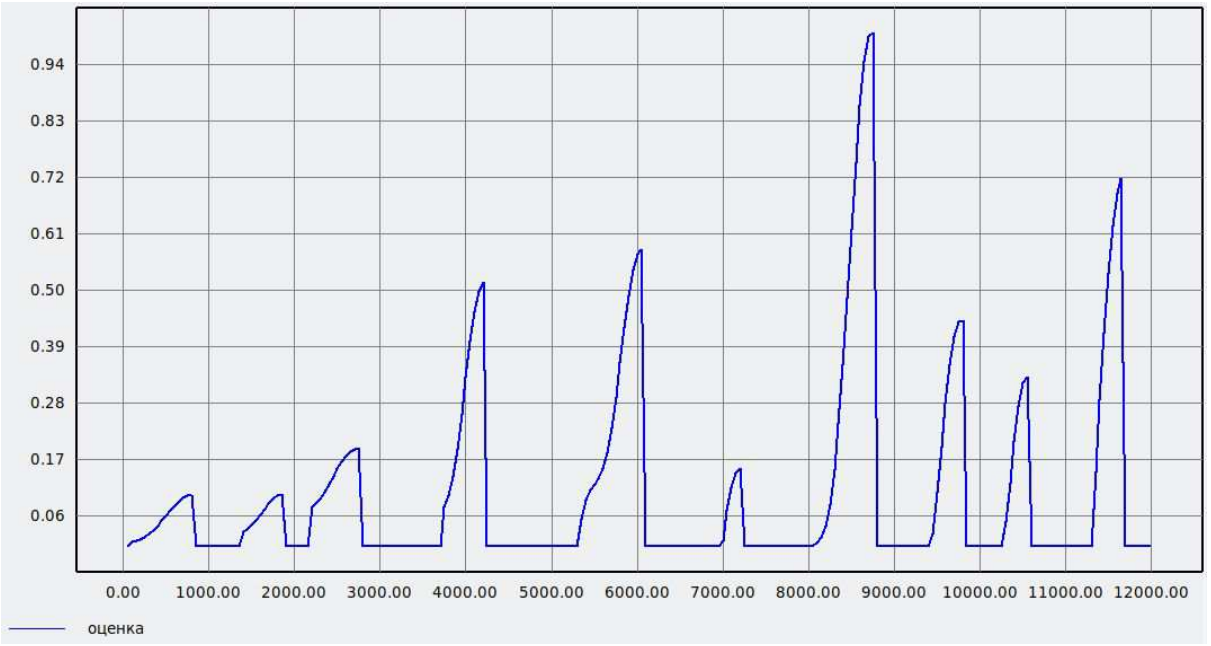

Рисунок 4.20 — Оценка идентификации

Как видно по рисунку 4.20, различные лавины оцениваются различной степенью опасности, чем больше значение оценки, тем вероятнее и сильнее лавина.

4.4 Апробация анти-лавинного алгоритма

Для основы был взят пороговый алгоритм и подвержен модификациям.

Во-первых, теперь алгоритм начинает свою работу только при идентификации лавины. Во-вторых, вместо простого изменения интенсивности

генерации монстров, алгоритм в случае нехватки монстров генерирует новых монстров (дополнительно к имеющимся генераторам), а в случае избытка снижает количество в обычном генераторе.

Для генерации монстров вычисляется разность количества героев и монстров, и, если она больше 0 (недостаток монстров), то в системе заводится новый генератор монстров с плотностью генерации равной разности, и отрицательным периодом повторения (для одноразовой генерации).

Для снижения количества алгоритм, как и прошлый, пересчитывает интенсивность генерации монстров и устанавливает ее на 100 ходов, после которых возвращается исходная.

Для проверки алгоритма было произведено 2 серии экспериментов по 100 экспериментов в каждой (малое количество обуславливается большим временем на генерации, по сравнению с предыдущей версией алгоритма). Также, в отличии от предыдущих экспериментов, были добавлены выбросы в виде временного увеличения генерации игроков (массовые приходы). Они необходимы, чтобы максимально приблизить систему к реальной. Еще одним отличием стало увеличении разброса характеристик монстров и героев.

В случае генерации без поглощения, из 100 экспериментов все 100 успешно прошли. Далее представлен пример метрик успешного эксперимента.

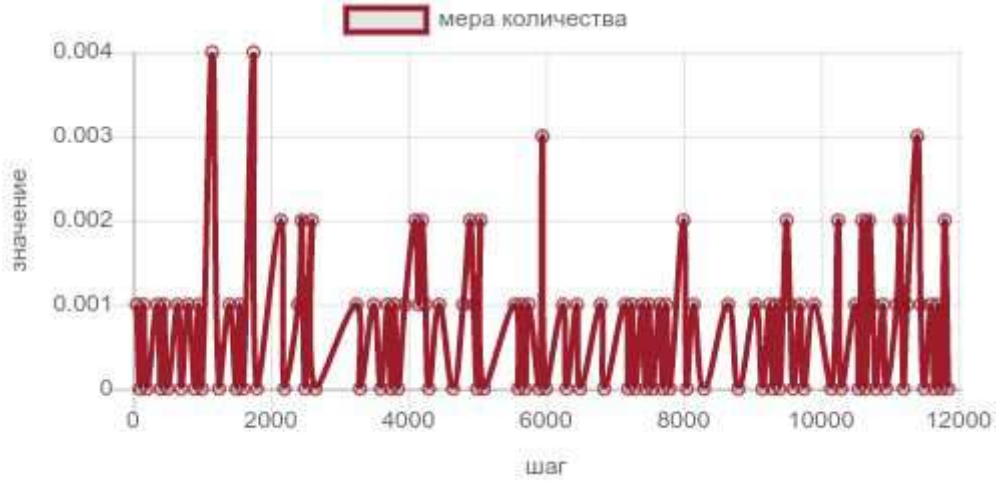

Рисунок 4.21 — Метрика количества

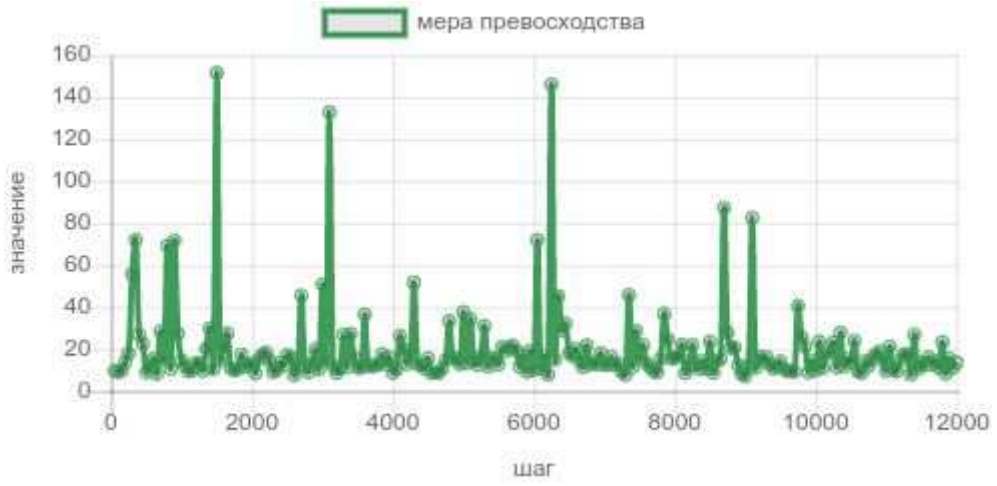

Рисунок 4.22 — Метрика превосходства

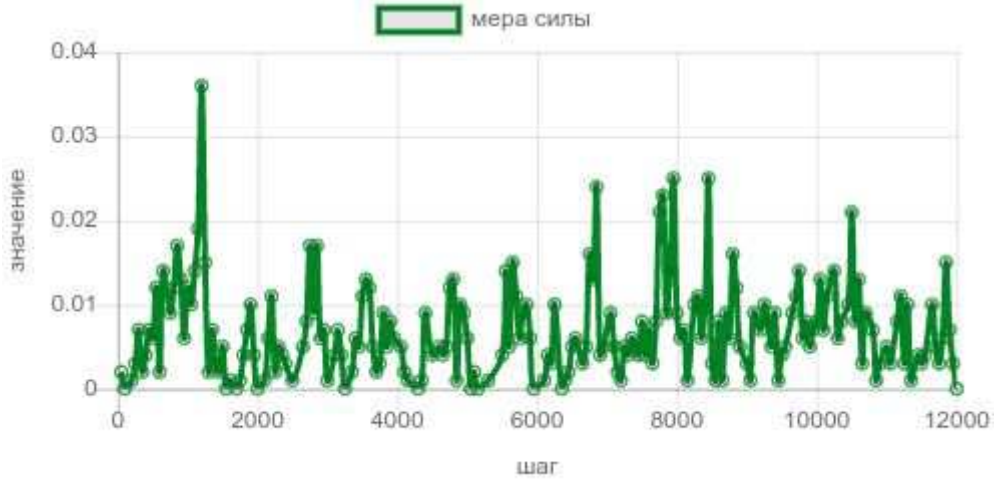

Рисунок 4.23 — Метрика силы

Как видно из рисунков, метрики силы и количества находятся в допустимых границах, а также намного меньше по значениям, чем в предыдущей версии алгоритма. Значение метрики превосходства хуже, чем у предыдущего алгоритма, но это обусловлено большим разбросом характеристик монстров и героев.

В случае генерации с поглощением, из 100 экспериментов лишь 66 были успешны. Далее приведены метрики успешного эксперимента.

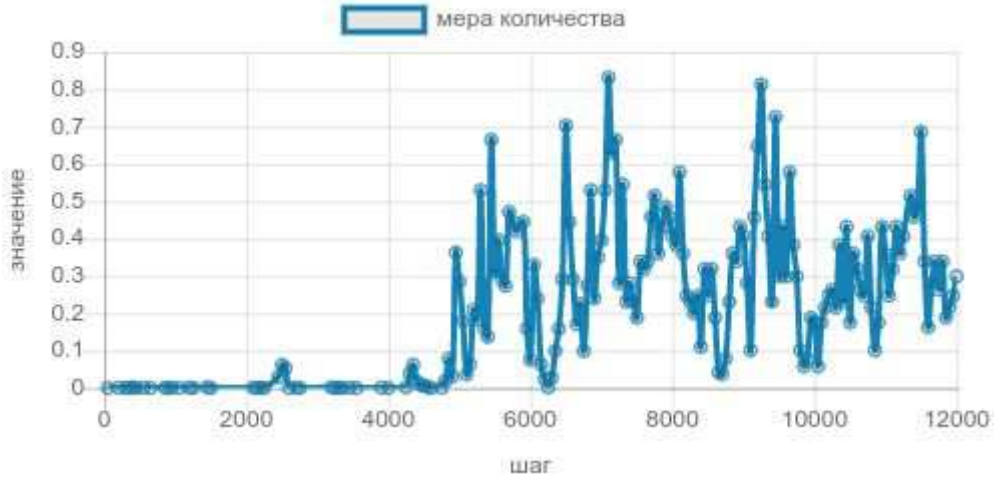

Рисунок 4.24 — Метрика количества

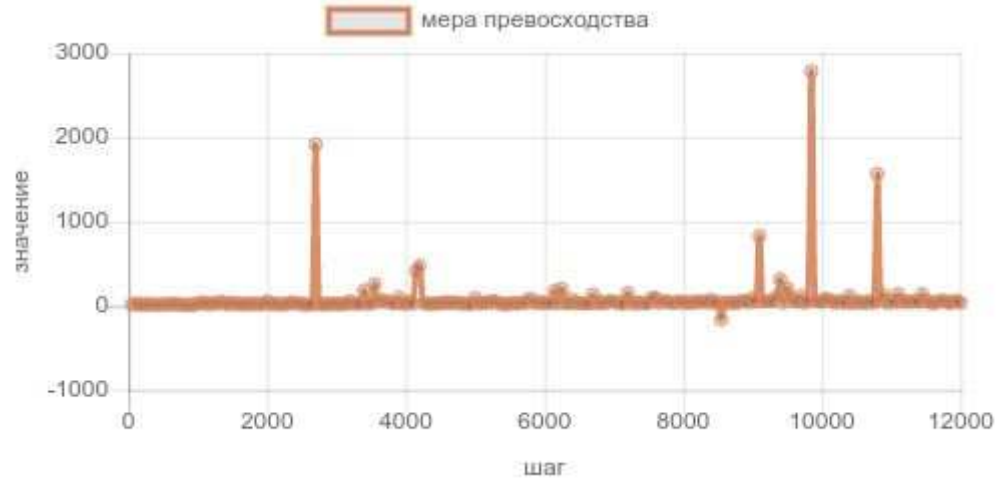

Рисунок 4.25 — Метрика превосходства

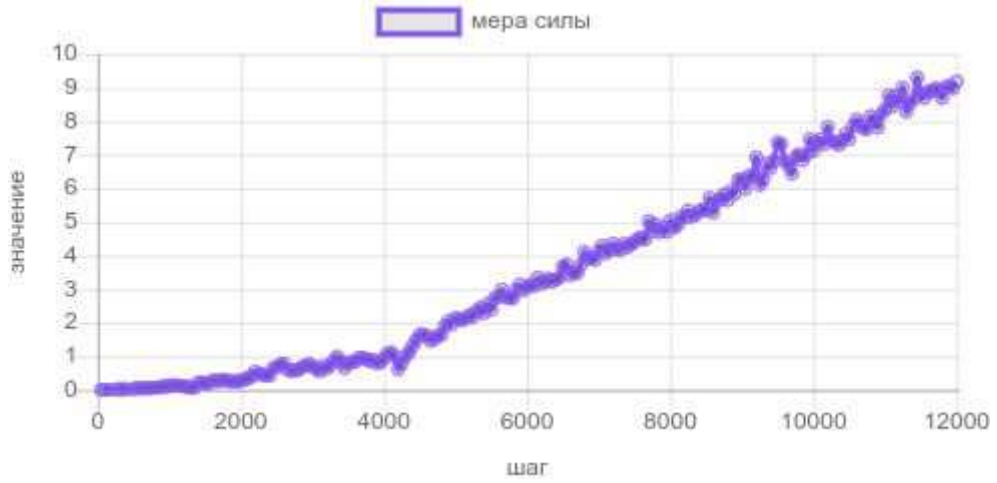

Рисунок 4.26 — Метрика силы

Как видно, метрики силы и количества находятся в допустимых границах. Всплески метрики количества и превосходства и рост метрики силы обусловлены появлением «боссов».

Теперь посмотрим на неудачные эксперименты. Для этого рассмотрим данные о количестве экспериментов, провалившихся на определенном шаге.

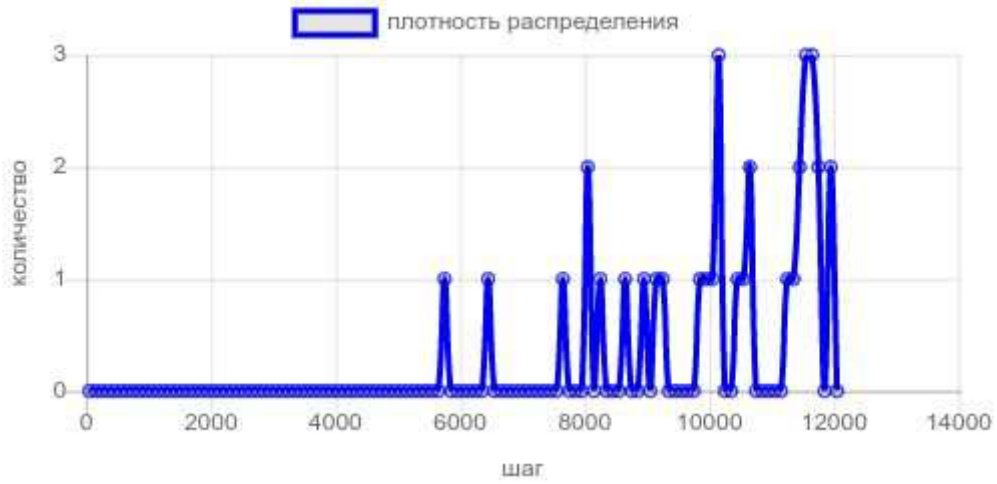

Рисунок 4.27 — Количество неудачных экспериментов

Как видно из рисунка, первый провал начинается в районе 6 000 шага, после чего количество провалов нарастает. В первую очередь это обусловлено появлением «боссов». Также на эти провалы повлияли сильные выбросы.

Для подтверждения достоверности данных произведем серию из 10000 итераций с поглощением, плотность распределения неудачных экспериментов представлена ниже.

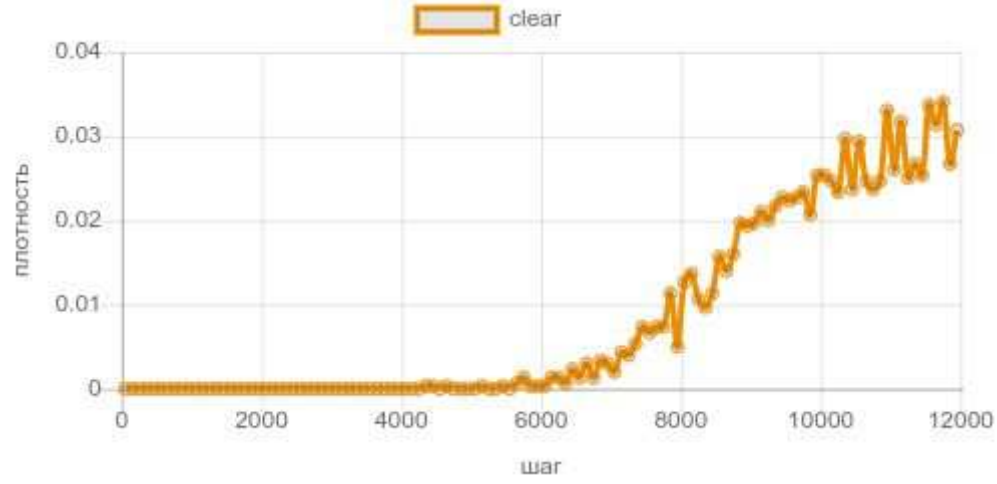

Рисунок 4.28 — Плотность неудачных экспериментов в 10 000 серии

Как видно из рисунка, увеличение числа неудач начинается приблизительно с 6 000 шага, после чего продолжает нарастать. Для этой серии экспериментов из 10 000 успешно прошли 7004, что составляет приблизительно 70% успешности.

4.5 Выводы по работе анти-лавинного алгоритма

По итогу работы анти-лавинного алгоритма можно сделать следующие выводы:

- алгоритм успешно справляется с генерацией без поглощения (небольшие игровые проекты);
- алгоритм хорошо справляется с генерацией с поглощением (большие игровые проекты);
- в случае генерации с поглощением алгоритм не может в автоматическом режиме справиться с рядом проблем — массовыми приходами игроков и появлением сильных боссов;
- для полноценной работы алгоритма на больших проектах требуется периодический контроль и вмешательство наблюдателя в систему (возможно простое внесение изменений в систему перед запланированным крупным событием, с которым алгоритм может не справиться, например, игровые акции или игровые скидки).

Таким образом, алгоритм позволил автоматизировать поддержание баланса при обычном состоянии игры и подавление малых всплесков внешних воздействий. В случае крупных всплесков, в особенности для больших систем, требуется вмешательство оператора для дальнейшей балансировки. Однако, даже в этом случае алгоритм сглаживает результаты воздействия внешнего всплеска и дает дополнительное время для принятия решения, пока система не перейдет в критическое состояние.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы, были разработаны правила игрового мира для моделирования, создана программная реализация модели этого мира, произведен анализ эффективности работы различных алгоритмов, выполнена модернизация и повторное тестирование лучшего из них.

В рамках разработки правил игрового мира были выбраны классические правила для многопользовательских игр — игроки сражаются против монстров, чтобы сделать игрового персонажа сильнее.

Для моделирования была написана программа, позволяющая работать с различными проверяемыми алгоритмами и условиями моделирования.

Для исследований было выбрано 2 вида алгоритмов — генетический и специальный анти-лавинный. Также для сравнения были произведены генерации без использования алгоритмов сохранения баланса. В результате проведения серии экспериментов с различными параметрами, наибольшую эффективность показал анти-лавинный алгоритм.

Для усовершенствования алгоритма было произведено изменение метода балансировки (способ управления генерацией новых сущностей), а также добавление идентификации лавинных процессов, для их успешного предотвращения. По результатам тестирования удалось добиться высоких результатов работы анти-лавинного алгоритма.

В результате серий экспериментов было наглядно видно, как обычные алгоритмы не справляются с поддержанием баланса. Однако после соответствующей модернизации алгоритма в том числе возможности идентификации лавин стало возможным поддержание баланса в автоматическом режиме. Однако на больших проектах из-за усложнения модели и появления большого количества внешних факторов требуется периодическое вмешательство наблюдателя. Но, благодаря работе анти-лавинного алгоритма, это вмешательство требуется редко.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Википедия — свободная энциклопедия [Электронный ресурс] : Игра — Режим доступа: https://ru.wikipedia.org/wiki/Игра.
2. Википедия — свободная энциклопедия [Электронный ресурс] : Компьютерная игра — Режим доступа: https://ru.wikipedia.org/wiki/Компьютерная_игра.
3. Википедия — свободная энциклопедия [Электронный ресурс] : Геймдизайн — Режим доступа: https://ru.wikipedia.org/wiki/Геймдизайн.
4. The Go Programming Language [Электронный ресурс] : The Go Programming Language — Режим доступа: https://golang.org/.
5. Git [Электронный ресурс] : Git — Режим доступа: https://git-scm.com/.
6. PostgreSQL [Электронный ресурс] : PostgreSQL: The World's Most Advanced Open Source Relational Database — Режим доступа: https://www.postgresql.org/.
7. Atom [Электронный ресурс] : Atom — Режим доступа: https://atom.io/.
8. Никониров, Г. В. Сохранение игрового баланса при помощи идентификации лавинообразных процессов / Г. В. Никониров // Сборник статей международной научно-практической конференции «Технологическая кооперация науки и производства: новые идеи и перспективы развития» — Казань, 23.03.2019 — Стерлитамак: АМИ, 2019. – С.33-35.
9. Gitlab [Электронный ресурс] : Файлы · develop · Григорий / magister · GitLab — Режим доступа: https://gitlab.com/gbh007/magister/tree/develop.
10. Gitlab [Электронный ресурс] : удаление отдельного класса для дуг (02125e93) · Commits · Григорий / magister · GitLab — Режим доступа: https://gitlab.com/gbh007/magister/commit/02125e93283e8a7893a514c8780a83f03b83dee2.
11. Gitlab [Электронный ресурс] : реализация работы с процессами (f7c65ffd) · Commits · Григорий / magister · GitLab — Режим доступа: https://gitlab.com/gbh007/magister/commit/f7c65ffd894520428c98800bfeb5f6d844a6378f.
12. Гладков, Л. А. Генетические алгоритмы: Учебное пособие. / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик — 2-е изд. — М: Физматлит, 2006. — 320 с.
13. Емельянов, В. В. Теория и практика эволюционного моделирования. / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик — М: Физматлит, 2003. — 432 с.
14. Скобцов, Ю. А. Основы эволюционных вычислений / Ю. А. Скобцов — Донецк: ДонНТУ, 2008. — 326 с.
15. Арнольд, В. И. Теория катастроф : монография / В. И. Арнольд. — Москва : Наука, 1990. — 128 с.
16. Gitlab [Электронный ресурс] : улучшения визуализации (eff1420d) · Commits · Григорий / magister · GitLab — Режим доступа:

https://gitlab.com/gbh007/magister/commit/eff1420d1755defd24f030d6f88fe9cb9436cf9c.

17. Gitlab [Электронный ресурс] : изменения визуализации данных (d543384d) · Commits · Григорий / magister · GitLab — Режим доступа: https://gitlab.com/gbh007/magister/commit/d543384d4c84ac65a00288edcdc74a06a36bfa2c.

18. The Go Programming Language [Электронный ресурс] : Profiling Go Programs - The Go Blog — Режим доступа: https://blog.golang.org/profiling-go-programs.

19. Gitlab [Электронный ресурс] : Файлы · develop · Григорий Никониров / magister_fast · GitLab — Режим доступа: https://gitlab.com/gbh007/magister_fast/tree/develop.

20. Gitlab [Электронный ресурс] : переход на потоковое получение данных (428a0df9) · Commits · Григорий Никониров / magister_fast · GitLab — Режим доступа: https://gitlab.com/gbh007/magister_fast/commit/428a0df95a9d49fe4a45101fd98dc1d6fac13255.

21. Gzip [Электронный ресурс] : The gzip home page — Режим доступа: https://www.gzip.org/.

22. Gitlab [Электронный ресурс] : добавление логирования изменений данных (97565b2a) · Commits · Григорий Никониров / magister_fast · GitLab — Режим доступа: https://gitlab.com/gbh007/magister_fast/commit/97565b2a668789757cb58bfa526ca9cf643e162a.

23. Gitlab [Электронный ресурс] : переход на снапшоты (6d678cd6) · Commits · Григорий Никониров / magister_fast · GitLab — Режим доступа: https://gitlab.com/gbh007/magister_fast/commit/6d678cd6bc61d5fb27b79a432b02989630b82598.

24. Дрейпер, Н. Прикладной регрессионный анализ. / Н. Дрейпер, Г. Смит — М.: Издательский дом «Вильямс». 2007.

25. Википедия — свободная энциклопедия [Электронный ресурс] : Ядерная регрессия — Режим доступа: https://ru.wikipedia.org/wiki/Ядерная_регрессия

26. Nadaraya, E. A. On Estimating Regression/ E. A. Nadaraya //Theory of Probability and its Applications. — С. 141-143.

27. Watson, G. S. Smooth regression analysis / G. S. Watson // Sankhyā: The Indian Journal of Statistics, Series A. 26 — С. 359-372.

28. Идаятова, А. К. О сглаживании некоторых траекторий лавинообразных процессов / А. К. Идаятова, А. В. Медведев // Актуальные проблемы авиации и космонавтики — Красноярск, 2012. — Т. 1, № 8 — С. 298-299.

29. Ярлыкова, Л. К. О непараметрических алгоритмах сглаживания при моделировании лавинообразных процессов / Л. К. Ярлыкова, А. В. Медведев //

Актуальные проблемы авиации и космонавтики — Красноярск, 2013. — Т. 1, № 9 — С. 345-347.

30. Идаятова, А. К. О непараметрическом моделировании лавинообразных процессов / А. К. Идаятова, А. В. Медведев // Решетневские чтения — Красноярск, 2011. — Т. 2 — С. 455-456.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Базовая кафедра «Интеллектуальные системы управления»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Акунец Ю.Л.
подпись инициалы, фамилия
«01» 04 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

27.04.03 «Системный анализ и управление»
27.04.03.02 «Системный анализ данных и технологий принятия решений»

«Сохранение игрового баланса при помощи идентификации лавинообразных процессов»

Научный руководитель _____ 26.06.19 доцент, канд. техн. наук А.А.Даничев
подпись, дата должность, ученая степень

Выпускник _____ 25.06.19 Г.В. Никониров
подпись, дата

Рецензент _____ 01.07.19 доцент, канд. техн. наук Ю.Г. Кулаичев
подпись, дата должность, ученая степень инициалы, фамилия

Нормоконтролер _____ 01.07.19 ст. преподаватель Н.Б. Позолотина
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2019