

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.04.02 «Информационные системы и технологии»
Магистерская программа	Проектирование, разработка и эксплуатация информационных систем

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

Модели и алгоритмы интеллектуального поведения персонажа компьютерной игры

Студент	_____	<u>Гордеев Сергей Денисович</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Балашова И.Ю.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)
Рецензент	_____	<u>Афонин А.Ю.</u>
(должность, место работы)	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	<u>Козлов А.Ю.</u>
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2023

Содержание

Введение.....	4
1 Модели поведения толпы.....	7
1.1 Задача моделирования поведения толпы в компьютерной игре	7
1.2 Описание толпы с помощью броуновского движения	7
1.3 Непрерывные модели описания толпы	10
1.4 Использование клеточных автоматов при описании толпы	13
1.5 Мультиагентные методы.....	16
1.6 Ньютоновская механика	20
1.7 Гибридные модели.....	22
1.8 Сравнение подходов моделирования реалистичного поведения толпы по критерию реалистичности	25
1.9 Сравнение подходов моделирования реалистичного поведения толпы по критерию вычислительной сложности.....	26
1.10 Алгоритмы передвижения агента	28
Выводы	30
2 Разработка гибридной модели поведения толпы в паническом состоянии для экшен-игры	33
2.1 Структура гибридной модели поведения толпы в ситуации паники	33
2.2 Математическая модель передвижения агентов толпы в состоянии паники.....	34
2.3 Математическая модель социального кооперирования агентов толпы в состоянии паники	36
2.4 Математический алгоритм поиска кратчайшего безопасного пути в ситуации паники.....	38
Выводы	40
3 Реализация гибридной модели толпы в компьютерной игре.....	42
3.1 Реализация передвижения агента в состоянии паники.....	42
3.2 Реализация взаимодействия агентов толпы в состоянии паники	44

3.3	Реализация поиска пути в состоянии паники	46
3.4	Тестирование гибридной модели	47
	Выводы	57
	Заключение	58
	Список использованных источников	59
	Приложение А. Листинг модели поведения толпы неигровых персонажей	63

Введение

Актуальность работы. Толпа неигровых персонажей – важный атрибут многих компьютерных игр жанра экшен, в которых герой действует в просторных локациях, наполненных массовой. Одной из частых игровых ситуаций подобных игр является ситуация паники, в которой толпа неигровых персонажей пытается спастись от источника, несущего угрозу. Чем больше приближено к реальности поведение толпы персонажей в данной ситуации, тем сильнее игрок поверит в правдоподобность игрового окружения. Поэтому моделирование поведения толпы в ситуации паники является одной из актуальных задач повышения реалистичности игрового процесса.

Существует значительное число подходов к созданию моделей поведения толпы – модели на основе клеточных автоматов [1], модели на основе ньютоновской механики [2], мультиагентные модели [3], гибридные модели [4] и др. Эти модели позволяют описать пребывание людей в местах массового скопления, а также эвакуацию людей из замкнутых пространств в чрезвычайных ситуациях. Однако применять эти модели для описания поведения неигровых персонажей в ситуации паники затруднительно, т.к. их реализация требует слишком высокой вычислительной мощности.

Целью магистерской диссертации является разработка гибридной модели поведения толпы в ситуации паники и алгоритма поиска безопасного пути для повышения реалистичности поведения неигровых персонажей компьютерной экшен-игры.

Задачи исследования. Для достижения указанной цели необходимо решить следующие задачи:

- провести анализ известных моделей поведения толпы;
- разработать модель поведения толпы неигровых персонажей компьютерной игры в ситуации паники;
- разработать алгоритм поиска кратчайшего безопасного пути при возникновении угрозы;

- разработать компьютерную экшен–игру, реализующую разработанные модели и алгоритмы поведения неигровых персонажей.

Методы исследования. Для решения поставленных задач использовались методы ньютоновской механики, мультиагентные методы, методы оптимизации.

Научная новизна работы заключается в следующем:

- разработана гибридная модель поведения толпы неигровых персонажей компьютерной игры в ситуации паники, отличительной особенностью которой является учет как физических законов передвижения объектов толпы, так и их социального кооперирования;

- разработан алгоритм поиска кратчайшего безопасного пути при возникновении угрозы, учитывающий динамически возникающие препятствия на пути следования толпы неигровых персонажей.

Практическая значимость исследований. Разработана компьютерная экшен–игра, в которой реализованы предложенные в работе модель и алгоритм поведения неигровых персонажей в ситуации паники.

Основные положения, выносимые на защиту:

- гибридная модель поведения толпы неигровых персонажей компьютерной игры в ситуации паники, учитывающая как физические законы передвижения объектов толпы, так и их социальное кооперирования;

- алгоритм поиска кратчайшего безопасного пути при возникновении угрозы, учитывающий факт наличия динамически возникающих препятствий на пути следования объектов толпы неигровых персонажей.

Апробация работы. Основные положения и результаты диссертационной работы докладывались и обсуждались на конференциях:

- IX Международная научно–практическая конференция (школы–семинара) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук»;

- X Ежегодная всероссийская межвузовская научно–практическая конференция «Информационные технологии в науке и образовании. Проблемы и перспективы».

Публикации. Основные результаты работы опубликованы в 2 печатных изданиях.

Структура и объем работы. Работа состоит из введения, трех тематических глав, заключения, списка литературы и приложений. Общий объем основного текста – 72 страницы, включая 30 рисунков. Список литературы изложен на 4 страницах и содержит 46 наименований.

1 Модели поведения толпы

1.1 Задача моделирования поведения толпы в компьютерной игре

Моделирование толпы в играх является одной из ключевых задач для достижения реалистичности игрового процесса. Для достижения этой цели, разработчики игр используют различные методы и технологии.

Одной из основных задач при моделировании толпы в играх является достижение реалистичности поведения агентов. Для этого необходимо учитывать такие факторы, как различные типы агентов, их характеристики и цели. Например, при моделировании толпы людей на улице, необходимо учитывать, что разные люди могут двигаться с разной скоростью, останавливаться на светофорах или в очередях, а также выполнять другие действия, которые являются естественными для реального мира.

Другой важной задачей является обеспечение эффективности вычислений и оптимизации производительности. При моделировании больших толп, количество агентов может достигать сотен и тысяч, что требует высокой вычислительной мощности и оптимизации производительности. Поэтому важно использовать оптимизированные алгоритмы и структуры данных, чтобы ускорить вычисления и улучшить производительность.

Также важно учитывать аспекты управления и социального кооперирования агентов в толпе. В игровых ситуациях может возникать необходимость управления движением толпы, например, чтобы агенты не блокировали друг друга, не сталкивались или не пересекались. Поэтому важно учитывать такие аспекты, как маршрутизация, управление плотностью толпы и обнаружение столкновений.

1.2 Описание толпы с помощью броуновского движения

Модель броуновского движения [5] – это стохастическая модель движения, которая описывает случайное перемещение частиц во время столкновений с молекулами окружающей среды. Эта модель может быть применена в

моделировании толпы, где объекты двигаются по случайным траекториям, сталкиваясь друг с другом и с препятствиями.

Одним из преимуществ модели броуновского движения является ее простота и универсальность. Она может быть легко применена в различных ситуациях и может быть адаптирована для разных целей. Также модель броуновского движения позволяет моделировать случайное поведение объектов, что создает реалистичность в движении толпы.

Для моделирования толпы с помощью модели броуновского движения используется алгоритм Стеринга [6]. Он основан на принципе, что каждый объект в толпе должен следовать определенным правилам, чтобы избежать столкновений с другими объектами. Принципы, которые применяются в алгоритме Стеринга, включают избегание столкновений, следование за лидером, приближение к цели и другие.

Избегание столкновений – это основной принцип в алгоритме Стеринга. Он гарантирует, что объекты в толпе не сталкиваются друг с другом. Этот принцип достигается путем определения зоны, которую объект не должен пересекать. Если объект приближается к этой зоне, он должен изменить свое направление движения.

Следование за лидером – этот принцип применяется, когда в толпе есть объект, который движется в определенном направлении. Остальные объекты должны следовать за лидером, чтобы избежать столкновений.

Приближение к цели – этот принцип применяется, когда объект должен двигаться к определенной точке в пространстве. Объекты в толпе используют модель броуновского движения, чтобы подбираться к цели, совершая случайные шаги в ее направлении.

Все эти принципы объединены в алгоритме Стеринга, который позволяет моделировать движение толпы с помощью модели броуновского движения. Реализация алгоритма Стеринга в игре позволяет создавать реалистичные толпы, которые движутся по пространству, избегая столкновений и приближаясь к цели.

Минусы использованной модели броуновского движения:

1. Непредсказуемость. Броуновское движение делает перемещение персонажа непредсказуемым и неопределенным. Это может затруднить понимание игроком того, как его действия влияют на движение персонажа и как найти оптимальный путь к выходу.
2. Отсутствие гарантированного решения. Броуновское движение не обеспечивает гарантированного нахождения оптимального пути к выходу из лабиринта. Персонаж может блуждать вокруг и возвращаться в уже исследованные области, что может затянуть процесс игры и вызвать чувство неудовлетворенности у игрока.
3. Отсутствие контроля. Игрок не имеет прямого контроля над движением персонажа, так как он перемещается случайным образом. Это может вызвать чувство беспомощности у игрока, особенно если он хочет принять активное участие в нахождении пути к выходу.
4. Случайность вместо стратегии. Использование броуновского движения может отрицательно сказаться на разработке стратегии игры. Игроку может быть сложно разработать стратегию нахождения пути, так как движение персонажа не зависит от его решений и действий.
5. Возможность застревания. Из-за случайного движения персонажа с помощью броуновского движения, существует вероятность, что персонаж может застрять в тупиковой ситуации или циклическом перемещении, не найдя выход из лабиринта.

Пример игры с использованием броуновского движения – «Lost in the Maze». Это пазл-головоломка, в которой игроку предстоит найти выход из запутанного лабиринта. Цель игры состоит в том, чтобы провести главного персонажа к выходу, избегая препятствий и находя оптимальный путь с помощью броуновского движения.

На рисунке 1 изображен скриншот из игры «Lost in the Maze».

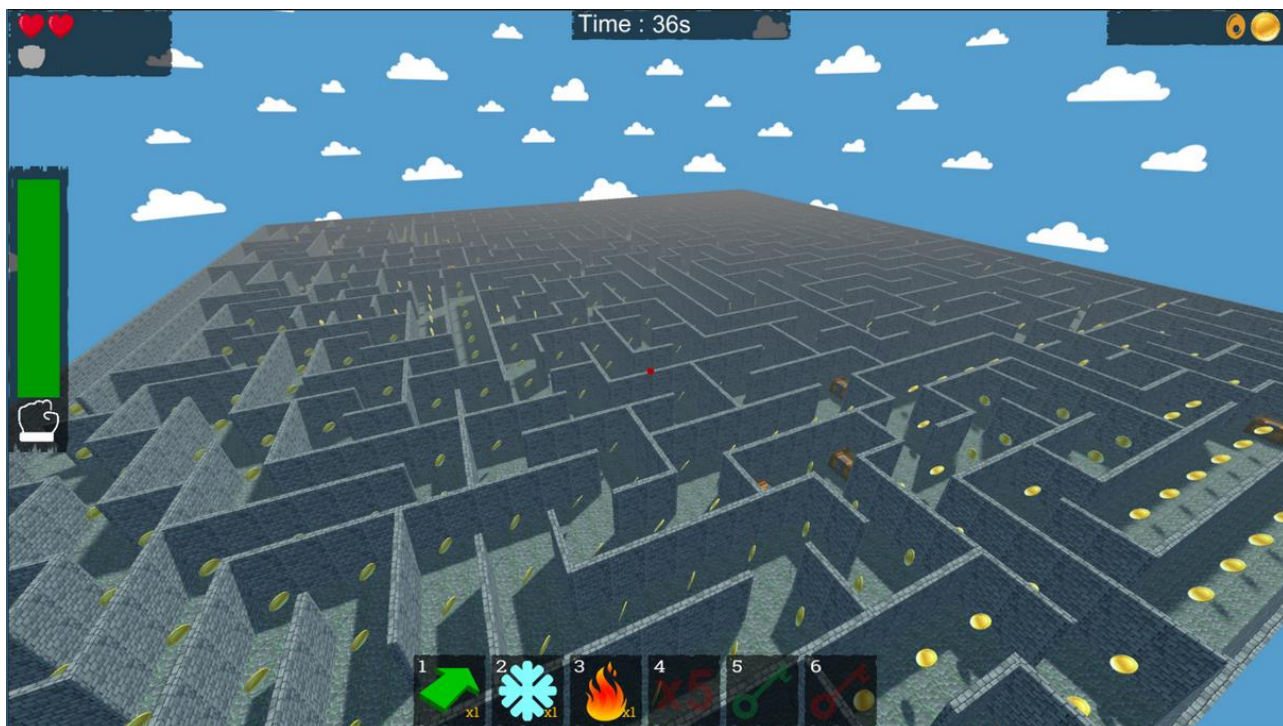


Рисунок 1 – Скриншот из игры «Lost in the Maze»

В данной игре персонаж движется случайным образом по лабиринту, используя броуновское движение. Каждый шаг персонажа определяется случайным вектором скорости и временем, что позволяет ему перемещаться в случайном направлении и на разные расстояния. Персонажу предстоит принимать решения о выборе пути, основываясь на текущем состоянии лабиринта и своих предыдущих перемещениях.

Таким образом при использовании модели броуновского движения необходимо учитывать минусы этой модели и искать альтернативные модели передвижения, которые обеспечивают более предсказуемое, эффективное и направленное поведение персонажа.

1.3 Непрерывные модели описания толпы

Непрерывные модели [7] используют непрерывное пространство и учитывают движение каждого индивидуума в толпе как непрерывный процесс.

Одним из наиболее распространенных типов непрерывных моделей является модель гидродинамики толпы. Она базируется на уравнениях Навье–

Стокса [8], которые используются для описания течения жидкостей и газов, и модифицированы для описания движения индивидуумов в толпе.

В модели гидродинамики толпы каждый индивидуум рассматривается как источник скорости, который влияет на скорость и направление движения других индивидуумов в его окрестности. Эта модель позволяет учитывать более сложные взаимодействия между индивидуумами, такие как взаимодействия, связанные с обходом препятствий и другими факторами, которые могут влиять на движение толпы.

Другой подход к непрерывному моделированию толпы – это использование моделей машинного обучения. Эти модели основаны на обучении с учителем или без учителя и могут быть использованы для предсказания движения толпы на основе данных о прошлом движении и других факторах.

Преимущества использования непрерывных моделей в моделировании толпы включают более точное моделирование взаимодействия между индивидуумами и способность учитывать различные факторы, такие как плотность толпы и размеры препятствий. Однако, эти модели могут быть более сложными и требуют большего количества вычислительных ресурсов.

Минусы использованной модели непрерывных моделей:

1. Высокая сложность. Реализация непрерывных моделей требует от разработчика глубокого понимания физических законов и математических алгоритмов. Это может усложнить разработку игры и потребовать дополнительных усилий и ресурсов.

2. Ресурсоемкость. Использование непрерывных моделей может потребовать значительных вычислительных ресурсов для реалистичного моделирования физических эффектов. Это может оказать негативное влияние на производительность игры, особенно на устройствах с ограниченными ресурсами.

3. Сложность балансировки. Использование непрерывных моделей может усложнить балансировку уровней и управление сложностью игры.

Физические эффекты могут сделать игру слишком сложной или, наоборот, слишком легкой, требуя тщательной настройки параметров и тестирования.

4. Непредсказуемость. Непрерывные модели могут привести к неконтролируемым и непредсказуемым ситуациям в игре. Например, воздействие гравитации и физические эффекты могут вызывать неожиданные движения и поведение персонажа, что может создать некоторую неопределенность и затруднить планирование и стратегию игрока.

5. Усложнение разработки. Реализация непрерывных моделей требует дополнительного времени и усилий со стороны разработчиков. Необходимо провести тщательное тестирование и отладку, чтобы убедиться в корректности и стабильности работы модели.

Пример игры с использованием непрерывных моделей – «Gravity Runner». Это платформер, в котором игроку предстоит управлять персонажем, преодолевая препятствия и собирая бонусы в гравитационной среде. Цель игры заключается в прохождении уровней, используя изменение гравитации и активное перемещение персонажа.

На рисунке 2 изображен скриншот из игры «Gravity Runner»

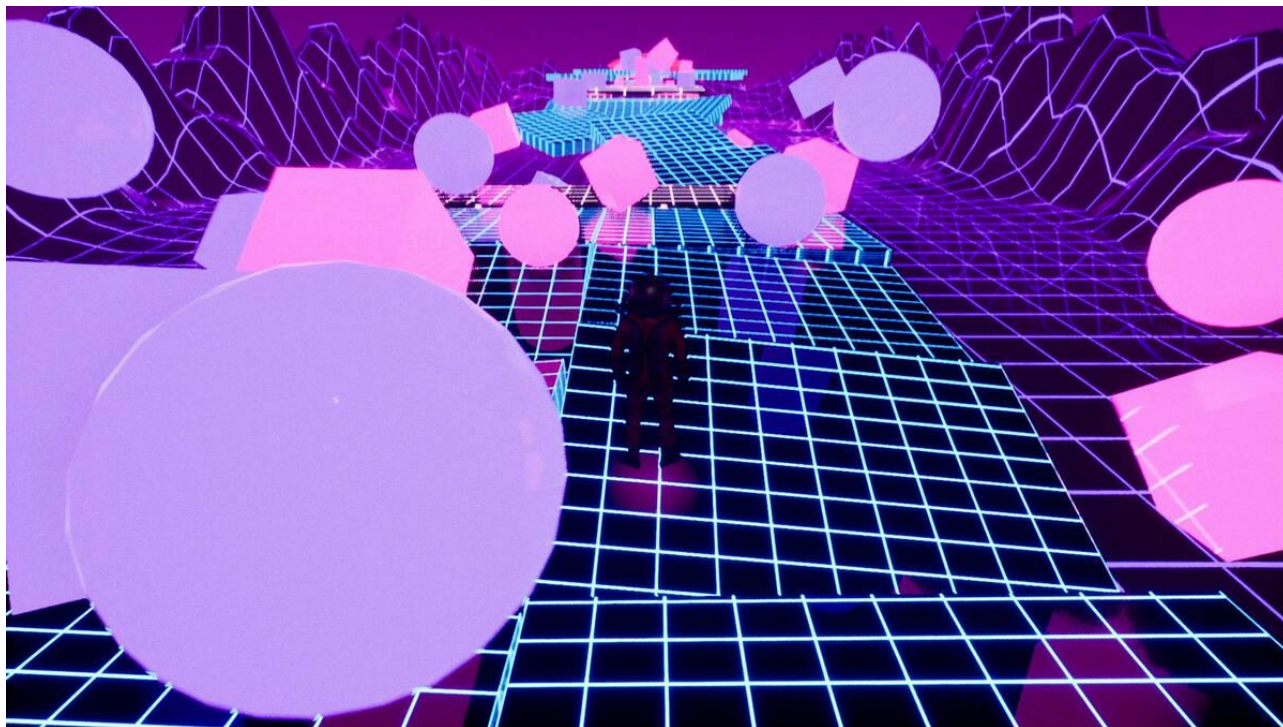


Рисунок 2 – Скриншот из игры «Gravity Runner»

В данной игре используется модель непрерывных моделей для определения перемещения персонажа и его взаимодействия с гравитационной средой. Персонаж может изменять свою гравитацию и направление движения, воздействуя на окружающую среду и преодолевая препятствия. Модель непрерывных моделей обеспечивает плавность движения и реалистичные физические эффекты в игре.

В целом, использование непрерывных моделей в играх может создавать реалистичный и динамичный геймплей, но при этом сопряжено с определенными сложностями и ограничениями.

1.4 Использование клеточных автоматов при описании толпы

Клеточные автоматы (КА) – это математическая модель, которая представляет собой сетку клеток, где каждая клетка может находиться в одном из конечного числа состояний, а обновление состояния каждой клетки зависит от состояния ее соседей. В моделировании толпы КА используются для описания движения каждого индивидуума внутри толпы.

В моделировании толпы с использованием КА, каждый индивидуум представляется отдельной клеткой на сетке. Состояние каждой клетки зависит от состояния ее соседей, что позволяет учитывать взаимодействие индивидуумов друг с другом. Например, если два индивидуума находятся слишком близко друг к другу, то их состояние изменяется, чтобы они начали двигаться в разные стороны.

Клеточные автоматы могут быть использованы для моделирования различных видов поведения толпы, таких как формирование очередей или поиск пути через сложные лабиринты. Например, чтобы моделировать формирование очередей, можно определить определенный паттерн изменения состояния клеток, который приводит к образованию линии из клеток.

Одним из преимуществ моделирования толпы с использованием КА является то, что такой подход позволяет учитывать взаимодействие индивидуумов друг с другом, что может приводить к более реалистичному

поведению толпы. Кроме того, КА могут быть легко распараллелены, что позволяет обрабатывать большие объемы данных быстрее, чем при использовании других подходов к моделированию толпы.

Минусы использованной модели клеточных автоматов:

1. Ограниченность пространства. Клеточные автоматы оперируют на сетке ячеек, что может ограничить возможности свободного движения и интерактивности игры. В отличие от моделей, основанных на физике, где объекты могут свободно перемещаться в пространстве, клеточные автоматы ограничены определенными позициями на сетке.

2. Упрощенная модель поведения. Клеточные автоматы работают на основе заданных правил поведения и соседства. В некоторых случаях это может приводить к упрощению моделирования сложных физических или интеллектуальных процессов. Клеточные автоматы не всегда могут достаточно точно воспроизвести реальное поведение объектов.

3. Ограниченная интерактивность. Взаимодействие игрока с клеточными автоматами ограничено размещением и настройкой автоматов на игровом поле. Возможности прямого управления или влияния на автоматические действия клеток могут быть ограничены.

4. Сложность настройки и балансировки. Работа с клеточными автоматами требует от разработчиков тщательной настройки правил поведения и параметров, чтобы достичь желаемого баланса игры. Неправильное настраивание правил может привести к нежелательным или несбалансированным результатам.

5. Низкая визуальная детализация. Использование клеточных автоматов может ограничить визуальную детализацию игры. Каждая клетка обычно представляется одним символом или спрайтом, что может не позволить достичь высокого уровня детализации и реализма графики.

Пример игры с использованием клеточных автоматов – «Cellular Defense». Это стратегическая игра, где игроку необходимо защищать свою территорию, используя клеточные автоматы. Игрок размещает различные типы клеточных

автоматов на игровом поле, которые автоматически взаимодействуют с вражескими клетками и защищают территорию от нападения.

На рисунке 3 изображен скриншот из игры «Cellular Defense».



Рисунок 3 – Скриншот из игры «Cellular Defense»

В данной игре используется модель клеточных автоматов для определения поведения и взаимодействия между клетками. Каждая клетка имеет свое состояние и правила поведения, которые определяют ее действия на основе состояний соседних клеток. Модель клеточных автоматов позволяет создавать разнообразные стратегии защиты и динамичную игровую среду.

В целом, моделирование толпы с использованием клеточных автоматов – это один из возможных подходов, который может быть эффективным при моделировании некоторых видов поведения толпы.

Некоторые исследователи предложили различные модификации клеточных автоматов для учета более сложных взаимодействий между индивидуумами в толпе. Например, одна из таких модификаций – это введение вероятностных правил для изменения состояния клеток, которые зависят не только от состояний соседних клеток, но и от случайных факторов, таких как шумы и случайные события.

Также были предложены и другие методы моделирования толпы, которые используют клеточные автоматы в комбинации с другими подходами, такими как моделирование сил и потенциалов или методы искусственного интеллекта.

Например, можно использовать КА для определения основных тенденций движения толпы, а затем применять методы искусственного интеллекта для определения индивидуальных характеристик каждого индивидуума.

Несмотря на некоторые ограничения, использование клеточных автоматов в моделировании толпы может быть полезным инструментом для разработки игр и симуляторов, а также для исследования социального поведения людей в различных ситуациях. Более того, такой подход может быть применен в различных областях, таких как градостроительство, планирование транспортных средств и проектирование зданий для оптимизации потоков людей и повышения безопасности.

1.5 Мультиагентные методы

Мультиагентные методы представляют собой подход к моделированию поведения толпы, где каждый индивидуум в толпе представлен отдельным агентом, который может взаимодействовать с другими агентами и окружающей средой. Мультиагентные модели являются более гибкими и адаптивными, поскольку они могут учитывать не только физические, но и социальные, психологические и культурные факторы, которые влияют на поведение людей.

Применение мультиагентных методов в моделировании толпы позволяет создавать более реалистичные сценарии поведения людей в различных условиях. Такие модели могут использоваться для анализа взаимодействия между людьми, для предсказания возможных последствий различных сценариев поведения и для оценки эффективности мер по управлению толпой.

Одним из примеров мультиагентных методов является модель «Social Force Model», которая была разработана Dirk Helbing и Peter Molnar [9]. В этой модели каждый агент в толпе представляет собой объект, обладающий массой и скоростью, который может взаимодействовать с другими агентами и окружающей средой. Для каждого агента определяются такие параметры, как желаемая скорость, желаемое направление движения и предпочитаемое расстояние до соседних агентов.

В модели «Social Force Model» взаимодействие между агентами и окружающей средой основывается на концепции «социальных сил». Эти силы могут быть положительными или отрицательными и включают в себя такие факторы, как взаимодействие между агентами, взаимодействие между агентами и стенами, а также факторы, связанные с желаемой скоростью и направлением движения.

Модель «Social Force Model» была успешно применена для моделирования различных сценариев поведения толпы, таких как движение внутри зданий, эвакуация людей из зданий в случае чрезвычайных ситуаций, движение на транспорте и т.д. Эта модель также может использоваться для определения оптимальных путей движения

Мультиагентные модели позволяют реализовать сложные поведенческие паттерны и учитывать большое количество факторов, которые влияют на поведение толпы. Однако, такие модели могут быть достаточно сложны в реализации и требуют большого количества вычислительных ресурсов. Также, для корректного моделирования необходимо точно определить все параметры модели, что может потребовать большого количества времени и средств.

Одним из основных преимуществ мультиагентных моделей является их способность учитывать взаимодействия и зависимости между агентами и объектами окружающей среды. Также они могут быть легко расширены и модифицированы для адаптации к конкретным условиям и задачам.

Минусы мультиагентной модели:

1. Сложность разработки. Реализация мультиагентных систем может быть сложной задачей, требующей детального проектирования, разработки и отладки. Взаимодействие и координация между множеством агентов может потребовать значительных усилий и ресурсов.

2. Высокие вычислительные затраты. Работа с мультиагентными системами может требовать значительных вычислительных ресурсов. Обработка действий и взаимодействий между множеством агентов может быть ресурсоемкой задачей, особенно при увеличении размера и сложности среды.

3. Проблемы с масштабируемостью. Управление и координация большого числа агентов в игре может представлять сложности с масштабируемостью. При увеличении количества агентов возникают проблемы с производительностью и эффективностью работы системы.

4. Сложность балансировки. Балансировка поведения и взаимодействия агентов может быть сложной задачей. Необходимо учесть различные факторы, чтобы достичь желаемого геймплея и предотвратить нежелательное или несбалансированное поведение агентов.

5. Непредсказуемость и эмерджентность. Мультиагентные системы могут проявлять свойства эмерджентности, когда сложное поведение и взаимодействие агентов возникают из простых правил. Это может привести к непредсказуемому и неожиданному поведению агентов, что может быть сложно контролировать и управлять.

При создании игр с открытым миром, где игрок может перемещаться по большому количеству локаций и взаимодействовать с большим числом NPC, важно создать иллюзию живой и динамичной игровой среды, где NPC будут вести себя естественно и предсказуемо.

Мультиагентные методы могут быть использованы для создания алгоритмов поведения NPC в игровой среде, учитывая их цели, взаимодействия друг с другом и с игроком, а также окружающую среду.

Кроме того, мультиагентные методы могут быть использованы для создания умных NPC, которые могут адаптироваться к поведению игрока и принимать решения в реальном времени на основе ситуации в игровой среде. Например, NPC могут анализировать действия игрока и реагировать на них соответствующим образом, изменяя свое поведение и цели.

Пример игры с использованием мультиагентных методов – «Urban Empire». Это стратегическая игра, где игроку предоставляется возможность управлять развитием и функционированием городской среды. Игрок должен строить и управлять инфраструктурой, размещать различные типы агентов

(жителей, транспорта, предприятий) и принимать решения, влияющие на развитие и благосостояние города.

На рисунке 4 изображен скриншот из игры «Urban Empire».



Рисунок 4 – Скриншот из игры «Urban Empire»

В данной игре используется модель мультиагентных методов для моделирования поведения и взаимодействия различных агентов в городской среде. Каждый агент имеет свои характеристики, цели и правила поведения, которые определяют его действия и взаимодействие с другими агентами. Модель мультиагентов позволяет создать динамичную и реалистичную городскую симуляцию, где различные агенты влияют на друг друга и на окружающую среду.

Важным аспектом применения мультиагентных методов в играх является их оптимизация для работы в реальном времени. Так как игры работают в режиме реального времени, алгоритмы поведения NPC должны работать быстро и эффективно, чтобы не создавать задержек в игровом процессе.

Таким образом, мультиагентные методы являются важным инструментом для создания реалистичного и живого поведения толпы в играх. Они позволяют создавать алгоритмы поведения NPC, учитывая их взаимодействия и зависимости друг от друга и от окружающей среды. Кроме того, они могут быть

использованы для создания умных NPC, которые адаптируются к поведению игрока.

1.6 Ньютоновская механика

Модель ньютоновской механики основывается на применении к агенту толпы двух сил, физической силы и психологической силы. Физическая сила заключается в применении физических законов для передвижения агента. Психологическая сила – это массовые или индивидуальные поведенческие особенности агента толпы. Примером массовой психологической силы является паника. Пример индивидуальной психологической силы – социальная дистанция.

Нагрузка на компьютер происходит за счет выполнения математических операций по физическим формулам. Психологическая сила, действующая на агента, взаимодействует с его физическими параметрами, что незначительно влияет на вычисление физических формул.

Применение модели ньютоновской механики для моделирования толпы в компьютерной игре имеет преимущество при моделировании динамических условий с ограниченным количеством агентов толпы, например, эвакуации, дорожного движения.

Минусы использованной модели ньютоновской механики:

1. Сложность управления. Ньютоновская механика требует от игрока понимания и управления физическими законами. Для достижения желаемой траектории и скорости движения корабля необходимо применять правильное ускорение и углы поворота. Это может потребовать определенной обучаемости и привыкания со стороны игрока.

2. Высокая сложность симуляции. Реализация реалистичной ньютоновской механики требует высокой вычислительной мощности и точных расчетов. В реальном времени необходимо учитывать множество факторов, таких как силы тяги, сопротивление воздуха и гравитацию. Это может вызвать сложности с оптимизацией и производительностью игры.

3. Высокий уровень сложности. Для некоторых игроков, особенно новичков, модель ньютоновской механики может быть сложной и трудноуправляемой. Игрокам может потребоваться время и практика, чтобы освоить управление кораблем и достичь точности и мастерства в пилотировании.

4. Отсутствие некоторых аспектов реальности. Хотя модель ньютоновской механики может быть достаточно реалистичной, она все равно является упрощенной версией физической реальности. Некоторые аспекты, такие как взаимодействие с другими объектами, может быть упрощено или исключено из-за ограничений игровой среды.

5. Ограниченность контекста. Ньютоновская механика, хотя и предоставляет реалистичную модель движения, имеет ограниченный контекст применения. Она наиболее подходит для игр, связанных с космическими или физическими средами, где действуют силы тяги и гравитации. В других типах игр, где ньютоновская механика не является релевантной, она может быть неэффективной или даже неприменимой.

Пример игры с использованием ньютоновской механики – «Space Velocity». Это космическая аркадная игра, в которой игроку предоставляется возможность пилотировать космический корабль и исследовать галактику. Игроку предстоит совершать маневры, управлять скоростью и маневренностью корабля, сражаться с врагами и выполнять различные миссии. На рисунке 5 изображен скриншот из игры «Space Velocity».



Рисунок 5 – Скриншот из игры «Space Velocity»

В игре «Space Velocity» используется модель ньютоновской механики для моделирования движения космического корабля. Модель учитывает законы Ньютона, включая силу тяги и гравитацию, чтобы определить движение корабля в космическом пространстве. Игрок может изменять ускорение и направление движения корабля, что влияет на его траекторию и скорость.

1.7 Гибридные модели

Гибридные методы комбинируют два или более подхода моделирования для более точного и эффективного моделирования поведения толпы. В моделировании толпы гибридные методы все чаще используются для моделирования масштабных толп в реальном времени, таких как в видеоиграх, тематических парках и симуляциях чрезвычайных ситуаций. Преимущество гибридных методов заключается в том, что они могут объединять преимущества различных подходов моделирования, что приводит к более реалистичному и точному поведению толпы.

Например, NPC могут использовать алгоритмы перемещения, основанные на модели броуновского движения, для создания естественных и нелинейных траекторий движения в толпе. Также могут быть использованы алгоритмы, основанные на клеточных автоматах или непрерывных моделях, для учета зависимостей между NPC и объектами окружающей среды.

Минусы использованной гибридной модели:

1. Сложность реализации. Гибридная модель требует совмещения и интеграции двух различных подходов – клеточных автоматов и мультиагентных систем. Это может быть сложным с точки зрения программирования и архитектуры игры. Разработчики должны иметь достаточные знания и опыт для успешной реализации и согласования двух моделей.

2. Вычислительная сложность. Использование гибридной модели может потребовать значительных вычислительных ресурсов, особенно при большом количестве клеток и агентов в игровом мире. Выполнение расчетов и

взаимодействий между клеточными автоматами и мультиагентными системами может стать вызовом для производительности игры.

3. Баланс и оптимизация. Сочетание двух различных моделей требует тщательного балансирования и оптимизации. Важно подобрать правильные параметры и правила для клеточных автоматов и мультиагентных систем, чтобы обеспечить согласованность и реалистичность игрового процесса. Это может потребовать многочисленных экспериментов и настроек со стороны разработчиков.

4. Сложность понимания для игроков. Гибридная модель может быть сложной для понимания и освоения игроками. Игрокам может потребоваться время и усилия, чтобы разобраться в механиках игры, основанных на двух различных моделях. Разработчики должны уделить особое внимание обучению и введению игроков в игровой мир.

5. Требуется сбалансированный дизайн. Чтобы гибридная модель была успешной, необходимо обеспечить сбалансированный дизайн игрового процесса. Взаимодействия между клеточными автоматами и мультиагентными системами должны быть интуитивными и логичными. Игрокам должно быть предоставлено достаточное количество контроля и возможностей для влияния на развитие города.

Один из примеров гибридного метода, используемого в моделировании толпы, – это сочетание клеточных автоматов и моделей на основе агентов. Клеточные автоматы могут моделировать общее движение толпы в среде, похожей на сетку, в то время как модели на основе агентов могут учитывать социальное и эмоциональное поведение отдельных членов толпы. Это позволяет более точно и реалистично моделировать поведение толпы в различных ситуациях.

Другой пример гибридного метода – это комбинация ньютоновской механики и моделей на основе агентов. Ньютоновская механика может использоваться для моделирования физических взаимодействий между объектами, в то время как модели на основе агентов могут учитывать социальное

поведение в толпе. Это позволяет создать более точную модель поведения толпы, учитывая, как физические, так и социальные аспекты.

Пример вышедшей игры с использованием гибридной модели – «City Builder: Fusion». Это стратегическая игра, в которой игроку предстоит создать и управлять развитием города. Игрок будет строить здания, управлять ресурсами, улучшать инфраструктуру и удовлетворять потребности горожан. Цель игры – развить процветающий и устойчивый город. На рисунке 6 изображен скриншот из игры «City Builder: Fusion».



Рисунок 6 – Скриншот из игры «City Builder: Fusion»

В игре «City Builder: Fusion» используется гибридная модель, объединяющая элементы клеточных автоматов и мультиагентных систем. Клеточные автоматы используются для моделирования роста и распространения городских зон, таких как жилые, коммерческие и промышленные районы. Мультиагентные системы применяются для моделирования поведения горожан, их взаимодействия, перемещения и принятия решений.

В целом, гибридные методы в моделировании толпы позволяют создавать более точные и реалистичные модели поведения толпы, что может быть полезно во многих областях, включая развлекательную и аварийную симуляцию.

1.8 Сравнение подходов моделирования реалистичного поведения толпы по критерию реалистичности

Броуновское движение представляет случайное движение частиц без определенного направления или цели. Оно не учитывает взаимодействия между частицами или окружающей средой, что ограничивает его реалистичность.

Непрерывные модели, основанные на физических законах и дифференциальных уравнениях, могут достичь высокой степени реалистичности. Они учитывают взаимодействия и физические свойства объектов, что позволяет им более точно отражать реальное поведение толпы.

Клеточные автоматы разбивают пространство на ячейки, каждая из которых имеет свое состояние. Они могут учитывать ограниченные взаимодействия между ячейками, но не всегда удается достичь высокой степени реалистичности в моделировании сложных поведений толпы.

Мультиагентные методы, основанные на моделировании поведения отдельных агентов в толпе, позволяют учесть их взаимодействия, социальные и психологические аспекты. Они способны достичь высокой степени реалистичности, особенно при учете различных факторов, таких как цели, среда и коммуникация между агентами.

Ньютоновская механика позволяет учесть физические законы, такие как законы движения и взаимодействия тел. Она может предоставить некоторую степень реалистичности в моделировании толпы, особенно в отношении физических свойств и коллизий между объектами. Учет массы, силы, ускорения и импульса может способствовать более реалистичному поведению агентов в толпе.

Гибридные модели комбинируют различные подходы и методы моделирования для достижения высокой степени реалистичности. Они могут

включать элементы броуновского движения, непрерывных моделей, клеточных автоматов, мультиагентных методов и ньютоновской механики, а также другие подходы. Использование комбинации различных методов позволяет достичь более точного и реалистичного моделирования поведения толпы.

В целом, моделирование реалистичного поведения толпы требует учета множества факторов, таких как взаимодействия, цели, физические свойства и социальные аспекты. Мультиагентные методы и гибридные модели обычно обеспечивают наиболее высокую степень реалистичности, так как они учитывают сложные взаимодействия и поведение отдельных агентов в толпе. Однако, непрерывные модели и ньютоновская механика также могут предоставить значительную реалистичность в моделировании физических свойств толпы. Броуновское движение и клеточные автоматы, хотя могут быть простыми в реализации, обычно ограничены в достижении высокой степени реалистичности. Выбор модели зависит от конкретных требований проекта и компромисса между реализмом и вычислительной сложностью.

1.9 Сравнение подходов моделирования реалистичного поведения толпы по критерию вычислительной сложности

Броуновское движение является относительно простым и вычислительно легким подходом. Оно не требует сложных вычислений или большого количества ресурсов, поскольку представляет случайное движение без учета окружающей среды или взаимодействия с другими объектами.

Непрерывные модели, основанные на физических законах и дифференциальных уравнениях, могут быть более вычислительно сложными. Они требуют решения уравнений движения, обработки взаимодействий и обновления состояния объектов с течением времени. В зависимости от точности моделирования и сложности системы, вычислительные затраты могут быть умеренными до высоких.

Клеточные автоматы, разбивающие пространство на ячейки, имеют умеренную вычислительную сложность. Обновление состояния ячеек и учет

взаимодействий между ними может потребовать некоторых вычислительных ресурсов. Однако, в зависимости от размера сетки и сложности правил, вычислительные затраты могут быть управляемыми.

Мультиагентные методы требуют моделирования поведения и взаимодействия между отдельными агентами. В зависимости от количества агентов и сложности модели, вычислительные затраты могут быть значительными. Обработка различных аспектов, таких как взаимодействия, коммуникация и принятие решений, может потребовать значительных ресурсов.

Ньютоновская механика основана на физических законах и требует решения дифференциальных уравнений движения объектов в системе. Это может быть вычислительно интенсивной задачей, особенно при большом количестве объектов в толпе и сложных взаимодействиях между ними. Вычислительные затраты могут значительно возрасти при учете коллизий, фрикционных сил и других физических аспектов.

Вычислительная сложность гибридных моделей зависит от их конкретной реализации и сочетания различных подходов. Если гибридная модель комбинирует вычислительно затратные методы, такие как ньютоновская механика и мультиагентные подходы, то вычислительная сложность может быть высокой. Однако, оптимальный выбор и сочетание методов может помочь снизить вычислительные затраты и достичь баланса между реалистичностью и вычислительной сложностью.

В целом, вычислительная сложность моделей реалистичного моделирования толпы может значительно варьироваться в зависимости от выбранного подхода. Броуновское движение и клеточные автоматы обычно имеют низкую вычислительную сложность, непрерывные модели и мультиагентные методы требуют более значительных вычислительных ресурсов, а ньютоновская механика и гибридные модели могут быть вычислительно интенсивными. При выборе модели необходимо учитывать баланс между реалистичностью и вычислительной сложностью, а также оценить

доступные вычислительные ресурсы для эффективного моделирования толпы в игре.

1.10 Алгоритмы передвижения агента

Одним из важных аспектов создания игрового мира является создание адекватных и реалистичных движений агента, которые не будут выбиваться из общей атмосферы игры.

Существует множество подходов и алгоритмов, которые могут быть использованы для реализации передвижения агента. Некоторые из них являются простыми, другие более сложными, но все они имеют свои преимущества и недостатки.

Один из самых простых и распространенных методов – это использование заранее заданных путевых точек. Агент движется от одной точки к другой, причем маршрут заранее известен и задан разработчиком. Этот метод может быть эффективным, если у агента есть четкий маршрут, который должен быть выполнен в игре. Однако, если агент должен перемещаться свободно, то этот метод не очень подходит, так как он не позволяет агенту реагировать на окружающую обстановку.

Другой метод – это использование алгоритмов поиска пути, таких как A^* [10] или Dijkstra [11], чтобы определить оптимальный маршрут агенту в реальном времени. Этот метод позволяет агенту реагировать на изменения в окружающей среде и выбирать оптимальный маршрут в зависимости от текущих обстоятельств. Однако, этот метод может быть довольно трудоемким и требовать больших вычислительных ресурсов.

Недавно, благодаря развитию машинного обучения и глубокого обучения, стали возможны новые подходы к реализации передвижения агента. Один из таких подходов – это использование нейронных сетей для обучения агента перемещению в реальном времени. Агент обучается на основе наблюдаемых данных, и тем самым может научиться перемещаться в общих условиях,

реагировать на изменения в окружающей обстановке и выполнять сложные действия.

Кроме того, существует ряд других подходов, таких как использование моделирования массы–пружины или методы, основанные на физике, которые могут быть использованы для реализации передвижения агента.

Один из подходов к реализации передвижения агента основан на алгоритмах навигации. Эти алгоритмы позволяют агенту обходить препятствия, следовать за игроком или другими агентами, и находить оптимальный путь до нужной точки на карте. Одним из самых популярных алгоритмов навигации является A^* .

A^* – это алгоритм поиска пути, который используется для поиска оптимального пути между двумя точками на карте, учитывая препятствия на пути. Он основан на поиске в ширину с учетом эвристики, которая помогает выбрать путь на основе прогнозируемой стоимости достижения конечной точки. A^* используется во многих играх, чтобы определить путь, по которому агент должен идти, чтобы достичь своей цели.

Другой подход к реализации передвижения агента – это использование физического движка. Физические движки имитируют физические законы и позволяют объектам в игре взаимодействовать друг с другом и с окружающей средой. Физические движки могут использоваться для определения коллизий между объектами, а также для расчета перемещения агента в ответ на силы, действующие на них.

В современных играх используются различные комбинации этих подходов для создания реалистичного передвижения агента. К примеру, в играх с открытым миром, как GTA V, применяются подход поиска пути, физический движок и нейронные сети для создания реалистичного поведения агентов, которые должны уметь уклоняться от машин и людей, двигаться в заданном направлении, останавливаться при перекрестках и т.д.

Еще одним подходом к реализации передвижения агента является использование траекторий движения. В этом случае для каждого агента заранее

задаются определенные траектории, по которым они будут перемещаться. Этот подход может использоваться для моделирования движения толпы в заданных областях, таких как улицы, торговые центры и т.д. Он может быть особенно эффективен, если разработчики хотят, чтобы агенты выполняли определенные действия на своем пути, например, общались с другими агентами или совершали определенные покупки в магазинах.

Наконец, существует подход, основанный на использовании генетических алгоритмов и машинного обучения. В этом случае агенты обучаются на основе данных о движении реальных людей и использовании генетических алгоритмов для определения наиболее эффективных стратегий передвижения. Этот подход может создавать наиболее реалистичное движение агента, но требует больших затрат на разработку и обучение алгоритмов.

Выводы

В процессе сравнения моделей реалистичного поведения толпы по критериям реалистичности и вычислительной сложности, можно сделать вывод:

По критерию реалистичности, модели, основанные на физических законах и непрерывных моделях, такие как ньютоновская механика, достигают более высокого уровня реалистичности. Они позволяют моделировать точные физические движения и взаимодействия в толпе, а также учитывать различные факторы, такие как коллизии и взаимодействия с окружающей средой.

Броуновское движение и клеточные автоматы предоставляют простые и эффективные подходы для моделирования толпы, но их уровень реалистичности ограничен. Они хорошо подходят для создания атмосферы множества агентов, но не могут точно моделировать сложные взаимодействия и поведение.

Мультиагентные методы предоставляют высокий уровень реалистичности, учитывая взаимодействия и коммуникацию между отдельными агентами в толпе. Они могут достичь точного моделирования сложного поведения, но требуют значительных вычислительных ресурсов и сложной реализации.

По критерию вычислительной сложности, броуновское движение и клеточные автоматы обладают низкой вычислительной сложностью. Они требуют меньше ресурсов для реализации и обработки, что делает их более доступными и эффективными в реальном времени. Однако, более точные и реалистичные модели, такие как непрерывные модели, мультиагентные методы и ньютоновская механика, могут быть вычислительно более сложными и требовать больших вычислительных ресурсов.

В результате анализа подходов и алгоритмов передвижения агента можно сделать следующие выводы:

Использование путевых точек является простым и эффективным способом управления передвижением агентом. Этот подход позволяет определить заданный маршрут и точки назначения, по которым агент будет перемещаться. Однако он ограничен в гибкости и адаптивности, поскольку требует предварительного определения всех путевых точек и маршрутов.

Алгоритмы поиска пути, такие как A^* или Dijkstra, позволяют агенту находить оптимальный путь от текущего положения к целевой точке, учитывая препятствия и пространство передвижения. Эти алгоритмы обеспечивают адаптивность и реактивность агента в изменяющейся среде, однако их вычислительная сложность может быть значительной при большом количестве агентов и сложных сценариях.

Использование нейронных сетей [12] для моделирования поведения агента позволяет создавать более интеллектуальные и адаптивные системы передвижения. Нейронные сети могут обучаться на основе данных и оптимизировать свои веса для достижения желаемых результатов. Это позволяет агенту адаптироваться к различным ситуациям и динамически изменять свое поведение. Однако требуется большой объем данных для обучения и вычислительные ресурсы для тренировки и выполнения нейронных сетей.

Использование физического движка позволяет моделировать реалистичную физику передвижения агента, учитывая коллизии, гравитацию и другие физические законы. Этот подход создает естественное и реалистичное

поведение агента, особенно в сценах с физическими взаимодействиями. Однако вычислительная сложность физического моделирования может быть значительной, особенно при большом количестве агентов.

Использование траекторий движения позволяет задать определенные пути и паттерны передвижения агента. Траектории могут быть предопределенными или динамическими, в зависимости от ситуации и контекста. Этот подход предоставляет точное управление над движением агента и позволяет создавать разнообразные и предсказуемые паттерны поведения. Однако он ограничен в гибкости и адаптивности, так как требует предварительного определения траекторий и их обновления при изменении условий среды.

Подход, который объединяет генетический алгоритм и машинное обучение для оптимизации и настройки параметров передвижения агента. Генетический алгоритм используется для эволюции и отбора наилучших решений, а машинное обучение помогает моделировать и адаптироваться к окружению. Этот подход позволяет создавать агента с адаптивным и улучшающимся поведением, но может потребовать значительных вычислительных ресурсов для обучения и оптимизации модели.

В целом, каждый из этих подходов имеет свои преимущества и недостатки, и разработчики игр могут выбирать подход, который лучше всего подходит для их конкретной игры. Независимо от выбранного подхода, моделирование реалистичного движения агента может значительно улучшить игровой опыт и создать более атмосферную игру для игроков.

2 Разработка гибридной модели поведения толпы в паническом состоянии для экшен–игры

2.1 Структура гибридной модели поведения толпы в ситуации паники

Предлагаемая в данном диссертационном исследовании гибридная модель, объединяет несколько компонентов для достижения реалистичного моделирования толпы агентов. Эта модель включает в себя мультиагентную модель, ньютоновскую механику, алгоритм A^* и клеточные автоматы, что позволяет обеспечить более реалистичного и интерактивное поведение агентов в игровом мире.

Использование мультиагентного подхода предполагает рассмотрение неигровых персонажей экшен–игры как программных агентов, обладающих индивидуальным поведением, целями и характеристиками, что позволяет им взаимодействовать друг с другом и окружением.

Мультиагентная модель определяет правила взаимодействия между агентами, такие как:

1. Реакции на окружающие события. Каждый агент обладает способностью воспринимать окружающую среду и реагировать на события. Это может включать обнаружение опасных мест, противников или других агентов.
2. Команды и коммуникация. Агенты могут взаимодействовать друг с другом посредством близкой дистанции друг к другу. В ситуации, когда несколько агентов попали в опасное место, они могут передавать друг другу данные о своем положении и об опасности. Команды могут быть использованы для координирования действий и принятия общей стратегии безопасного отступления.
3. Совместные действия и кооперация. В ситуации, когда несколько агентов находятся в опасном месте, они могут принять решение о совместном действии для сбора вместе и безопасного отступления. Это может включать

формирование группы или зоны безопасности, где агенты собираются и движутся вместе.

Для достижения реалистичного передвижения и взаимодействия агентов с окружением используется модель, основанная на законах Ньютона. Ньютоновская механика определяет физические свойства агентов, такие как гравитация, трение, столкновения и воздействие сил. Агенты движутся и взаимодействуют с окружением, исходя из этих физических законов.

Для нахождения оптимального маршрута и обхода препятствий в игровом мире используется алгоритм A^* . Алгоритм A^* основывается на оценке стоимости пути от начальной точки до целевой точки, учитывая географические особенности и препятствия.

Клеточные автоматы используются для моделирования окружающего мира и его особенностей. В рамках гибридной модели клеточные автоматы определяют характеристики, свойства окружения, такие как типы территории, препятствия и различные зоны влияния и облегчают расчет пути для алгоритма A^* .

Такая структура гибридной модели позволяет создавать более реалистичную и интерактивную игровую среду, где агенты в состоянии паники взаимодействуют друг с другом и с окружением, соблюдая физические законы, находя оптимальные маршруты и учитывая особенности окружающего мира, созданные с помощью клеточных автоматов. Это обеспечивает более реалистичный и захватывающий игровой опыт для игроков.

2.2 Математическая модель передвижения агентов толпы в состоянии паники

Математическая модель, основанная на ньютоновской механике, позволяет описать реалистичное передвижение, поворот агента и столкновения в состоянии паники. Для этого используются следующие формулы:

Движение агента описывается рядом формул. Скорость (V) персонажа можно выразить как:

$$V = V_0 + a * t,$$

где V_0 – начальная скорость персонажа;

a – ускорение персонажа;

t – время.

Ускорение (a) персонажа быть выражено следующей формулой:

$$a = \frac{F_{\text{тяжести}} + F_{\text{столкновения}}}{m},$$

где m – масса персонажа;

$F_{\text{тяжести}}$ – сила тяжести;

$F_{\text{столкновения}}$ – сила столкновения.

Положение (x) персонажа в зависимости от времени можно выразить как:

$$x = x_0 + V_0 * t + \frac{1}{2} * a * t^2,$$

где x_0 – начальное положение персонажа;

t – время.

Поворот агента описывается угловой скоростью и углом поворота.

Угловую скорость (ω) персонажа можно выразить как:

$$\omega = \omega_0 + a * t,$$

где ω_0 – начальная угловая скорость персонажа;

a – угловое ускорение персонажа,

t – время.

Угол поворота (θ) персонажа в зависимости от времени можно выразить как:

$$\theta = \theta_0 + \omega_0 * T + \frac{1}{2} * a * t^2,$$

где θ_0 – начальный угол поворота персонажа.

Для расчета коэффициента силы удара при столкновении используются ряд формулы.

Расчет коэффициента силы удара (C) двух агентов:

$$C = \frac{-(1 + e) * (Vr_1 - Vr_2)}{\frac{1}{m_1} + \frac{1}{m_2}}$$

где m_1 и m_2 – массы объектов 1 и 2 соответственно;

V_{r1} и V_{r2} – относительные скорости объектов 1 и 2 вдоль линии столкновения;

e – коэффициент восстановления, который определяет степень отскока или пружинистости при столкновении, обычно принимает значения от 0 (полностью неупругое столкновение) до 1 (полностью упругое столкновение).

При столкновении агента массой (m_1), скоростью (V_1) со статическим объектом, который не движется, сила столкновения ($F_{\text{столкновения}}$) может быть вычислена с учетом коэффициента силы удара (C):

$$F_{\text{столкновения}} = \frac{C * m_1 * V_1}{\Delta t},$$

где Δt – время столкновения.

При столкновении двух агентов с массами (m_1, m_2), скоростями (V_1, V_2), сила столкновения ($F_{\text{столкновения}}$) может быть вычислена с использованием коэффициента силы удара (C):

$$F_{\text{столкновения}} = \frac{C * (m_1 * V_1 - m_2 * V_2)}{\Delta t}.$$

Использование вышеприведенных формул позволяет описать передвижение персонажей с использованием ньютоновских законов.

2.3 Математическая модель социального кооперирования агентов толпы в состоянии паники

Математическая модель социального кооперирования персонажей внутри толпы в ситуации паники описывает силы притяжения и отталкивания между агентами.

Опишем силу притяжения между агентами в ситуации паники. Предположим, что каждый агент влияет на других агентов внутри определенного радиуса действия. Формула для расчета силы притяжения между двумя агентами может быть определена следующим образом:

$$F_{\text{притяжения}} = K_{\text{притяжения}} * (X_1 - X_2),$$

где $F_{\text{притяжения}}$ – сила притяжения между агентами;

$K_{\text{притяжения}}$ – коэффициент притяжения;

X_1 и X_2 – позиции агентов.

Для предотвращения столкновений между агентами можно использовать силу отталкивания. Формула для расчета силы отталкивания между двумя агентами может быть определена следующим образом:

$$F_{\text{отталкивания}} = \frac{K_{\text{отталкивания}} * (X_1 - X_2)}{d^2},$$

где $F_{\text{отталкивания}}$ – сила отталкивания между агентами;

$K_{\text{отталкивания}}$ – коэффициент отталкивания;

X_1 и X_2 – позиции агентов;

d – расстояние между агентами.

Расчет общей силы на каждого агента:

$$F_{\text{общ}} = F_{\text{притяжения}} + F_{\text{отталкивания}},$$

где $F_{\text{общ}}$ – общая сила на агента.

Учитывая расчет формул мультиагентной модели совместно с формулой расчета ускорения ньютоновской механики, итоговая формула расчета ускорения агента:

$$a = \frac{F_{\text{общ}} + F_{\text{тяжести}} + F_{\text{столкновения}}}{m},$$

где a – ускорение агента;

$F_{\text{общ}}$ – общая сила на агента;

$F_{\text{тяжести}}$ – сила тяжести;

$F_{\text{столкновения}}$ – сила столкновения;

m – масса агента.

Таким образом, путем комбинирования этих формул и применения их к каждому агенту в системе, можно создать математическую модель социального кооперирования между агентами толпы в состоянии паники путем отдаления и приближения к другим агентам.

2.4 Математический алгоритм поиска кратчайшего безопасного пути в ситуации паники

Перед процессом построения математического алгоритма поиска кратчайшего безопасного пути с использованием алгоритма A^* необходимо выполнить следующие шаги:

1. Определение пространства поиска. Сначала необходимо определить пространство, в котором будет осуществляться поиск пути. Это может быть двумерная сетка или трехмерное пространство в зависимости от конкретной задачи.

2. Определение начальной и конечной точек. Задается начальная точка, откуда начинается поиск пути, и конечная точка, к которой нужно найти оптимальный маршрут.

3. Создание графа или сетки. В зависимости от пространства поиска, создается граф или сетка, представляющая возможные позиции или узлы, между которыми можно перемещаться.

4. Определение стоимости переходов. Каждому переходу между узлами графа или сетки назначается стоимость, которая может быть определена на основе расстояния, времени, энергозатрат и других факторов.

5. Определение эвристической функции. Эвристическая функция используется для оценки оставшейся стоимости до конечной точки. Часто используется эвристическая функция расстояния, такая как эвклидово расстояние или расстояние Манхэттена.

6. Реализация алгоритма A^* . Сам алгоритм A^* состоит из итеративного процесса поиска пути. На каждой итерации алгоритма происходит выбор узла с наименьшей стоимостью из множества открытых узлов. Затем рассчитывается стоимость и эвристическая оценка для соседних узлов, и их добавляют в открытые или закрытые узлы в зависимости от условий.

7. Остановка алгоритма. Алгоритм продолжает работу до тех пор, пока не будет найден путь до конечной точки или все доступные узлы будут

исследованы. Если путь до конечной точки найден, можно восстановить последовательность узлов, образующих оптимальный путь.

8. Проверка и оптимизация. После реализации алгоритма A^* необходимо провести проверку и тестирование модели. Можно использовать различные сценарии и ситуации для проверки корректности и эффективности поиска пути. При необходимости можно внести оптимизации, чтобы улучшить производительность и скорость работы модели.

9. Интеграция с другими компонентами. Математическая модель поиска пути A^* может быть интегрирована в более крупную систему или приложение. Например, в игровой индустрии она может быть связана с системой искусственного интеллекта, физическим движком и другими компонентами игрового движка.

10. Поддержка и обновление. После внедрения модели поиска пути необходимо предусмотреть ее поддержку и возможные обновления. В процессе эксплуатации системы могут возникать новые требования или проблемы, которые требуют доработки и улучшения модели.

После выполнения шагов строится следующий математический алгоритм поиска кратчайшего безопасного пути и состоит из следующих шагов:

1. Инициализация сетки клеток, где каждая клетка представляет собой определенное пространственное положение на игровом поле. Определение препятствий на сетке клеток.

2. Определение эвристической функции, которая будет использоваться для оценки стоимости пути от текущей клетки до целевой клетки.

В данном случае, учитывая начальную дистанцию от препятствия, можно модифицировать эвристическую функцию следующим образом:

$$H_{(n)} = h_{(n)} + d,$$

где $h_{(n)}$ – эвристическая оценка от клетки n до целевой клетки (например, эвклидово расстояние);

d – начальная дистанция от препятствия.

3. Определение стоимости переходов между соседними клетками. Если клетка является препятствием, ей можно присвоить высокую стоимость, чтобы обойти ее. Стоимость перехода на свободную клетку равна 1. Стоимость перехода на занятую клетку равна 9.

4. Расчет стоимости пути от начальной клетки до текущей клетки, учитывая стоимость переходов и стоимость пути от начальной клетки до предыдущей клетки. Обновление общей оценки клетки, которая является суммой стоимости пути и эвристической функции:

$$F_{(n)} = g_{(n)} + H_{(n)},$$

где $g_{(n)}$ – стоимость пути от начальной клетки до текущей клетки.

5. Выбор оптимальной клетки. В каждой итерации алгоритма выбирается клетка с наименьшей общей оценкой $f(n)$ из списка доступных клеток для дальнейшей обработки.

6. Расчет стоимости перехода и обновление параметров клеток. Во время расчета стоимости перехода от текущей клетки к соседним клеткам, учитывается начальная дистанция от препятствия. Если новая стоимость перехода меньше текущей стоимости перехода к соседней клетке, обновляется параметр этой клетки.

7. Повторение шагов 5 и 6. Продолжается выбор и обработка клетки, пока не достигнет целевой клетки или не пройдет по всем доступным клеткам.

8. Восстановление пути. После достижения целевой клетки восстанавливается оптимальный путь, двигаясь от целевой клетки к начальной по родительским ссылкам.

Таким образом, вышеуказанные шаги позволяют построить математическую модель поиска пути алгоритмом A^* и учетом начальной дистанции от препятствия.

Выводы

Разработка математической модели реалистичного передвижения агентов толпы в состоянии паники с использованием ньютоновской механики и

мультиагентных методов, а также математической модели поиска пути алгоритмом A^* предоставляет значительные преимущества для моделирования и симуляции поведения агентов компьютерной игры в ситуации паники.

Использование ньютоновской механики позволяет достичь реалистичности в передвижении агентов, учитывая физические законы движения, силы притяжения и отталкивания, а также столкновения с другими объектами. Это позволяет создать более правдоподобную и естественную анимацию и взаимодействие между агентами и окружающей средой в ситуации паники.

Мультиагентные методы обеспечивают возможность моделирования сложного взаимодействия между множеством агентов. Они позволяют агентам обмениваться информацией, принимать решения на основе своего внутреннего состояния и состояния окружающих агентов. Такой подход позволяет моделировать различные ситуации, такие как совместные действия, согласованное передвижение и эвакуацию из опасных зон.

Использование алгоритма поиска пути алгоритмом A^* позволяет агентам эффективно планировать маршрут в сложной среде с препятствиями. Алгоритм A^* обеспечивает быстрое нахождение пути, учитывая стоимость прохождения каждой клетки и эвристическую оценку расстояния до цели.

3 Реализация гибридной модели толпы в компьютерной игре

3.1 Реализация передвижения агента в состоянии паники

Для описания алгоритма передвижения агента в состоянии паники была построена диаграмма активности. Диаграмма активности представлена на рисунке 7.

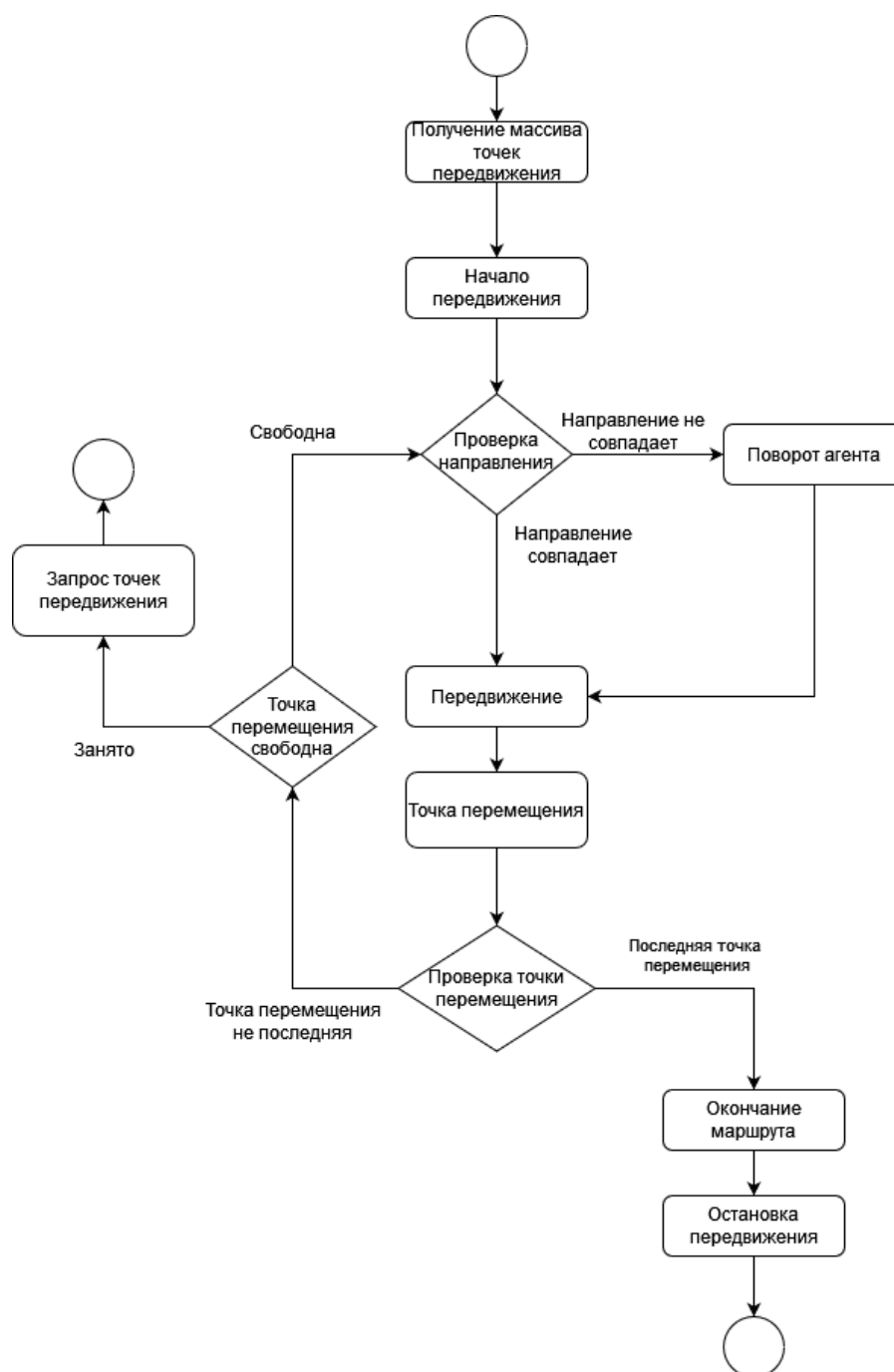


Рисунок 7 – Диаграмма активности передвижения агентов в состоянии паники

На диаграмме активности представлены действия от получения точек пути от алгоритма поиска пути, до плавного начала и остановки движения с помощью модели ньютоновской механики.

На рисунке 8 представлена диаграмма классов передвижения агентов в состоянии паники.

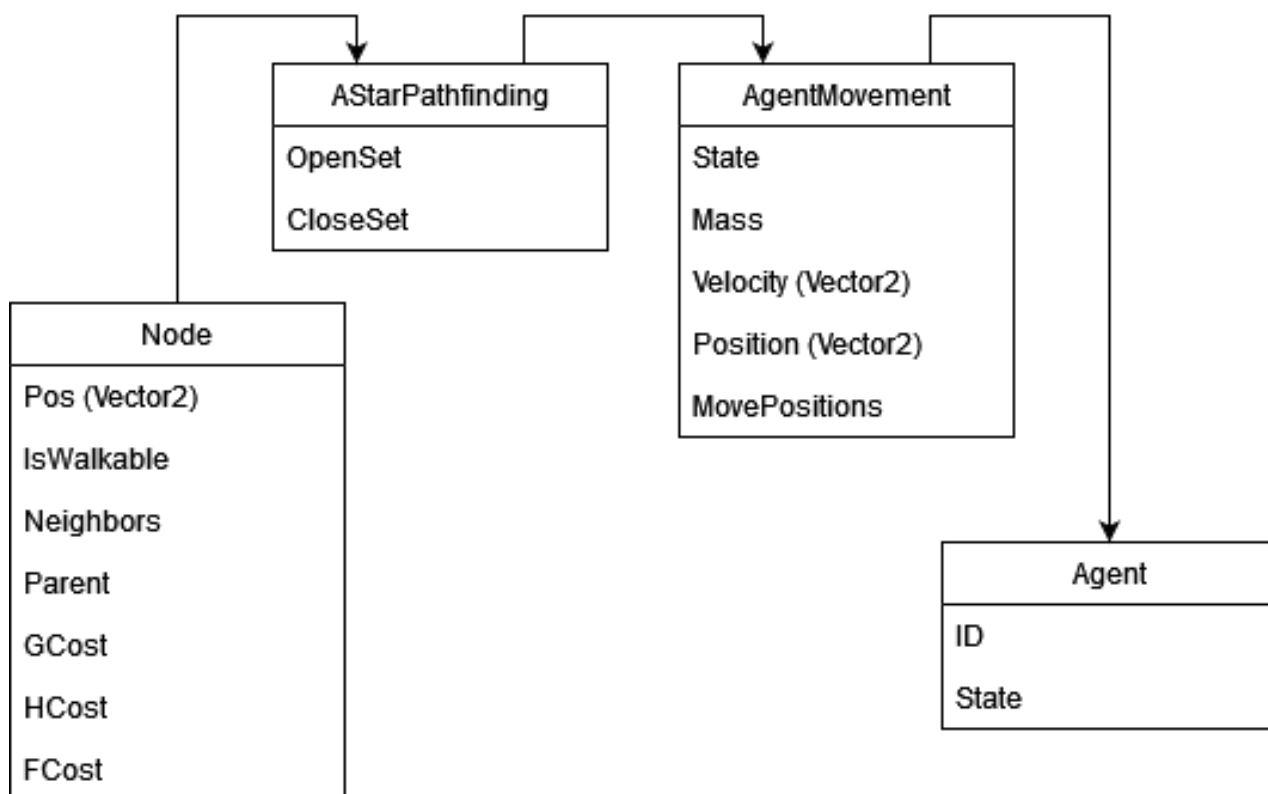


Рисунок 8 – Диаграмма классов передвижения агентов в состоянии паники

На диаграмме классов передвижения агентов в состоянии паники представлены классы: алгоритм поиска пути «AStarPathfinding» содержит массив «OpenSet» и массив «CloseSet» в которых хранятся открытые и закрытые точки перемещения. Агент «Agent» содержит поле «ID» в котором хранится номер агента и поле «State» в котором хранится состояние агента. Точка на карте «Node» содержит поле «Pos» хранит позицию точки на карте, поле «IsWalkable» состояние точки, «Neighbors» хранит соседние точки на карте, «Parent» хранит класс родителя, Поля «GCost», «HCost», «FCost» хранят значения стоимости пути по точке. Класс физических расчетов для передвижения «AgentMovement» содержит поле «Mass» в котором хранится вес агента, поле «Velocity» хранит

ускорение агента, поле «Position» текущая позиция агента, поле «MovePosition» хранит открытые точки перемещения агента.

3.2 Реализация взаимодействия агентов толпы в состоянии паники

Описания взаимодействия агентов толпы в состоянии паники продемонстрирована на диаграмме активности рисунок 9.

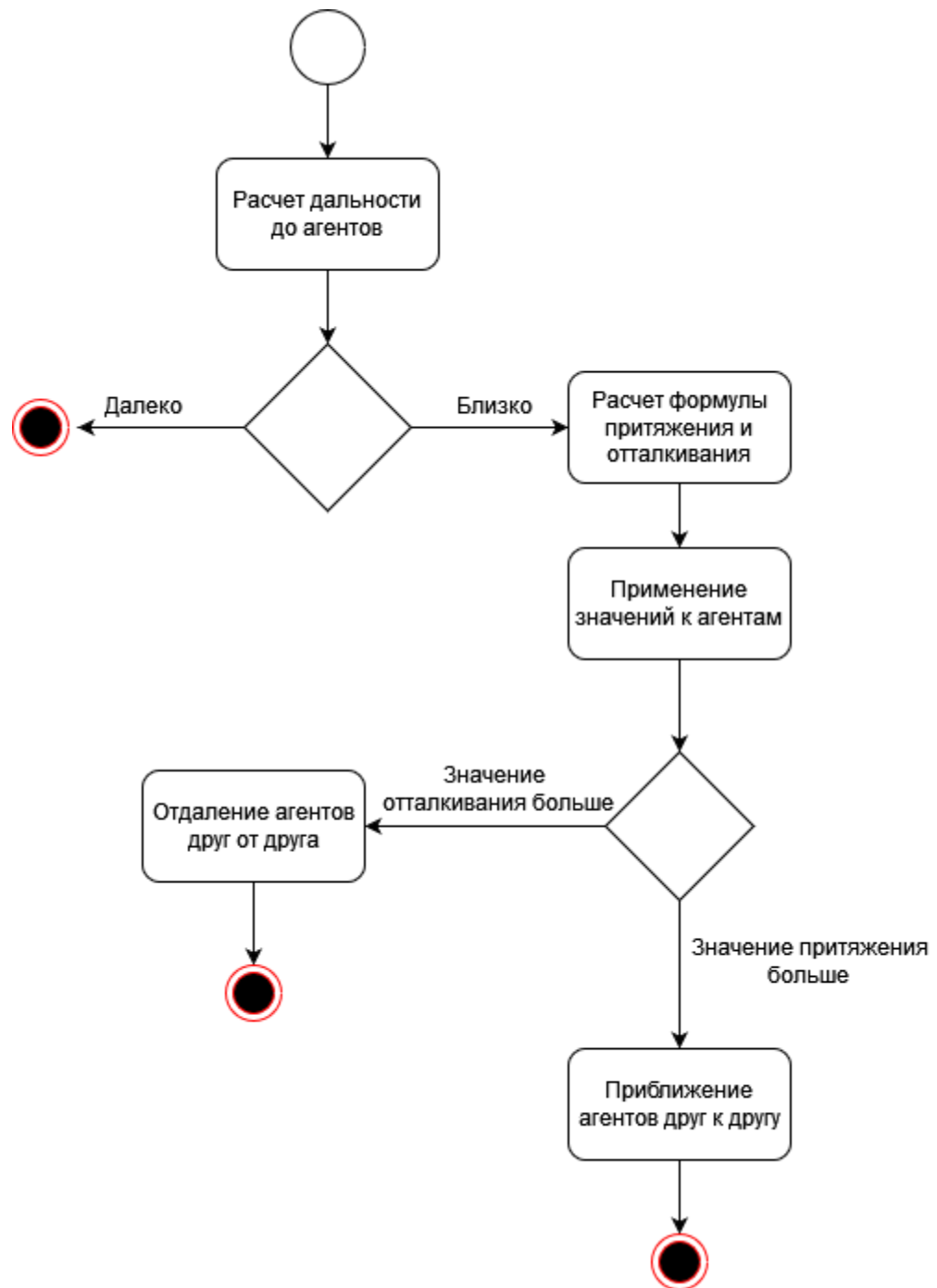


Рисунок 9 – Диаграмма активности описание взаимодействия агентов толпы в состоянии паники

На диаграмме активности представлены действия с расчетом и проверкой дальности агентов друг к другу. Если два агента слишком далеко друг от друга происходит выход из модели. Если агенты находятся в радиусе действия модели идет расчет по формуле для приближения друг к другу или отдаления друг от друга.

На рисунке 10 изображена диаграмма классов описание взаимодействия агентов толпы.

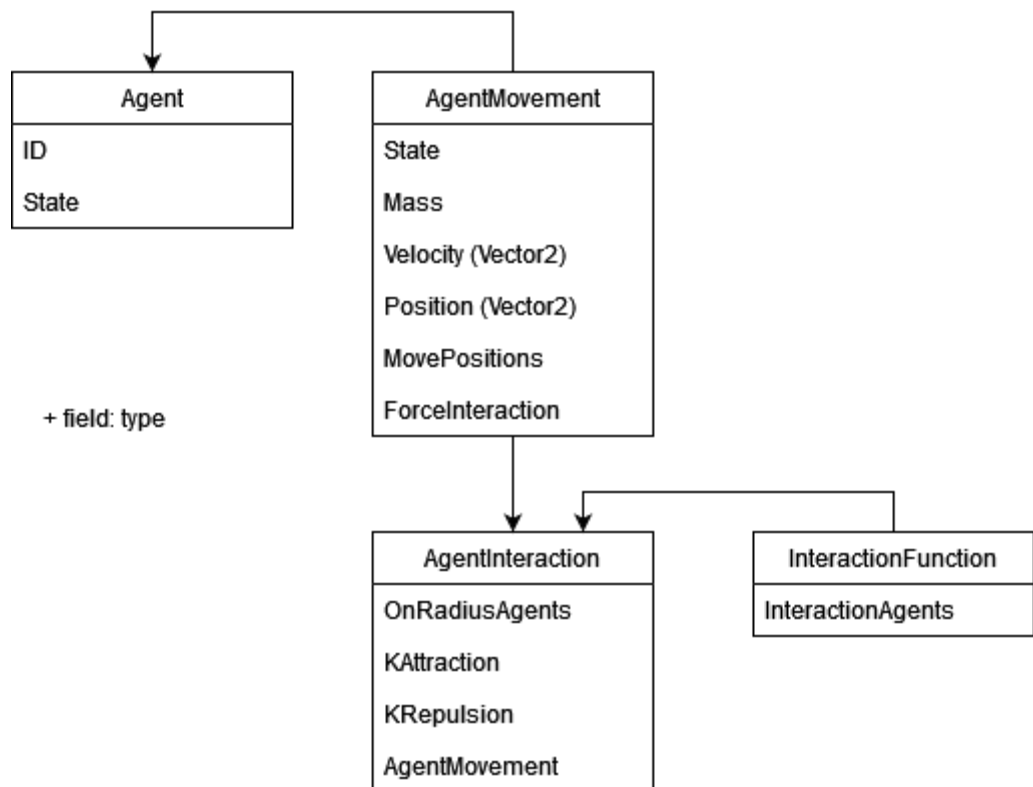


Рисунок 10 – Диаграмма классов описание взаимодействия агентов толпы в состоянии паники

На диаграмме классов реализации взаимодействия агентов толпы в состоянии паники представлены классы: агент «Agent», класс физических расчетов для передвижения «AgentMovement», класс агента для взаимодействий «AgentInteraction» содержит поле «OnRadiusAgents» хранит классы агентов в ближайшем радиусе, поля «KAttraction» и «KRepulsion» хранят индивидуальные коэффициенты отталкивания и притяжения, класс расчетов взаимодействий агентов «InteractionFunction» содержит поле «InteractionAgents» в котором хранятся классы агентов для взаимодействия.

3.3 Реализация поиска пути в состоянии паники

Для описания поиска пути в состоянии паники построена диаграмма активности, представленная на рисунке 11. На диаграмме отображен алгоритм перебора точек на карте с проверкой свободна ли точка, является ли она конечной и проверкой точки на нахождение ее в массивах. В конечном результате получается массив открытых точек на карте, по которым агент перемещается.

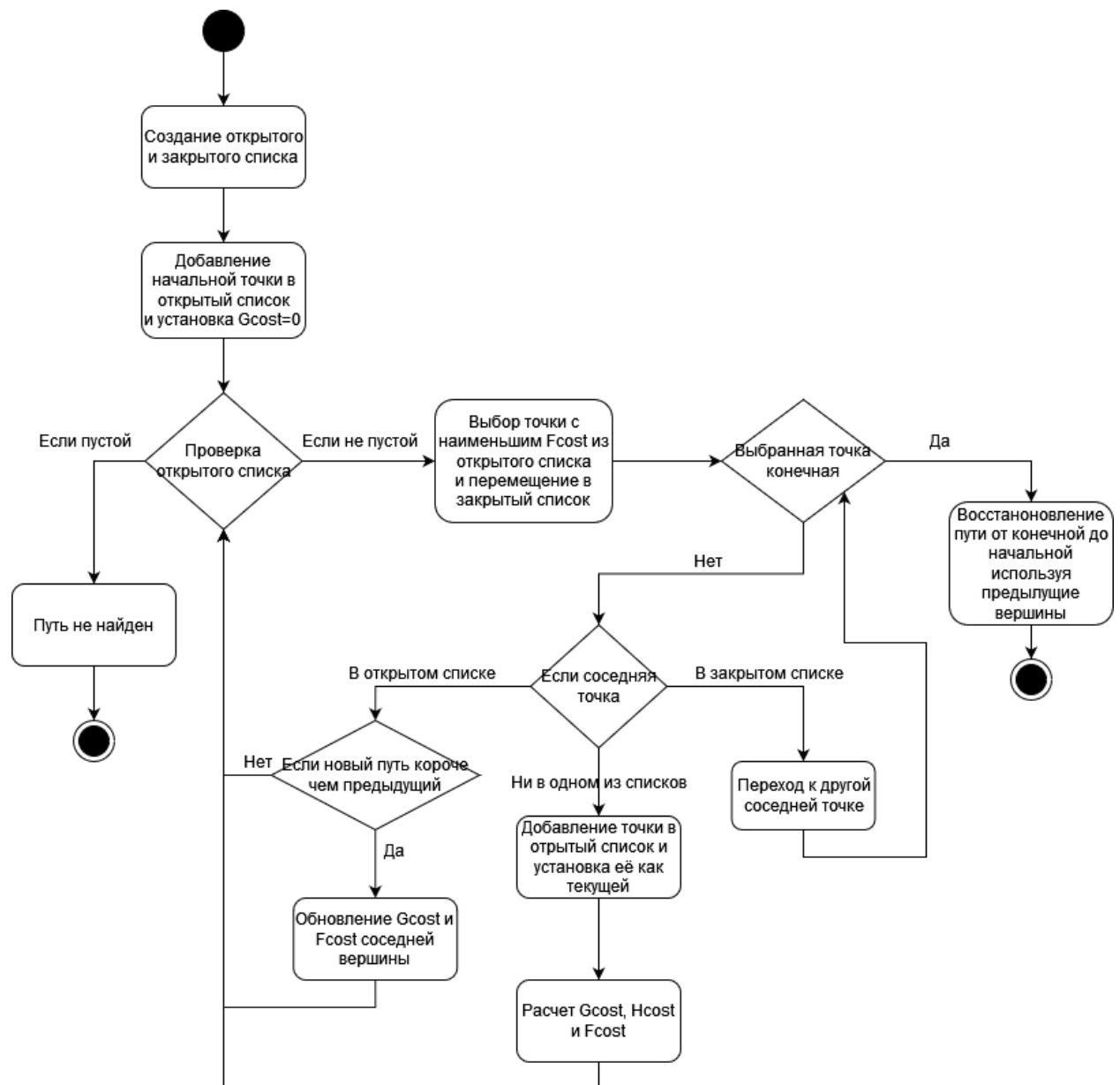


Рисунок 11 – Диаграмма активности описание поиска пути в состоянии паники

На диаграмме активности описан алгоритм поиска пути в состоянии паники.

Для описания поиска пути в состоянии паники построена диаграмма классов, представленная на рисунке 12.

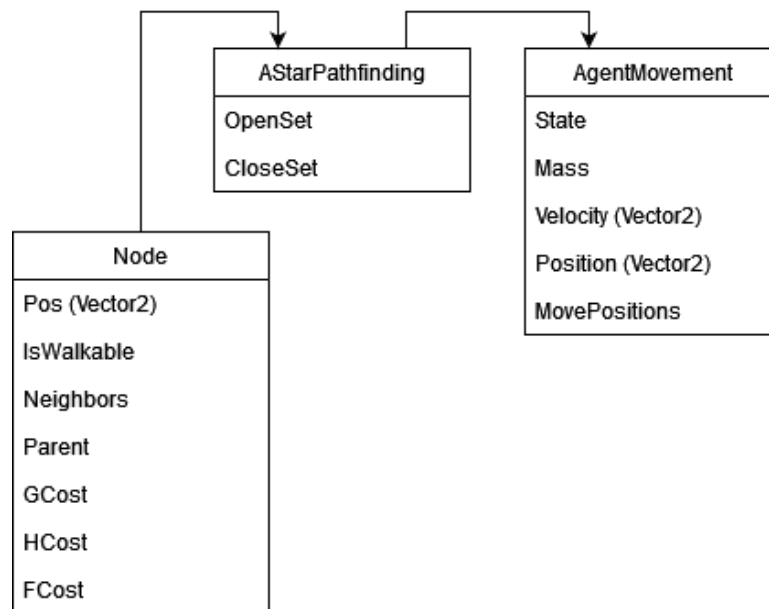


Рисунок 12 – Диаграмма классов описания поиска пути в состоянии паники

На диаграмме классов описания поиска пути в состоянии паники отображены классы необходимые для поиска пути такие как: класс точки на карте, класс физических расчетов для передвижения, класс алгоритма поиска пути.

3.4 Тестирование гибридной модели

В рамках данного исследования была разработана гибридная модель передвижения агентов в игровом движке Unity [13]. Для тестирования производительности и эффективности модели была создана специальная карта, на которой агенту необходимо пройти маршрут от одной стороны до другой, преодолевая препятствия.

Основная цель тестирования заключается в проверке ресурсопотребления модели, ее масштабируемости и времени вычисления в зависимости от количества агентов. Тестирование начинается с 4 агентов, а затем на каждой

попытке их количество увеличивается на 2, достигая в конечном итоге 512 агентов. В ходе тестирования собираются данные о загрузке процессора, используемой памяти, а также о времени, требуемом для вычисления пути и перемещения агентов. Эти данные позволят оценить производительность модели и ее способность эффективно обрабатывать большое количество агентов. На рисунке 13 изображена 4 агента тестирования.

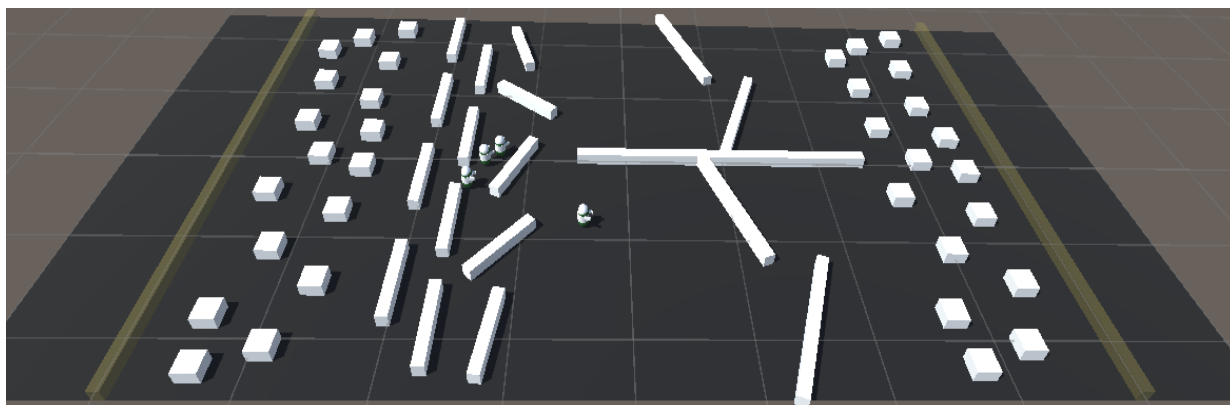


Рисунок 13 – Первое тестирование 4 агента на карте

На рисунке 14 изображены графики нагрузки процессора и памяти.

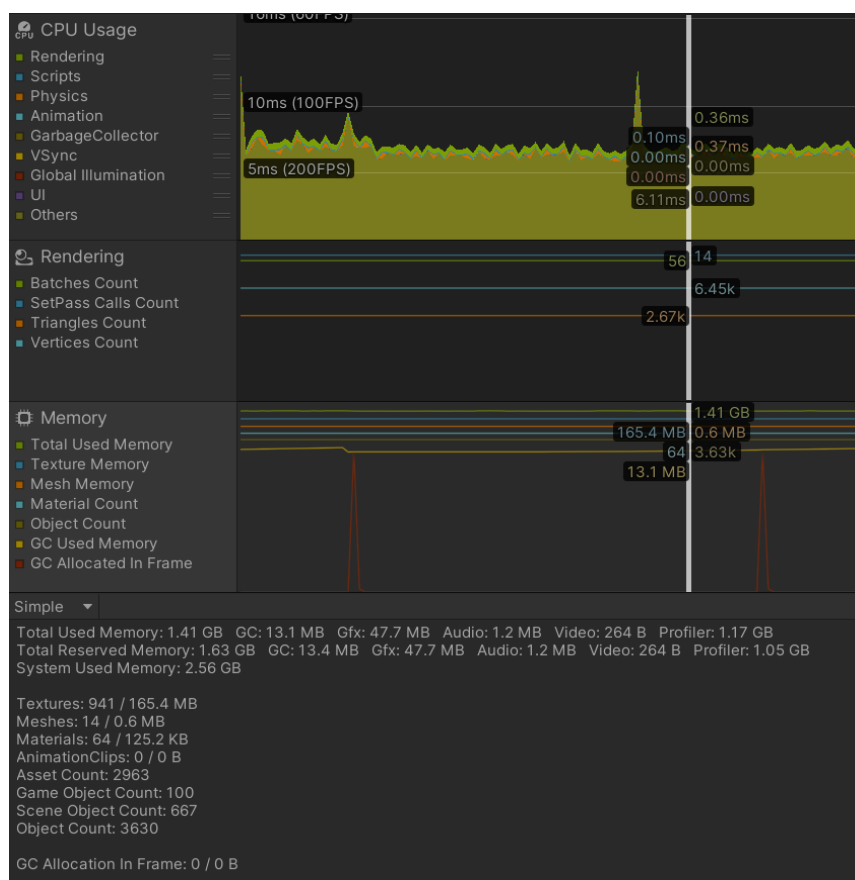


Рисунок 14 – Графики результатов первого тестирования 4 агентов

Результаты первого тестирования 4 агентов: выполнение кода гибридной модели за 0.37 миллисекунды, показатель частоты обновления кадров больше 100, общее заполнение памяти 1.41 гигабайт. Так как система занимает одинаковое количество памяти во всех тестированиях из чего следует что на изменения общей заполняемой памяти влияет тестируемая гибридная модель.

На рисунке 15 изображено второе тестирование гибридной модели.

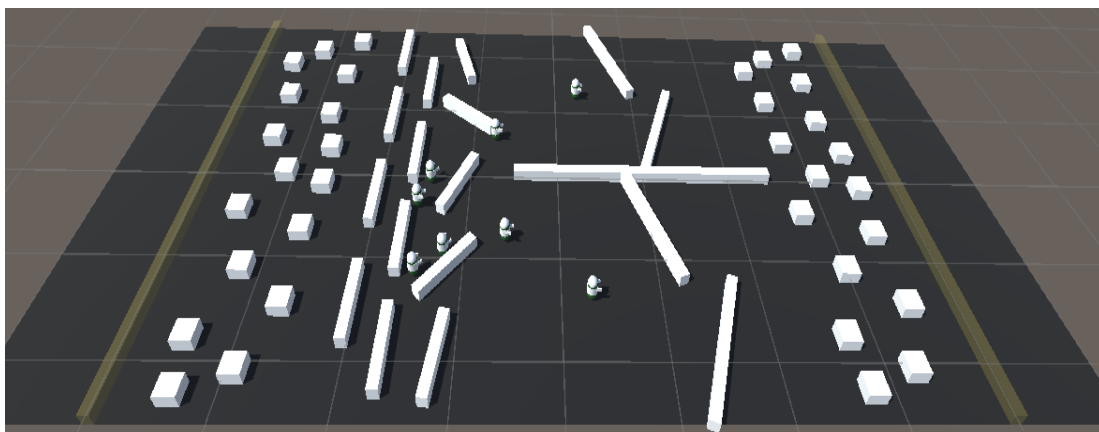


Рисунок 15 – Изображение второго тестирования с 8 агентами на карте

На рисунке 16 изображен график результатов второго тестирования 8 агентов.

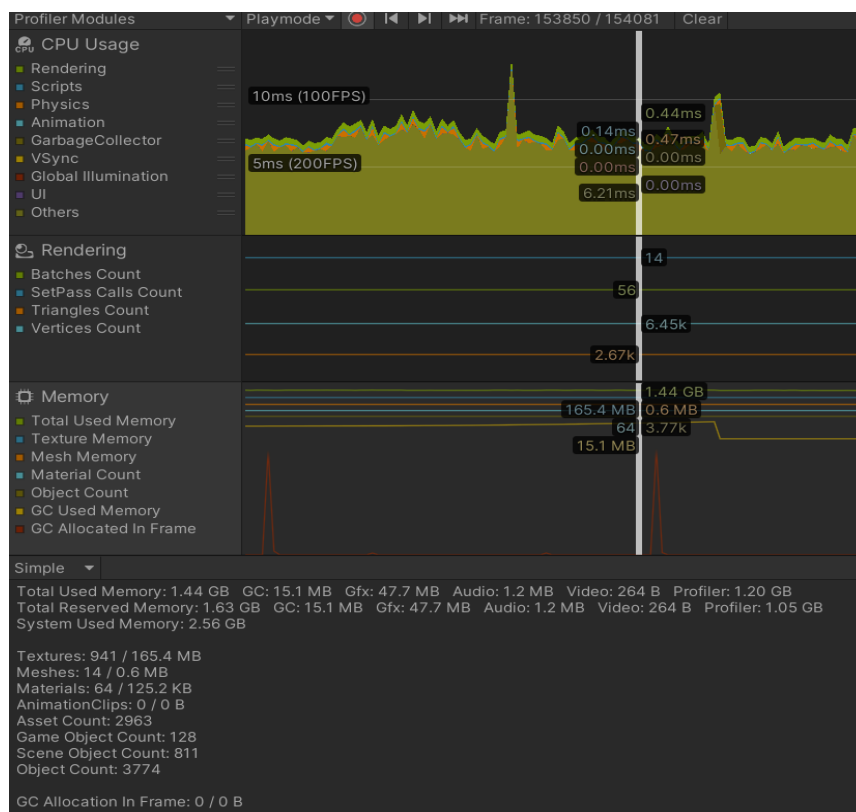


Рисунок 16 – График результатов второго тестирования 8 агентов.

Результаты второго тестирования 8 агентов: выполнение кода гибридной модели за 0.47 миллисекунды, показатель частоты обновления кадров больше 100, общее заполнение памяти 1.44 гигабайт.

На рисунке 17 изображено третье тестирование гибридной модели.

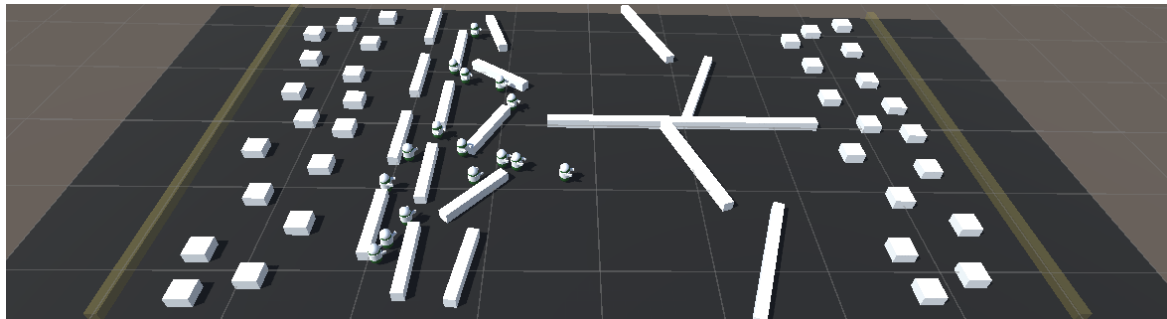


Рисунок 17. Третье тестирование гибридной модели 16 агентов.

На рисунке 18 изображен график результатов третьего тестирования гибридной модели 16 агентов.

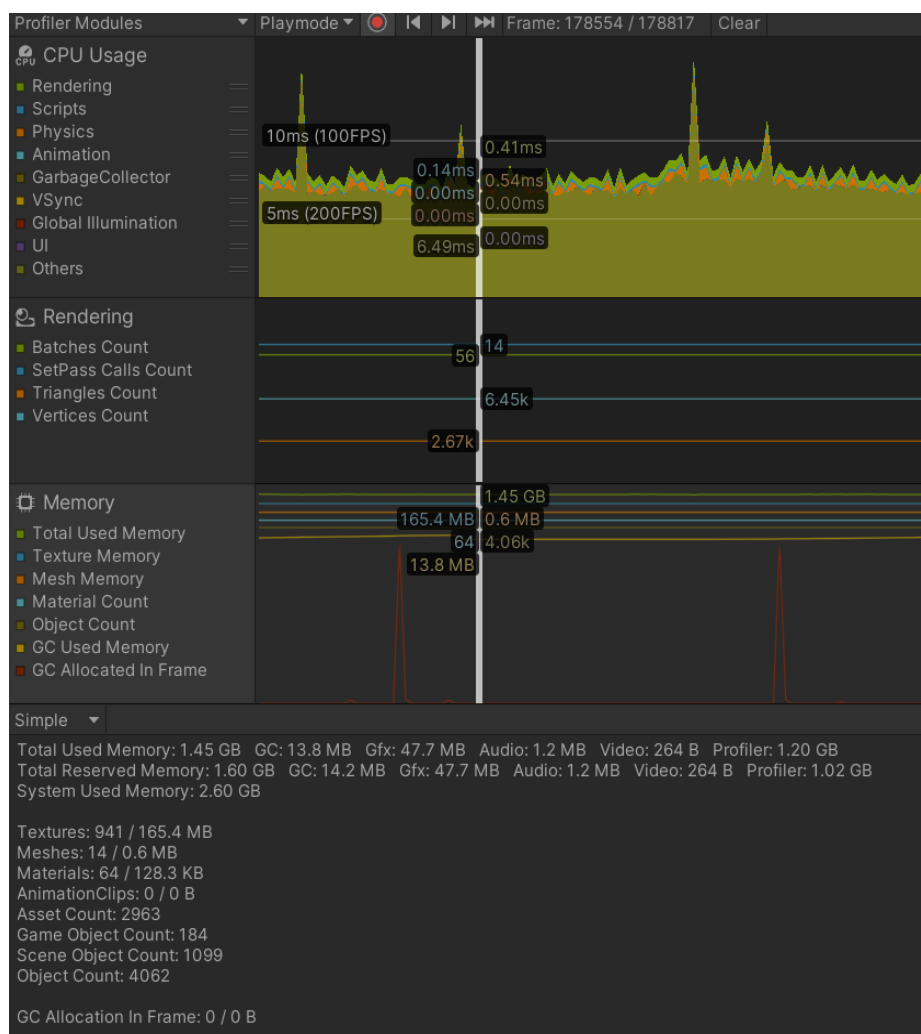


Рисунок 18 – График результатов третьего тестирования 16 агентов

Результаты третьего тестирования 16 агентов: выполнение кода гибридной модели за 0.54 миллисекунды, средний показатель частоты обновления кадров больше 100, общее заполнение памяти 1.45 гигабайт.

На рисунке 19 изображено четвертое тестирование гибридной модели 32 агентов.

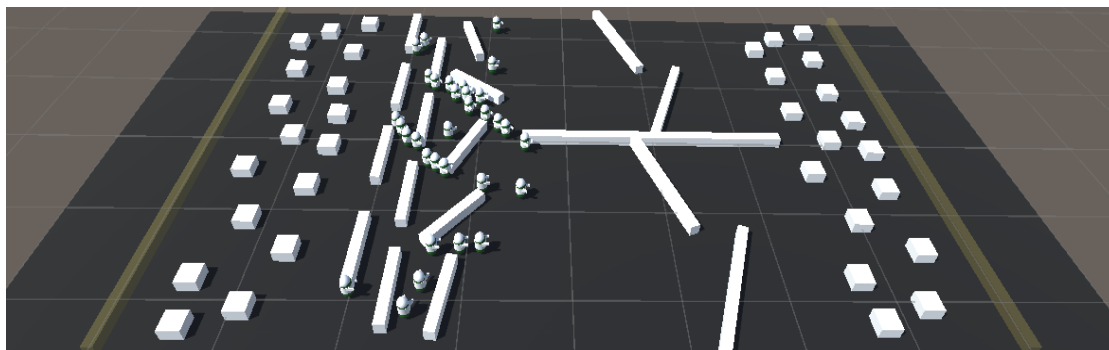


Рисунок 19 – Четвертое тестирование гибридной модели 32 агентов

На рисунке 20 изображен график результатов тестирования гибридной модели 32 агентов.

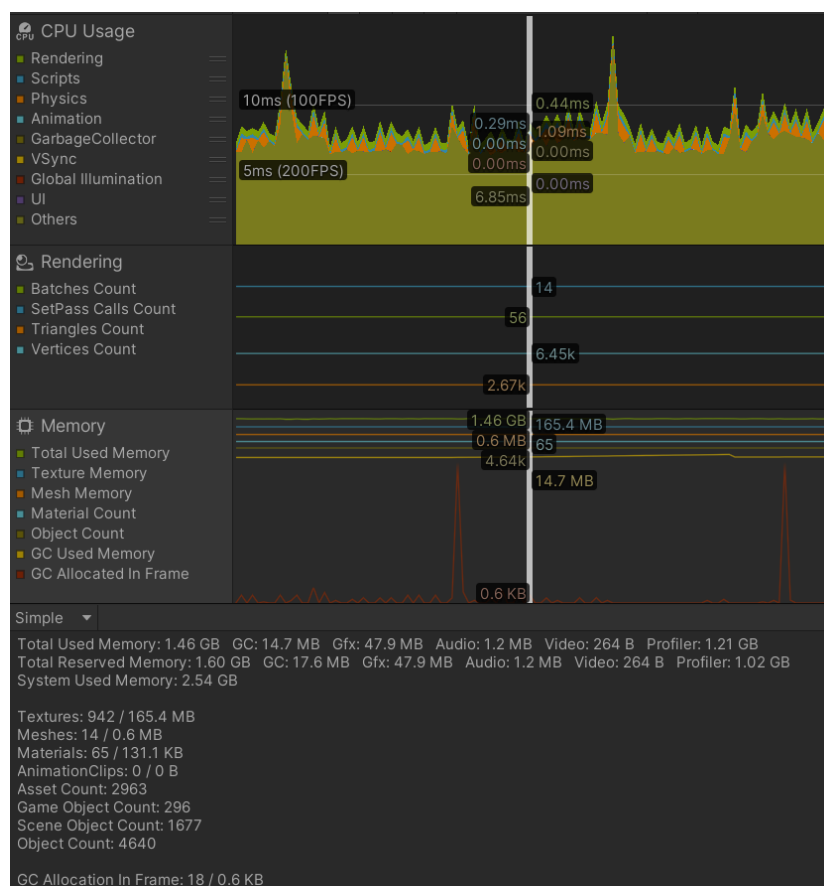


Рисунок 20 – График результатов четвертого тестирования гибридной модели 32 агентов

Результаты четвертого тестирования 32 агентов: выполнение кода гибридной модели за 1.09 миллисекунды, средний показатель частоты обновления кадров больше 100, общее заполнение памяти 1.46 гигабайт.

На рисунке 21 изображено пятое тестирование гибридной модели.

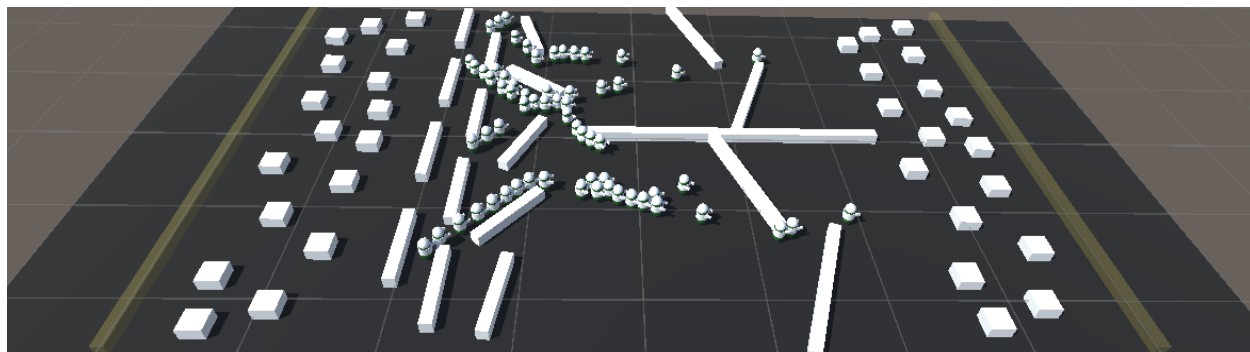


Рисунок 21 – Пятое тестирование гибридной модели 64 агента

На рисунке 22 изображен график результатов тестирования гибридной модели 64 агента.

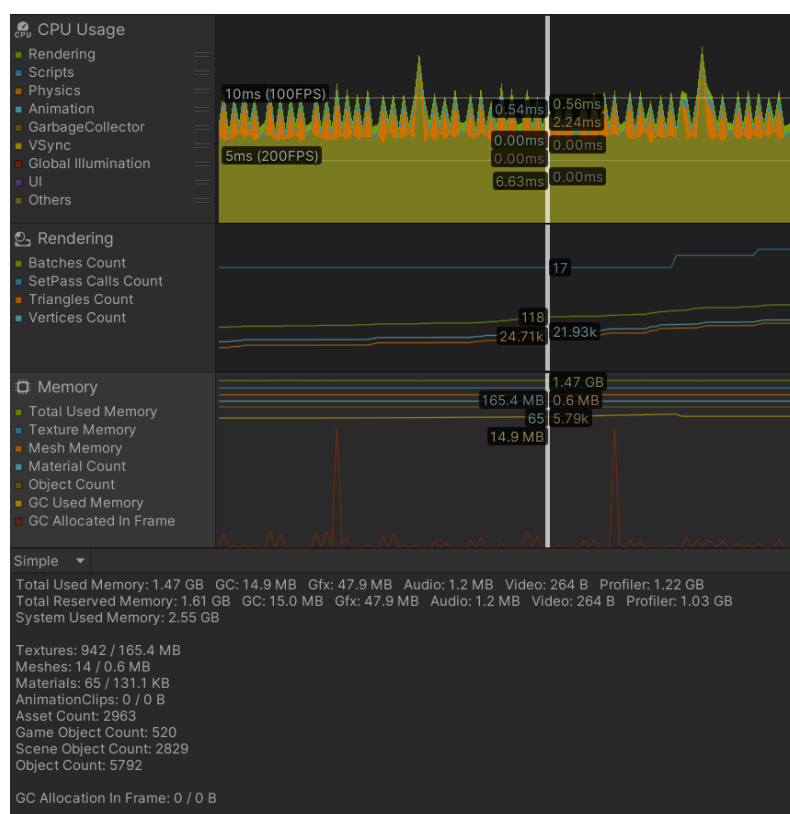


Рисунок 22 – График результатов пятого тестирования 64 агента

Результаты пятого тестирования 64 агентов: выполнение кода гибридной модели за 2.24 миллисекунды, средний показатель частоты обновления кадров чуть больше 100, общее заполнение памяти 1.47 гигабайт.

На рисунке 23 изображено шестое тестирование гибридной модели

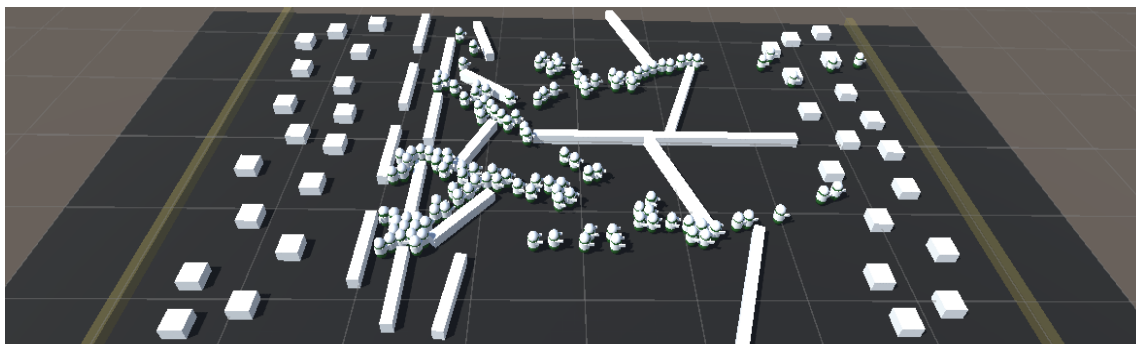


Рисунок 23 – Шестое тестирование гибридной модели 128 агентов

На рисунке 24 изображен график результатов шестого тестирования гибридной модели 128 агента.

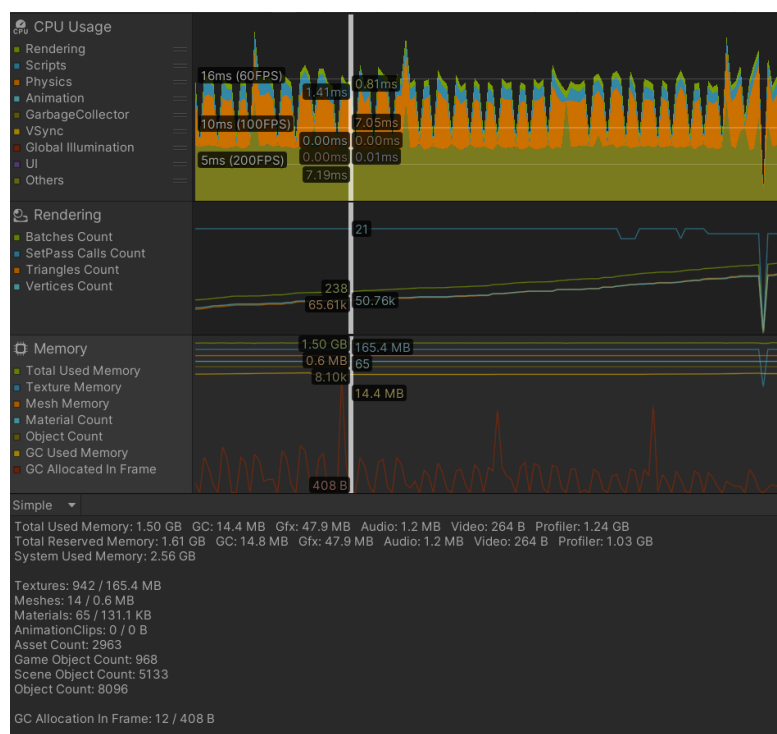


Рисунок 24 – График результатов шестого тестирования гибридной модели 128 агентов

Результаты шестого тестирования 128 агентов: выполнение кода гибридной модели за 7.05 миллисекунды, средний показатель частоты обновления кадров чуть больше 60, общее заполнение памяти 1.50 гигабайт.

На рисунке 25 изображено седьмое тестирование гибридной модели 256 агентов через 10 секунд после начала.

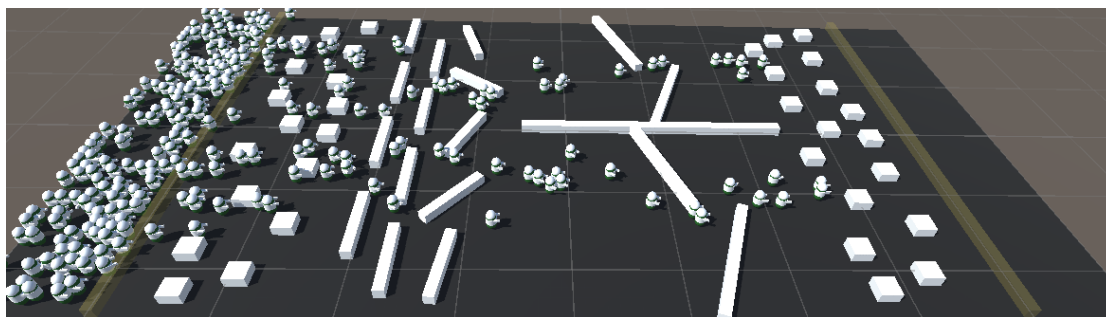


Рисунок 25 – Через 10 секунд начала седьмого тестирования гибридной модели 256 агентов

На рисунке 26 изображен график результатов седьмого тестирования гибридной модели через 10 секунд после старта.

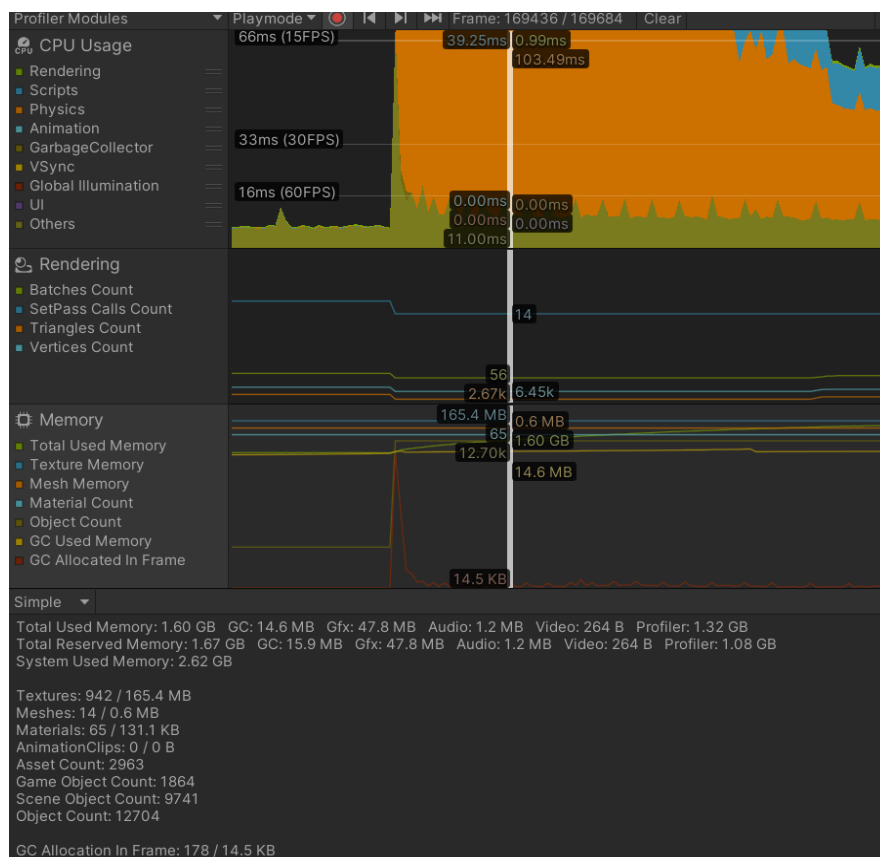


Рисунок 26. График результатов тестирования гибридной модели через 10 секунд после начала.

Результаты седьмого тестирования 256 агентов, через 10 секунд после старта тестирования: выполнение кода гибридной модели за 103.49

миллисекунды, средний показатель частоты обновления кадров ниже 15, общее заполнение памяти 1.60 гигабайт.

В течении первых 10 секунд после старта по причине выполнения кода гибридной модели 103.49 миллисекунд и обновления частоты кадров ниже 15 играть в игру нет возможности.

На рисунке 27 изображено седьмое тестирование гибридной модели через 20 секунд после старта тестирования.

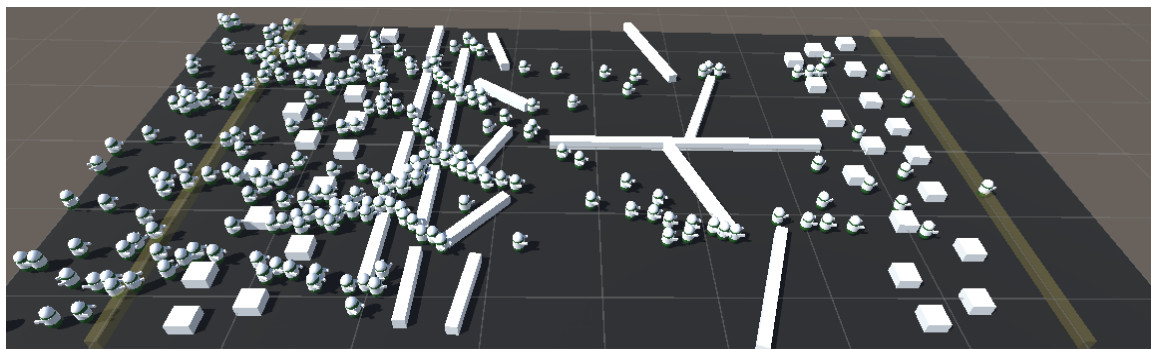


Рисунок 27 – Через 20 секунд начала седьмого тестирования гибридной модели
256 агентов

На рисунке 28 изображен график результатов седьмого тестирования гибридной модели через 20 секунд после старта.

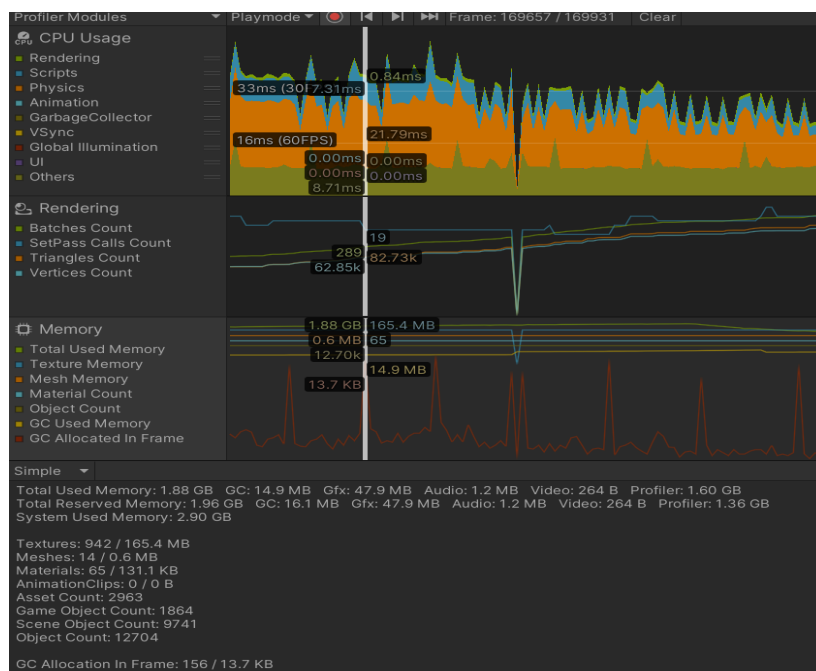


Рисунок 28 – График результатов тестирования гибридной модели через 20
секунд после начала тестирования

Результаты седьмого тестирования 256 агентов, через 20 секунд после старта тестирования: выполнение кода гибридной модели за 21.79 миллисекунды, средний показатель частоты обновления кадров 30, общее заполнение памяти 1.88 гигабайт. Через 20 значения выполнение кода гибридной модели стали ниже, и частота обновления кадров стало стабильным и подходящим для игры.

На рисунке 29 изображено восьмое тестирование гибридной модели через 20 секунд после старта тестирования.

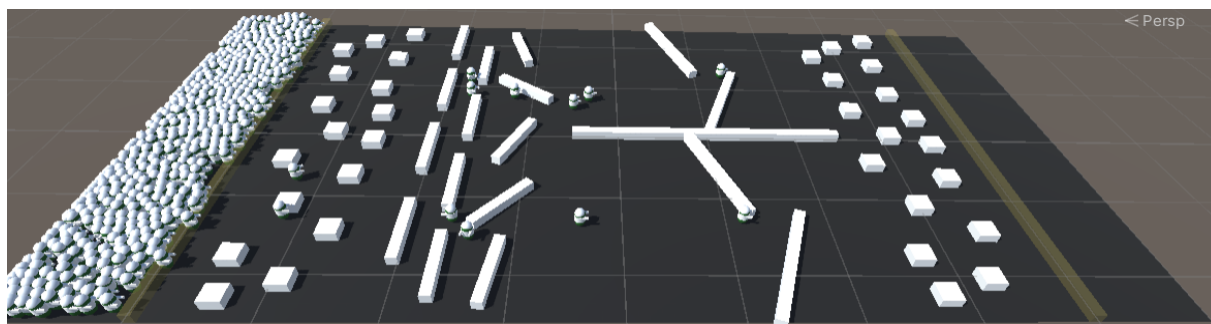


Рисунок 29 – Через 20 секунд начала восьмого тестирования гибридной модели
512 агентов

На рисунке 30 изображен график результатов восьмого тестирования гибридной модели через 20 секунд после старта.

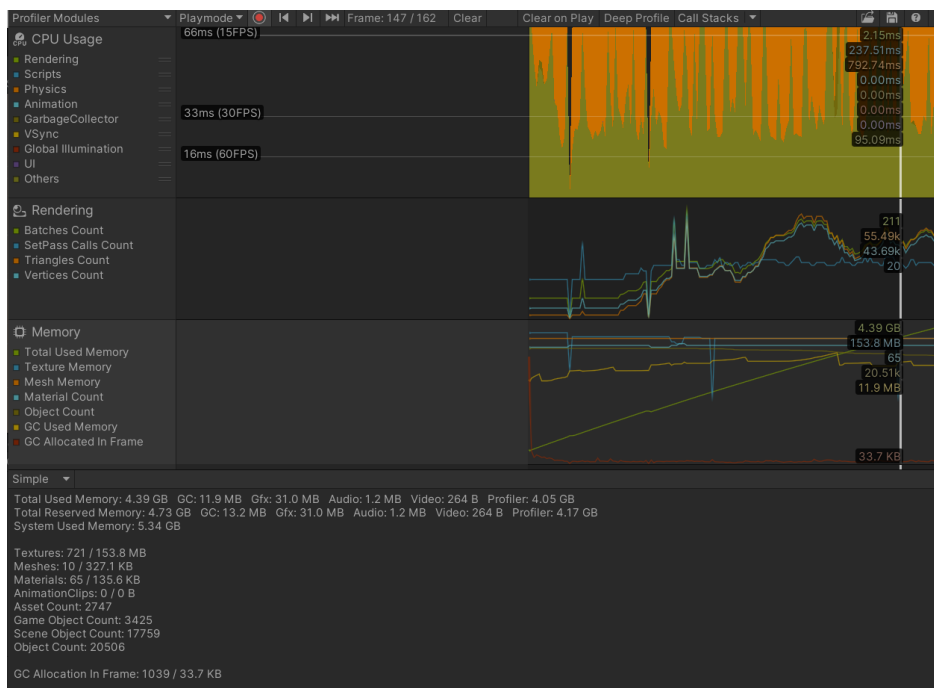


Рисунок 30 – График результатов тестирования гибридной модели через 20
секунд после начала тестирования

Результаты восьмого тестирования 512 агентов, через 20 секунд после старта тестирования: выполнение кода гибридной модели за 792.74 миллисекунды, средний показатель частоты обновления кадров 1, общее заполнение памяти 4.39 гигабайт. Через 20 значений выполнение кода гибридной модели не изменили и остаются высокими не возможными для игры.

Выводы

В ходе разработки гибридной модели были разработаны и использованы диаграммы активности и диаграммы классов. Диаграммы активности позволили определить последовательность действий и потоки управления в рамках гибридной модели. Они предоставили возможность описать различные сценарии поведения и процессы, включая передвижение, взаимодействие агентов и поиск пути. Диаграммы активности помогли лучше структурировать и визуализировать логику работы системы.

Диаграммы классов, в свою очередь, предоставили возможность определить компоненты системы, а также связи между ними. Они помогли сформировать модульную структуру гибридной модели, выделить ключевые классы и их взаимодействие. Диаграммы классов способствуют более точному пониманию архитектуры системы и упрощают процесс разработки и сопровождения кода.

В рамках данного исследования была разработана и протестирована гибридная модель передвижения агентов в игровом движке Unity. Для тестирования производительности и эффективности модели была создана специальная карта, на которой агенту необходимо пройти маршрут от одной стороны до другой, преодолевая препятствия. По результатам тестирования можно выяснить что разработанная гибридная модель может стабильно функционировать на 128 агентов. На 256 агентах программе нужно некоторое время на расчеты, после чего модель стабильно работает на частоте обновления кадров 30. Тестирование на 512 агентах показала что спустя 20 секунд после старта значения ресурсопотребления не снизились возможности для игры нет.

Заключение

В ходе выполнения диссертационного исследования был проведен анализ известных моделей поведения толпы, что позволило изучить основные принципы и подходы к моделированию поведения толпы персонажей. Было выявлено, что в ситуациях паники агенты толпы обладают характерными особенностями и стремятся максимально быстро и безопасно уйти от опасности.

На основе проведенного анализа была разработана модель поведения толпы неигровых персонажей компьютерной игры в ситуации паники. Эта модель учитывает кооперирование агентов, их индивидуальные параметры и стремление к безопасности. Агенты реагируют на опасность, изменяют свое движение и выбирают кратчайший безопасный путь для ухода от угрозы.

Для реализации алгоритма поиска кратчайшего безопасного пути при возникновении угрозы были использованы принципы алгоритма A*. Этот алгоритм позволяет эффективно находить оптимальный путь с учетом препятствий и угроз в окружающей среде.

На основе разработанной модели поведения и алгоритма поиска пути была создана компьютерная экшен-игра. В этой игре игрок имеет возможность наблюдать за неигровыми персонажами, которые реализуют разработанные модели и алгоритмы поведения в состоянии паники. Такая игра позволяет практически проверить эффективность и реалистичность разработанных моделей и алгоритмов.

Выполнение диссертационного исследования привело к созданию модели поведения толпы неигровых персонажей в состоянии паники и соответствующего алгоритма поиска пути. Разработанные модели и алгоритмы были успешно реализованы в компьютерной экшен-игре, что позволило протестировать их работоспособность и эффективность. Данная работа имеет практическую ценность в области разработки игр и симуляций, а также может быть полезна для исследований в области поведения толпы и безопасности людей в экстремальных ситуациях.

Список использованных источников

1. Клеточные автоматы – реализация и эксперименты [Электронный ресурс] – Режим доступа <https://www.osp.ru/pcworld/2003/08/166226> (Дата обращения 13.04.2023)
2. Helbing D. Simulating dynamical features of escape panic / Helbing D., Farkas I., Vicsek T. // Nature 407. – 2000. – с. 487.
3. Новая программа детализировала модели поведения толпы [Электронный ресурс] – Режим доступа <https://news.itmo.ru/ru/science/it/news/6461/> (Дата обращения 04.04.2023)
4. Тюкачев Н.А. Моделирование поведения толпы с использованием локальных скалярных полей [Электронный ресурс] – Режим доступа: https://www.researchgate.net/publication/230757270_Modelirovanie_povedenia_tolpy_s_ispolzovaniem_lokalnyh_skalarnyh_polej (дата обращения 15.01.2023)
5. Простейшая модель броуновского движения и фракталы [Электронный ресурс] – Режим доступа <https://habr.com/ru/articles/672118/> (Дата обращения 2.04.2023)
6. Алгоритм стеринга [Электронный ресурс] – Режим доступа <https://www.red3d.com/cwr/steer/gdc99/> (дата обращения 10.05.2023)
7. Моделирование поведения толпы на основе интеллектуальной динамики взаимодействующих агентов [Электронный ресурс] – Режим доступа <https://bijournal.hse.ru/data/2015/06/01/1097822010/7.pdf> (Дата обращения 06.03.2023)
8. Уравнение Навье–Стокса [Электронный ресурс] – Режим доступа <https://habr.com/ru/articles/470742/> (Дата обращения 23.04.2023)
9. Social Force Model for Pedestrian Dynamics [Электронный ресурс] – Режим доступа https://www.researchgate.net/publication/1947096_Social_Force_Model_for_Pedestrian_Dynamics (Дата обращения 15.04.2023)

10. Введение в алгоритм A* [Электронный ресурс] – Режим доступа <https://habr.com/ru/articles/331192/> (Дата обращения 13.04.2023)
11. Алгоритм Дейкстры [Электронный ресурс] – Режим доступа <https://blog.skillfactory.ru/glossary/algorithm-dejkstry/> (Дата обращения 27.04.2023)
12. Что такое нейронные сети [Электронный ресурс] – Режим доступа <https://aws.amazon.com/ru/what-is/neural-network/> (Дата обращения 26.04.2023)
13. Unity игровой движок [Электронный ресурс] – Режим доступа <https://unity.com/ru/pages/more-than-an-engine> (Дата обращения 27.04.2023)
14. Crowd simulation [Электронный ресурс] – Режим доступа https://translated.turbopages.org/proxy_u/en-ru.ru.49f93792-63c44a93-43374935-74722d776562/https/en.wikipedia.org/wiki/Social_force_model (дата обращения 15.01.2023)
15. Динамика [Электронный ресурс] – Режим доступа: <https://educon.by/index.php/materials/phys/dinamika> (дата обращения 08.02.2023)
16. Денисова В.А. Моделирование динамики частиц [Электронный ресурс] – Режим доступа: http://tm.spbstu.ru/images/f/fe/Denisova_Master_Thesis.pdf (дата обращения 15.01.2023)
17. «Моделирование толпы: принципы и методы» – М.Г. Соколов
18. «Компьютерное моделирование сложных систем» – Л.В. Канторович, Е.А. Куртев
19. «Computer Simulation of Liquids» – М.Р. Allen, D.J. Tildesley
20. «Introduction to the Modeling and Analysis of Complex Systems» – Hiroki Sayama
21. «Unity in Action: Multiplatform Game Development in C#» – Joe Hocking
22. «Mathematical Models in Population Biology and Epidemiology» – Fred Brauer, Carlos Castillo-Chavez
23. «Simulation Modeling and Analysis» – Averill M. Law, W. David Kelton

24. «Mathematical Modeling: Models, Analysis and Applications» – Konstantin Borovkov
25. «Performance Testing Guidance for Web Applications» – Microsoft Corporation
26. «The Art of Application Performance Testing: Help for Programmers and Quality Assurance» – Ian Molyneaux
27. «Математические модели в биологии» – Н.Н. Воробьев
28. «Математические модели в экономике» – А.В. Шиндяков
29. «Моделирование и анализ динамических систем» – Д.Г. Кирюшин, А.В. Горбунов
30. «Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo» – Uri Wilensky, William Rand
31. «Crowd Simulation» – Daniel Thalmann, Soraia Raupp Musse
32. «The Unity Game Development Cookbook» – Paris Buttfield-Addison, Jon Manning, Tim Nugent
33. «An Introduction to Queueing Theory: Modeling and Analysis in Applications» – U. Narayan Bhat
34. «Modeling and Simulation in Science, Engineering and Technology» – Charles S. Peskin
35. «Game Testing: All in One» – Charles P. Schultz
36. «Software Testing: A Craftsman's Approach» – Paul C. Jorgensen
37. Martin Fowler, «UML Distilled: A Brief Guide to the Standard Object Modeling Language»
38. Lars Pareto, Janis Stirna, «Using UML for Modeling and Designing Large-Scale Agent-Based Systems»
39. IBM Knowledge Center, «UML basics: An introduction to the Unified Modeling Language»
40. Visual Paradigm, «UML Diagrams Guide»
41. Книга Дональда Гэнфилда «System Modeling in Cellular Biology: From Concepts to Nuts and Bolts»

42. Müller, M., & Parisi, D. (2008). Crowd simulation. *Synthesis Lectures on Computer Graphics and Animation*, 3(2), 1–123.
43. Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4), 25–34.
44. Rios–Martinez, J., & Guerra, E. (2011). A survey on crowd simulation algorithms and their applications. *Artificial Intelligence Review*, 36(3), 251–275.
45. Van den Berg, J., Lin, M. C., & Manocha, D. (2008). Reciprocal n–body collision avoidance. *ACM Transactions on Graphics*, 27(5), 1–10.
46. Karamouzas, I., Skinner, B., & Guy, S. J. (2014). Pedestrian simulation in crowd modeling: A survey. *ACM Transactions on Graphics*, 33(6), 1–29

Приложение А
ЛИСТИНГ МОДЕЛИ ПОВЕДЕНИЯ ТОЛПЫ НЕИГРОВЫХ
ПЕРСОНАЖЕЙ
(рекомендуемое)

Листинг А.1 – Исходный код модели поведения толпы неигровых персонажей

```
public class AStar : MonoBehaviour
{
    private List<Node> openList;
    private List<Node> closedList;

    public List<Node> FindPath(Node startNode, Node goalNode)
    {
        openList = new List<Node>();
        closedList = new List<Node>();

        startNode.G = 0;
        startNode.H = CalculateHeuristic(startNode, goalNode);
        startNode.Parent = null;

        openList.Add(startNode);

        while (openList.Count > 0)
        {
            Node currentNode = GetLowestFScoreNode(openList);

            if (currentNode == goalNode)
                return ConstructPath(currentNode);

            openList.Remove(currentNode);
            closedList.Add(currentNode);

            foreach (Node neighbor in currentNode.Neighbors)
            {
                if (!neighbor.IsWalkable ||
closedList.Contains(neighbor))
                    continue;

                int tentativeGScore = currentNode.G +
CalculateDistance(currentNode, neighbor);

                if (!openList.Contains(neighbor))
                    openList.Add(neighbor);
                else if (tentativeGScore >= neighbor.G)
                    continue;

                neighbor.Parent = currentNode;
                neighbor.G = tentativeGScore;
                neighbor.H = CalculateHeuristic(neighbor,
goalNode);
            }

            return null;
        }

        private int CalculateDistance(Node nodeA, Node nodeB)
```

```

    {
        int dstX = Math.Abs(nodeA.X - nodeB.X);
        int dstY = Math.Abs(nodeA.Y - nodeB.Y);

        return dstX + dstY;
    }

private int CalculateHeuristic(Node node, Node goalNode)
{
    int dstX = Math.Abs(node.X - goalNode.X);
    int dstY = Math.Abs(node.Y - goalNode.Y);

    return dstX + dstY;
}

private Node GetLowestFScoreNode(List<Node> nodes)
{
    Node lowestNode = nodes[0];

    for (int i = 1; i < nodes.Count; i++)
    {
        if (nodes[i].F < lowestNode.F)
            lowestNode = nodes[i];
    }

    return lowestNode;
}

private List<Node> ConstructPath(Node goalNode)
{
    List<Node> path = new List<Node>();
    Node currentNode = goalNode;

    while (currentNode != null)
    {
        path.Insert(0, currentNode);
        currentNode = currentNode.Parent;
    }

    return path;
}

}

public class Node
{
    public int X { get; set; }
    public int Y { get; set; }
    public bool IsWalkable { get; set; }
    public List<Node> Neighbors { get; set; }
    public Node Parent { get; set; }
    public int G { get; set; }
    public int H { get; set; }
    public int F { get { return G + H; } }
}

```

```

public Node(int x, int y, bool isWalkable)
{
    X = x;
    Y = y;
    IsWalkable = isWalkable;
    Neighbors = new List<Node>();
}

}

public class Program
{
    public static void Main()
    {
        // Создание узлов карты
        Node[,] map = new Node[20, 20];

        for (int x = 0; x < 20; x++)
        {
            for (int y = 0; y < 20; y++)
            {
                bool isWalkable = true; // Установите значение
isWalkable в зависимости от вашей карты

                map[x, y] = new Node(x, y, isWalkable);
            }
        }

        // Создание связей соседей для узлов карты
        for (int x = 0; x < 20; x++)
        {
            for (int y = 0; y < 20; y++)
            {
                Node currentNode = map[x, y];

                // Добавьте соседей currentNode в зависимости
от вашей карты и логики соседства

                // Пример: Добавление соседей сверху, снизу,
слева, справа
                if (y > 0)
                    currentNode.Neighbors.Add(map[x, y - 1]);
                if (y < 19)
                    currentNode.Neighbors.Add(map[x, y + 1]);
                if (x > 0)
                    currentNode.Neighbors.Add(map[x - 1, y]);
                if (x < 19)
                    currentNode.Neighbors.Add(map[x + 1, y]);
            }
        }

        // Задание начального и конечного узлов для поиска пути
        Node startNode = map[0, 0];
    }
}

```

```

Node goalNode = map[19, 19];

// Инициализация алгоритма A*
AStar aStar = new AStar();

// Поиск пути
List<Node> path = aStar.FindPath(startNode, goalNode);

// Вывод результатов
if (path != null)
{
    Debug.Log("Путь найден:");
    foreach (Node node in path)
    {
        Debug.Log($"X: {node.X}, Y: {node.Y}");
    }
}
else
{
    Debug.Log("Путь не найден");
}
}

}

public class InteractionAgent : MonoBehaviour
{
    [SerializeField] private AgentMover agentMover;
    [SerializeField] private InteractionFunction
interactionFunction;

    public AgentMover GetAgentMover { get => agentMover; }
    public InteractionFunction SetInteractionFunction { set {
interactionFunction = value; } }

    private void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.TryGetComponent(out AgentMover
otherAgent))
        {
            var agentsId = "";
            if (agentMover.AgentID == otherAgent.AgentID)
                return;
            if (agentMover.AgentID < otherAgent.AgentID)
            {
                agentsId = agentMover.GetStringAgentID +
otherAgent.GetStringAgentID;
            }
            else
            {
                agentsId = otherAgent.GetStringAgentID +
agentMover.GetStringAgentID;
            }
        }
    }
}

```

```

        }

        if
(interactionFunction.CheckInteractionAgents (agentsId))
        {

            otherAgent.SetFollowAgent (GetAgentMover.GetAgentParameters.end
Position ,GetAgentMover.GetAgentParameters.speed);
        }
    }

    private void OnCollisionExit(Collision other)
    {
        if (other.gameObject.TryGetComponent(out AgentMover
otherAgent))
        {
            var agentsId = "";
            if (agentMover.AgentID == otherAgent.AgentID)
                return;
            if (agentMover.AgentID < otherAgent.AgentID)
            {
                agentsId = agentMover.GetStringAgentID +
otherAgent.GetStringAgentID;
            }
            else
            {
                agentsId = otherAgent.GetStringAgentID +
agentMover.GetStringAgentID;
            }

            if
(interactionFunction.CheckAndDeleteValue(agentsId))
            {
                //otherAgent.SetDontFollow();
            }
        }
    }
}

public struct AgentParameters
{
    public int ID;
    public bool isMoved;
    public float speed;
    public float mass;
    public float distanceStoped;
    public Vector3 endPosition;
}

public class AgentMover : MonoBehaviour

```

```

{
    [SerializeField] private NavMeshAgent agent;
    [SerializeField] private InteractionAgent myInteraction;
    [SerializeField] private AgentParameters agentParameters;

    public Vector3 SetEndPosition
    {
        set
        {
            agentParameters.endPosition = value;
            CurrentEndPos = value;
        }
    }

    public int AgentID { get => agentParameters.ID; set { agentParameters.ID = value; } }
    public string GetStringAgentID { get => agentParameters.ID.ToString(); }
    public InteractionAgent GetInteractionAgent { get => myInteraction; }
    public InteractionFunction SetInteractionFunctionOnClass { set { myInteraction.SetInteractionFunction = value; } }
    public AgentParameters GetAgentParameters { get => agentParameters; }
    public bool SetIsMoved { set { agentParameters.isMoved = value; } }

    private Vector3 CurrentEndPos;

    private void OnEnable()
    {
        agentParameters = new AgentParameters();

        agentParameters.speed = Random.Range(4, 7);
        agent.speed = agentParameters.speed;
        agent = GetComponent<NavMeshAgent>();
        agentParameters.isMoved = true;

        CurrentEndPos = agentParameters.endPosition;
    }

    private void Update()
    {
        if (!agentParameters.isMoved)
            return;
        agent.destination = CurrentEndPos;
    }

    public void DestroyAgent()
    {
        Destroy(gameObject);
    }

    public void SetFollowAgent(Vector3 _newPos, float _speed)

```

```

        {
            CurrentEndPos = _newPos;
            agent.speed = _speed;
        }
    }

public class Finish : MonoBehaviour
{
    [SerializeField] private AgentSpawner spawner;

    private int agentsCount = 0;
    public int AgentsCount { set { agentsCount = value; } }

    private int subtractingAgents
    {
        set
        {
            agentsCount--;
            if (agentsCount <= 0)
            {
                EndTry();
            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.TryGetComponent(out AgentMover agent) &&
            agent.GetAgentParameters.isMoved)
        {
            subtractingAgents = 1;
            agent.SetIsMoved = false;
            agent.DestroyAgent();
        }
    }

    private void EndTry()
    {
        spawner.StartNextTry();
    }
}

public class AgentSpawner : MonoBehaviour
{
    [SerializeField] private int currentAgentCount = 1;
    [SerializeField] private AgentMover agentPrefab;
    [SerializeField] private AgentMover[] createdAgents;
    [SerializeField] private Finish finish;

```

```

        [SerializeField] private InteractionFunction
interactionFunction;
        private int agentsId = 1;

        private void Start()
        {
            StartNextTry();
        }

        private void SpawnAgents()
        {
            int count = SetCountAgents();
            createdAgents = new AgentMover[count];
            finish.AgentsCount = currentAgentCount;
            for (int i = 0; i < count; i++)
            {
                var agent = Instantiate(agentPrefab);
                agent.transform.SetParent(transform, false);
                agent.transform.localPosition = new
Vector3(Random.Range(24f,-24f), .5f, Random.Range(5f, -5f));
                createdAgents[i] = agent;
                agent.gameObject.SetActive(true);
                agent.SetEndPosition = GetEndPosititon();
                agent.AgentID = agentsId;
                agent.SetInteractionFinctionOnClass =
interactionFunction;
                agentsId++;
            }
        }

        private Vector3 GetEndPosititon()
        {
            return new Vector3(Random.Range(-24f,24f),1,45f);
        }

        private int SetCountAgents()
        {
            currentAgentCount *= 2;
            return currentAgentCount;
        }

        public void StartNextTry()
        {
            SpawnAgents();
        }
    }

    public class InteractionFunction : MonoBehaviour
    {
        [SerializeField] private List<string> interactionAgentsID =
new List<string>();
    }

```

```

[SerializeField] private float kAttraction = 0.5f;
[SerializeField] private float kRepulsion = 0.5f;

public bool CheckInteractionAgents(string agentsId)
{
    foreach (var agent in interactionAgentsID)
    {
        if(agent == agentsId)
        {
            return false;
        }
    }

    interactionAgentsID.Add(agentsId);
    return true;
}

public bool CheckAndDeleteValue(string agentsId)
{
    foreach (var agent in interactionAgentsID)
    {
        if (agent == agentsId)
        {
            interactionAgentsID.Remove(agentsId);
            return true;
        }
        else
        {
            return false;
        }
    }

    return true;
}
}

```