УПРАВЛЕНИЕ ОБРАЗОВАНИЯ ГОРОДА ПЕНЗЫ
Муниципальное бюджетное общеобразовательное учреждение
«Гимназия № 53» г. Пензы
(МБОУ «Гимназия № 53» г. Пензы)

XXV научно-практическая конференция школьников города Пензы

Моделирование поведения толпы с использованием клеточных автоматов

Выполнил:
Симонова Софья, учащаяся 11 класса
Научный руководитель:
Артюхина Елена Владимировна, старший преподаватель кафедры «Компьютерные технологии» ПГУ

Пенза, 2021

Оглавление

Введение

Зачастую поведение толпы исследовалось в рамках социологии и психологии с целью исследования событий, случающихся в группах людей, объединенных общей целью, и функционирующих как единое целое. В таких случаях люди начинают частично терять свою индивидуальность и совершать поступки в рамках общего поведения толпы. С ростом населения крупных городов, исследования поведения массовых скоплений людей в последнее время становятся все важнее.

Актуальность:

Моделирование поведения толпы помогает решать задачи в следующих областях:

- Толпа как часть транспортной системы. Оптимизация пропускной способности транспортных систем.
- Культурно-массовые и спортивные мероприятия. Действия толпы в экстремальных ситуациях (пожар, террористический акт).
- Паломничества. Проверка маршрутов массового движения людей на предмет критических областей для избегания давки.
- Политические демонстрации. Управление поведением толп.
- Пожарная эвакуация. Анализ зданий на предмет соответствия нормам безопасности.
- Визуализация в компьютерной графике. Моделирование реалистичных толп с поведенческой точки зрения.

Это позволит разработать рекомендации специалистам, помогающие обеспечивать высокий уровень безопасности в обыденных и экстремальных ситуациях, оптимизировать различные здания и сооружения с точки зрения эффективности прохождения людского потока, а также эффективно управлять транспортным потоком.

Целью работы является реализация модели поведения толпы с использованием клеточного автомата.

Задачи:

Для достижения цели необходимо решить следующие задачи:

- рассмотреть принципы работы клеточного автомата,
- провести анализ алгоритмов, используемых для моделирования толпы,
- реализовать модель поведения толпы с использованием клеточных автоматов, с применением редактора Unity3D.

.

Объектом исследования является толпа, поведение которой описывается как в нормальных, так и в экстремальных условиях.

Предметом исследования является программа, позволяющая создавать имитационную модель поведения толпы в различных ситуациях.

Новизна проекта заключается в получении новых для ученика знаний в области моделирования, клеточных автоматов и навыков программирования на языке программирования в среде разработки Microsoft Visual Studio с использованием редактора Unity3D.

Практическая ценность. Практическая ценность данной работы заключается в том, что реализованная модель поведения толпы позволяет проводить имитационное моделирование поведения толпы в различных ситуациях и условиях. Оценить как всю толпу, так и отдельных её участников. Также есть возможность разделения агентов на группы с определенными характеристиками и в процессе работы отслеживать и изменять различные параметры, чтобы рассмотреть всевозможные ситуации.

1. Теоретическая часть

1.1 Проблема поведения толпы

Толпа – неструктурированное скопление нескольких субъектов на какой-либо местности, лишенное явно признанной общности целей, но взаимосвязанное между собой сходством эмоционального состояния и общего объекта внимания. Участниками толпы могут быть любые индивиды, способные передвигаться и решать в каком направлении им дальше двигаться: это могут быть люди, роботы и животные (птицы, рыбы и т.д.).

Очевидно, что поведение каждого человека вне толпы индивидуально и обуславливается множеством факторов, связанных с его интеллектуальной деятельностью. Но, если посмотреть на ситуацию в толпе, то человек теряет свою сущность и начинает подчиняться законам стадного поведения. Следовательно, толпа приобретает новые свойства, не характерные для каждого независимого ее участника.

Существует конкретная связь коллективного человеческого поведения, как в обычных, так и в опасных, экстремальных условиях (такие как: террористические атаки, пожары, наводнения) не только с архитектурным и городским планированием, а также с проведением всевозможных массовых мероприятий (благотворительных акций, концертов, митингов) [1]. Кроме того, с некоторой частью задач о моделировании поведения толпы может быть связана с проектированием автомобильных дорог и развязок тупиков [2].

Диапазон использования моделей поведения толпы не ограничен только областью городского и архитектурного планирования, но также включает компьютерную графику, которая используется во многих современных фильмах и телевизионных программах. В дополнение к созданию модели поведения толпы их создатели также должны создавать реалистичные визуализации толпы.

Основная задача для моделирования толпы – это описать поведение многих сущностей в некоторых условиях. Распространенным представлением данной сущности является человек в толпе, но на это нет никаких фундаментальных ограничений: похожим методом моделируется поведение стай птиц, рыб, животных и различных групп роботов.

На сегодняшний день моделирование толпы является развивающейся областью науки, в большинстве благодаря увеличению населения Земли, а также процессу мировой глобализации. Также известно, что города миллионники привлекают все больше и больше иммигрантов, и одновременно из-за этого увеличивается нагрузка на транспортную сеть таких городов, так и затрудняет процессы городского планирования. Благодаря важности параметров потока людей и транспорта появился особая геоинформационная система, которая состоит из симуляторов толпы, которые позволяют измерять, оптимизировать и визуализировать такие потоки.

С годами население крупных городов [3], во всех государствах только растет (особенно в странах с тенденцией к снижению численности населения, например, в России). В таких городах большое количество объектов городского планирования, как автомагистрали и общественные места, были спроектированы и построены, не учитывая нынешнего роста численности населения в крупных городах, что на текущий момент приводит к скоплению людей, препятствующих нормальному функционированию этих объектов. Такая же проблема является более острой в момент возникновения чрезвычайной ситуации, когда нагрузка на такие объекты возрастает в несколько раз, а заторы могут привести к более медленной эвакуации и смерти людей.

Правильная модель поведения толпы даст возможность смоделировать возникновение таких ситуаций и заранее внести изменения как в уже существующие, так и в планируемые объекты:

- обнаружение узких мест на дорогах и зонах с высокой плотностью движения,
- проверка разных способов планирования объектов с целью найти наиболее оптимальный поток, смотря со стороны пропускной способности;
- предусмотреть развития событий разного рода публичных мероприятий (благотворительных акций, концертов, митингов) в условиях чрезвычайной ситуации.

1.2 Методы для создания моделей поведения толпы

На данный момент есть много различных подходов для создания модели поведения толпы, и разделить их можно на несколько видов:

- подход с использованием клеточных автоматов. В таких методах место, где происходит движения агентов, отображено в виде набора ячеек, образующих периодическую сетку с заранее определенными правилами перехода. Эти правила определяют текущее состояние клетки в следующий момент времени, используя состояния клеток, находящихся от этой по соседству на данный момент;
- подход, основанный на физических процессах жидкостной и газовой динамики. В этом случае каждый элемент толпы является частицей, состояние которой определяется уравнениями динамики жидкости или газа (такие, как уравнения Навье-Стокса или Бернулли);
- подход, основанный на Ньютоновской механике. В такого рода подходах все силы, которые действуют на агента в толпе, отображаются в виде физических сил, а движение агента происходит с использованием второго закона Ньютона;
- подход, основанный на мультиагентных методах. Что касается этого подхода, то в процессе моделирования динамическая система не описывается полностью, а используется набор элементарных правил для перемещения и взаимодействия агентов;
- гибридные подходы, использующиеся в комбинации с несколькими из вышеуказанных подходов.

1.2.1 Клеточный автомат

В 1940-е годы, для изучения роста кристаллов, Станислав Улам использовал простую модель решетки. В это же время, Джон фон Нейман работал над проблемой самовоспроизводящихся систем. Изначально теория фон Неймана основывалась на том, что один объект собирает другого. Данная модель всем известна как кинематическая. После создания такой модели, фон Нейман осознал все сложности для создания самовоспроизводящегося объекта и в приобретении нужных частей для его строительства. Тогда Улам предложил фон Нейману взять модель, которая похожа на ту, что он использовал, когда изучал рост кристаллов. Таким образом появился первый клеточный автомат.

Клеточный автомат – дискретная модель, получившая широкое изучение в таких науках, как физика, математика, моделирование, статистика и др. Представляет собой регулярную решетку клеток, каждая ячейка которой находится в состоянии из конечного множества, таких как, 0 и 1. Размер решетки не ограничен. У каждой клетки существуют соседи.

Для того, чтобы увидеть клеточный автомат в действии, нужно задать исходные состояния клеток и некоторые правила, по которым клетки будут менять свое состояние. Каждый шаг или итерация, используя правила, меняет состояния всех клеток. В основном одни и те же правила применяются для всех клеток, но это не обязательно.

При создании модели поведения толпы учитываются реальные ситуации, столкновение агентов, хаотичное движение и др. Все это позволит максимально приблизить реалистичности модели. Также рассмотрим гетерогенные модели, которых очень мало, так как они позволяют отслеживать поведения каждого агента внутри толпы, что делает модель уникальной. Будут использованы физические материалы, параметры, а также явления, которые помогут оценить корректность модели.

Даже растения регулируют получение и уменьшение газообразных веществ при помощи клеточных автоматов. Каждая устьица на поверхности листа функционируют подобно клетке клеточного автомата.

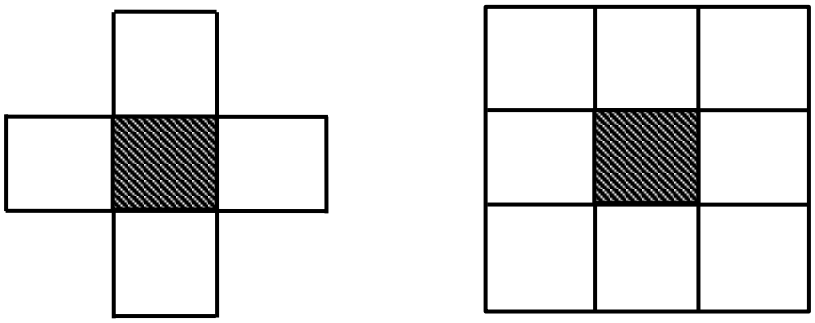
Клеточные автоматы также применяются в проектировании зданий и замкнутых поме щений[4].

Теперь рассмотрим, как клеточный автомат можно рассмотреть в математическом пре дставлении.

Клеточным автоматом называется сеть из связанных элементов, имеющих конечное множество состояний и меняющих свое состояние в дискретные моменты времени (такты $t = 1, 2, 3, \ldots$). Состояние автомата в момент $t + 1$ определяется его состоянием и состоянием

ближайших соседей (окрестности автомата) в момент t. Чаще всего рассматривают двумерные клеточные автоматы, элементами которых являются клетки. То есть двумерный клеточный автомат похож на лист бумаги в клетку. Обычно клеточный автомат функционирует в некотором замкнутом пространстве, например, например, в квадратной решетке 100×100 клеток. Иногда рассматриваются бесконечные автоматы. Обычно рассматриваются однородные клеточные автоматы, правила изменения состояний которых одинаковы для всех клеток. Если правила поведения автомата не зависят от случайных факторов, то автомат называется детерминированным, в противном случае автомат называется стохастическим. Описанные клеточные автоматы являются автоматами без памяти. Реже рассматриваются автоматы с памятью. У таких автоматов состояние, в которое переходит автомат в момент $t+1$, зависит от состояний в моменты t и $t-1$ самого автомата и его окрестности.

Как уже было сказано, состояние автомата определяется его собственным состоянием и состоянием соседних автоматов (окрестности автомата). Наиболее распространенными видами окрестностей являются окрестности фон Неймана (рис. 1.1а) и Мура (рис. 1.1б).



а) б)

Рис. 1.1. Окрестности фон Неймана (а) и Мура (б)

Окрестность фон Неймана составляют 4 соседние клетки, окрестность Мура – 8 клеток.

Клеточный автомат, необходимый для описания поведения толпы, должен быть из двух возможных состояний клетки, которые соответствуют присутствию или отсутствию в этой точке существа, а также учитывать два типа движения толпы - хаотичную и направленную.

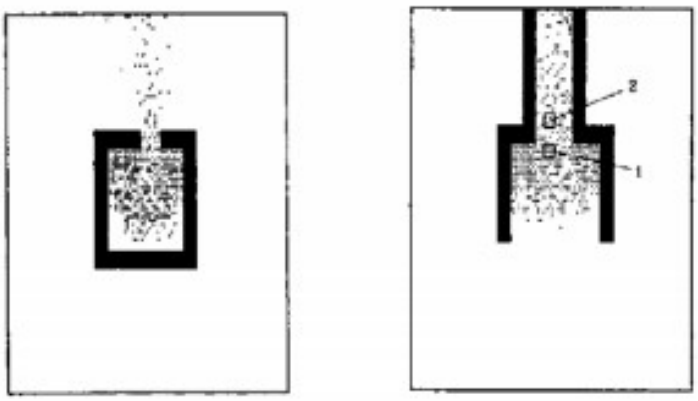


Рис. 1.2. Поведение людей в замкнутом пространстве, выходя из через узкую дверь и при движении по узкому коридору (справа)

На рисунке 1.2 представлены актуальные случаи для изучения поведения толп на основе подхода клеточных автоматов: оба рисунка показывают скопление людей в узких пространствах.

Основным преимуществом данного подхода является простота его реализации и простое описание правил, что является огромным плюсом в области моделирования в компьютерной графике. Кроме того, однородность моделируемой толпы представляет собой большой плюс для изучения однородных толп. Учитывая всю простоту, клеточные автоматы демонстрируют высокую степень соответствия между результатами моделирования и реально существующих экспериментов. Например, в [5] работе пассажиропоток в токийском метро был смоделирован на среднем размене станции Хамачи, а также проведена достоверность ре зультатов на основе реальных данных. В итоге модель показала высокий процент соотве тствия поведению реальных пассажиров в метро.

1.2.2 Мультиагентные методы

Впервые мультиагентный подход к моделированию толпы был предложен в работе Re ynolds [6], в которой приводилась модель поведения стаи птиц. Подобный объект моделирова ния был выбран по причине очень сложного характера движения стай птиц: несмотря на то, что стая состоит из отдельных птиц, общий характер движения обладает гибкостью, синхронизацией и создает впечатление наличия централизованного управления. Несмотря на это, характер такового движения складывается только из отдельных независимых поступков а гентов, сделанных на основе их локального восприятия окружающей обстановки.

В этой работе было введено понятие «боид» (англ. boid), обозначающее независимого, подобного птице агента. Боид является виртуальным агентом, который взаимодействует с подобными ему агентами путем передачи сообщений. Также в литературе, посвященной ра спределенным компьютерным системам, стаи птиц (а также косяки рыб и стада животных) представлены как устойчивые самоорганизующиеся системы.

Птица, участвующая в стае, должна обладать способностью координировать свои движения со своими соседями. Действия, объединенные в эту способность, не уникальны – все живые существа обладают ими. Реальные стаи птиц следуют двум противоположным ша блонам поведения: желание находиться как можно ближе к стае и избегание столкновения с соседями.

Данные шаблоны выработаны природой в процессе эволюции как механизм защиты от хищников, поиска пищи и размножения. Важно заметить, что способности стаи не меняются в худшую сторону при увеличении числа птиц в ней – не существует перегруженных стай.

Для процесса симуляции стаи возможно использовать следующую модель боида в виде набора правил, в процессе убывания приоритета:

1. Избегание столкновений с ближайшими соседями.
2. Согласование скорости.
3. Согласование направления движения в попытке не удаляться от центра стаи.

К достоинствам этого метода можно отнести относительно простой способ задания пра вил поведения для участников толпы (также с возможностью расширения их количества). Не достатком данного подхода является наличие довольно большого количества свободных пара метров системы, влияющих на поведение толпы в целом – выбор и верификация этих параме тров не рассматривается в большинстве работ по этой тематике.

Актуально реализовать модель поведения гетерогенной толпы на основании мультиаге нтного подхода с использованием клеточных автоматов, которые позволят более легко и эффе ктивно приблизить поведение виртуальных толп к поведению реальных. Для этого также воспользуемся 3D моделированием.

2. Практическая часть. Разработка программы

2.1 Редактор Unity3D

Для моделирования поведения толпы была использована межплатформенная среда разработки компьютерных игр Unity3D с использованием языка C# на основе клеточных автоматов.

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах.

Редактор Unity имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок использует для написания скриптов C#. Расчёты физики производит физический движок PhysX от NVIDIA. Графический API - DirectX (на данный момент DX 11, поддерживается DX 12)

Проект в Unity делится на сцены (уровни) — отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев, и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты — объекты, которые не имеют модели («пустышки»). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент Transform — он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Renderer, делающий модель объекта видимой.

К объектам можно применять коллизии (в Unity т. н. коллайдеры — collider), которых существует несколько типов.

Как правило, игровой движок предоставляет множество функциональных возможностей, позволяющих их задействовать в различных играх, в которые входят моделирование физических сред, карты нормалей, динамические тени и многое другое. В отличие от многих игровых движков, у Unity имеется два основных преимущества: наличие визуальной среды разработки и межплатформенная поддержка. Первый фактор включает не только инструментарий визуального моделирования, но и интегрированную среду, цепочку сборки, что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования. Под межплатформенной поддержкой предоставляется не только места развертывания (установка на персональном компьютере, на мобильном устройстве, консоли и т. д.), но и наличие инструментария разработки (интегрированная среда может использоваться под Windows и Mac OS).

2.2 Алгоритм поиска кратчайшего пути A*

Поиск A* (произносится «А звезда» или «А стар», от англ. *A star*) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется **эвристической функцией** «расстояние + стоимость» (обычно обозначаемой как $f(x)$). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как $g(x)$ и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как $h(x)$).

Функция $h(x)$ должна быть **допустимой эвристической оценкой**, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации $h(x)$ может

представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

Этот алгоритм был впервые описан в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем. Это по сути было расширение алгоритма Дейкстры, созда нного в 1959 году. Новый алгоритм достигал более высокой производительности (по времени) с помощью эвристики. В их работе он упоминается как «алгоритм A». Но так как он вычисляе т лучший маршрут для заданной эвристики, он был назван A*.

Обобщением для него является двунаправленный эвристический алгоритм поиска.

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просма тривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, *весь* пройденный до неё путь. Соста вляющая *g(x)* — это стоимость пути от начальной вершины, а не от предыдущей, как в жа дном алгоритме.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение *f(x)*, после чего этот узел раскрывается. На каждом эта пе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очере ди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение *f(x)* целевой вершины не окажется ме ньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множе ства решений выбирается решение с наименьшей стоимостью.

Чем меньше эвристика *h(x)*, тем больше приоритет, поэтому для реализации очереди можно использовать сортирующие деревья.

Множество просмотренных вершин хранится в `closed`, а требующие рассмотрения пути — в очереди с приоритетом `open`. Приоритет пути вычисляется с помощью функции *f(x)* внутри реализации очереди с приоритетом.

2.3 Правила объектов

Место действия происходит на ограниченной поверхности, состоящей из клеток. По всем клеткам могут ходить агенты. Каждый агент состоит в своей или общей группе и наделен параметрами: скорость, реакция, влияние от внешних факторов. Реакцией здесь является время задержки, перед началом движения к цели, а внешними факторами будут коэффициенты, влияющие на его скорость движения.

Помимо параметров самих участник толпы, у клеток поля есть свой параметр – цена, которая появляется в зависимости от количества агентов на данной клетке.

Если на клетке 3 или более агентов, то стоимость прохода по этой клетке возрастает, иначе начинает падать до минимальной. Цена соседних клеток остается минимальной. При этом во время генерации поля, каждая клетка получает свою стоимость, до появления объектов.

Ценой является затратность прохождения по данной клетке. Это означает, что есть у агента будет 2 почти одинаковых пути, то он выберет тот, который будет дешевле. А если расстояния этих путей будет сильно отличаться, но разница длины не будет превышать затрачиваемую цену за «сложный участок», то он выберет более длинный путь. Если же путь есть только один, то участник будет замедлен на соответствующее значение.

Данная цена, позволяет имитировать различные препятствия, некоторые из которых агент не в силах преодолеть, что расширяет возможности данной модели.

Помимо этого, в процессе работы программы есть возможность изменить вид моделируемого помещения и изменить значения, чтобы увидеть более непредсказуемые последствия.

2.4 Описание приложения

Изначально модель должна состоять как минимум из агентов и поля (помещение, место действия), на котором они должны будут найти выход за минимальное время. Но у них могут быть препятствия, замедляющие их скорость, блокирующие проход и даже другие агенты могут им мешать.

Создавать поле будем процедурно, генерируя одну клетку заданное количество раз. Так как модель является трехмерной, то клетка будет состоять из двух стен (левой и нижней) и пола (Рисунок 2.1).

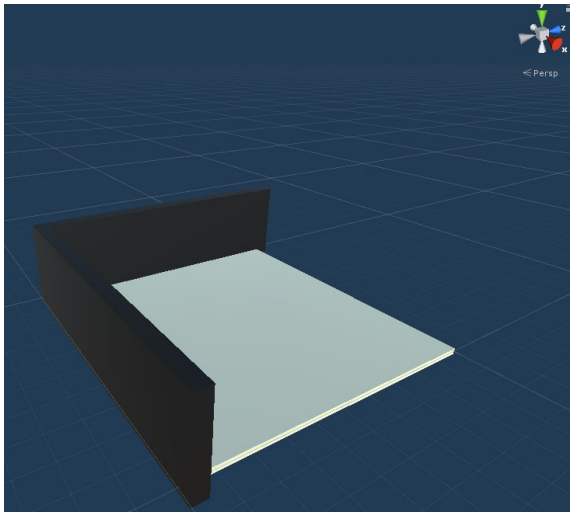


Рисунок 2.1. Трёхмерная клетка, состоящая из двух стен и пола

Для этого необходимо создать пустой GameObject, добавить в него Mesh Renderer (Рисунок 2.2), который содержит в себе много параметров, связанных с отображением объекта на сцене.

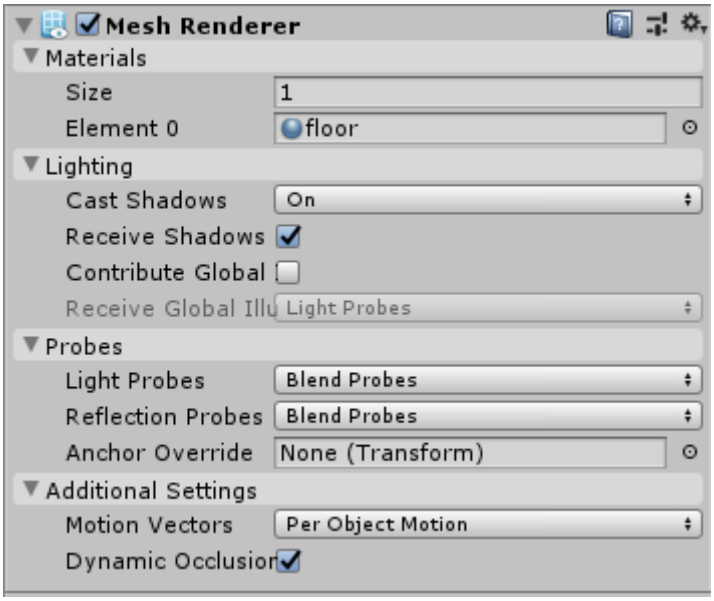


Рисунок 2.2. Mesh Renderer – настройки объекта для отображения его на сцене

И добавим в него Box Collider, который придаёт объекту физику, тем самым, не давая через него пройти другим объектам и другие возможности (Рисунок 2.3).

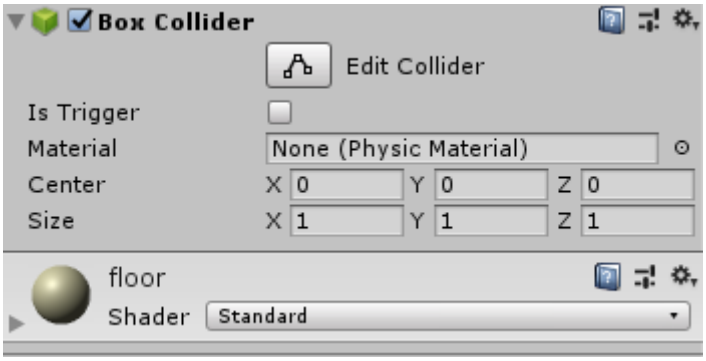


Рисунок 2.3. Box Collider – настройки объекта как физического материала

После этого нужно продублировать данные объекты с соответствующими координатами, чтобы получилась клетка как на рисунке 3.1. Объединим эти три объекта в новый пустой объект с названием Cell3D, на сцене он не нужен, так как генерироваться он будет с помощью скрипта. Создадим скрипт Cell, который будет содержать ссылки на левую и нижнюю стены и переместим в редакторе этот скрипт на префаб.

Для генерации клеток нам нужно знать место, в котором необходимо добавлять новую клетку, какие параметры необходимо добавлять для каждой клетки и т.д. Следовательно, необходимо создать новый пустой GameObject, в который добавить скрипт MazeSpawner. В этом классе нужно объявить глобальную публичную переменную, которая будет являться ссылкой на ранее созданный префаб Cell3D, и перенести в Unity префаб на публичную переменную.

Далее добавим новый скрипт, который будет отвечать за саму генерацию клеток. Для начала он будет содержать в себе ширину и высоту данного поля, который можно свободно изменить из самого редактора Unity. Данный генератор ячеек должен создавать новые клетки, чтобы не привязываться к реальному объекту создадим ещё один класс MazeGeneratorCell, который будет описывать в себе все содержимое клетки.

Листинг 1. Объект, описывающий параметры клетки:

```
public class MazeGeneratorCell
{
    public int X;
    public int Y;

    public bool WallLeft = true;
    public bool WallBottom = true;

    public bool Visited = false;

    public int DinstanceFromStart;
```

где x, y – координаты клетки, WallLeft, WallBottom – переменные, отвечающие за наличие соответствующей стены, по умолчанию присутствуют все стены, поэтому true, Visited – обозначает посещали ли мы это клетку, DinstanceFromStart – расстояние от начала, которое вычисляется для поиска последней клетки, чтобы убрать одну из стен.

Теперь мы можем создать двумерный массив ячеек MazeGeneratorCell, где задается двумерный массив заданных размеров, далее двумя циклами мы проходим по последним рядам верхней и правой сторон и убираем лишние, соответствующие стены. Далее вызвав функцию GenerateMaze из MazeSpawner и создав реальные объекты на основе абстрактных мы получим сетку из клеток.Теперь необходимо убрать некоторые стены, чтобы была возможность покинуть данное поле. Для этого используем алгоритм Recursive Backtracking, мы получим результат на рисунке 2.4.

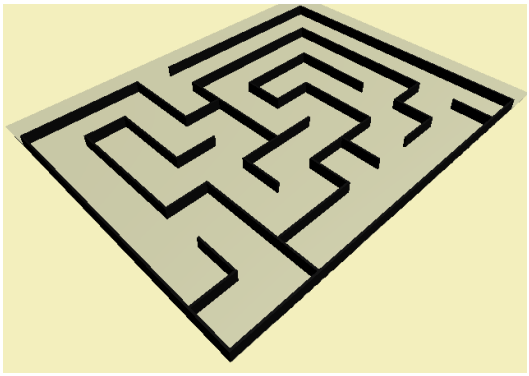


Рисунок 2.4. Полученные лабиринт с помощью алгоритма Recursive Backtracking

Теперь при каждом запуске будет генерироваться процедурно сгенерированный лабиринт с различными направлениями.

Теперь необходимо добавить новый объект движения. Создадим пустой объект 3D Capsule и добавим в него Mesh Renderer (для отображения), CapsuleCollider (для физических явлений) и Rigidbody, который управляет положением объекта через имитацию физики (Рисунок 2.5).

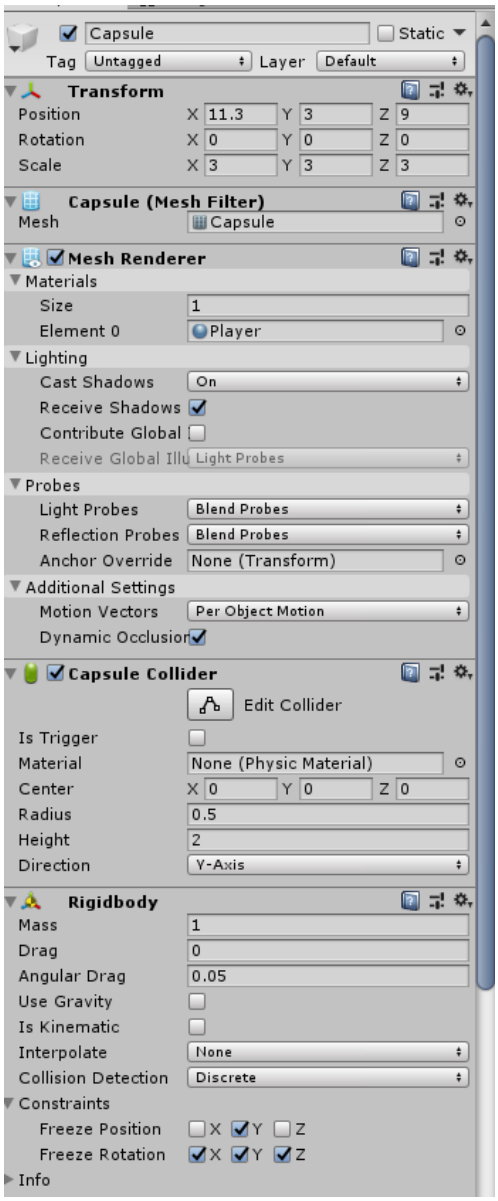


Рисунок 2.5. Mesh Renderer, Capsule Collider и Rigidbody – настройки для объекта движения, имитирующие физические явления

Теперь, чтобы вычислить клетки, по которым сможет пройти объект, нужно передать созданный объект NavMesh в скрипт MazeSpawner, где после генерации лабиринта вызвать метод BuilNavMesh.

Алгоритм NavMeshSurface, для вычисления клеток, по которым сможет пройти определенный тип объектов (Рисунок 2.6). Вычислив клетки, по которым может пройти определенный тип объектов, нужно задать свойства объектам движения под названием NavMeshAgent (Рисунок 2.7).

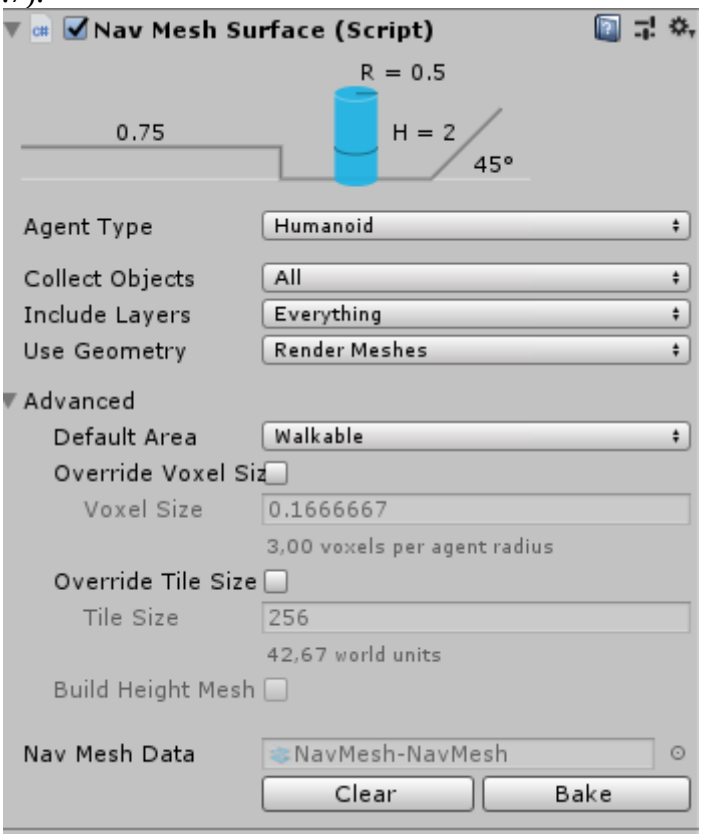


Рисунок 2.6. Данный скрипт вычисляет места, по которым сможет пройти объект заданного типа

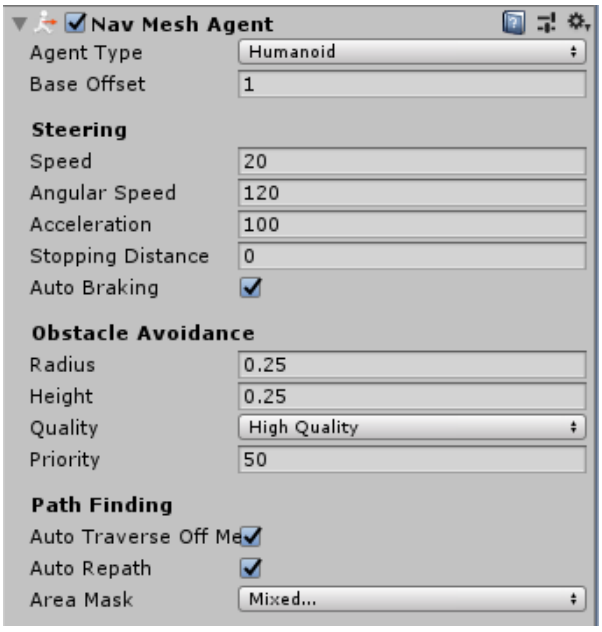


Рисунок 2.7. Задание типа NavMeshAgent для объектов движения

Для каждого объекта можно выбрать свой тип агента. Благодаря этому их можно группировать на разные толпы, задавать каждому свои параметры.

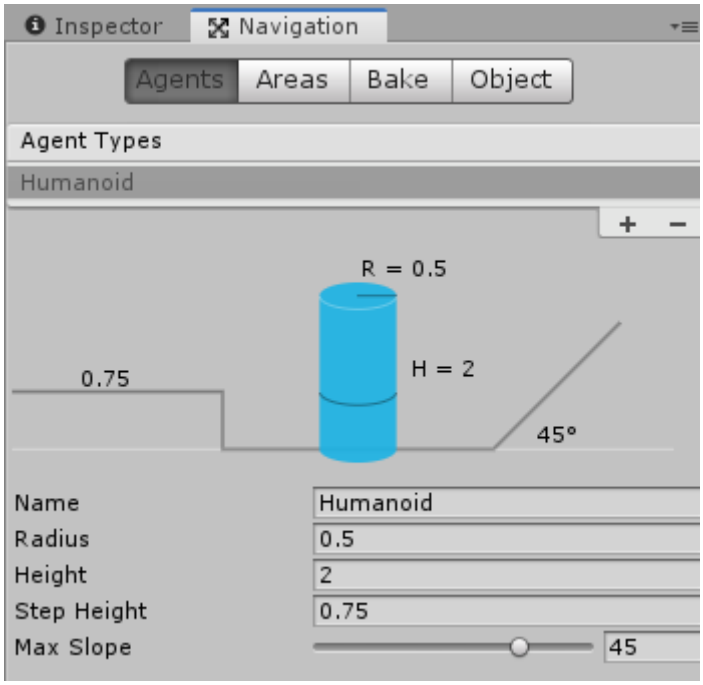


Рисунок 2.8. Взаимодействие агентов между собой

На рисунке 2.8 показана возможность задания характеристик для взаимодействия агентов разного типа между собой, а также и другие параметры, как стоимость движения по клетке (Рисунок 2.9).

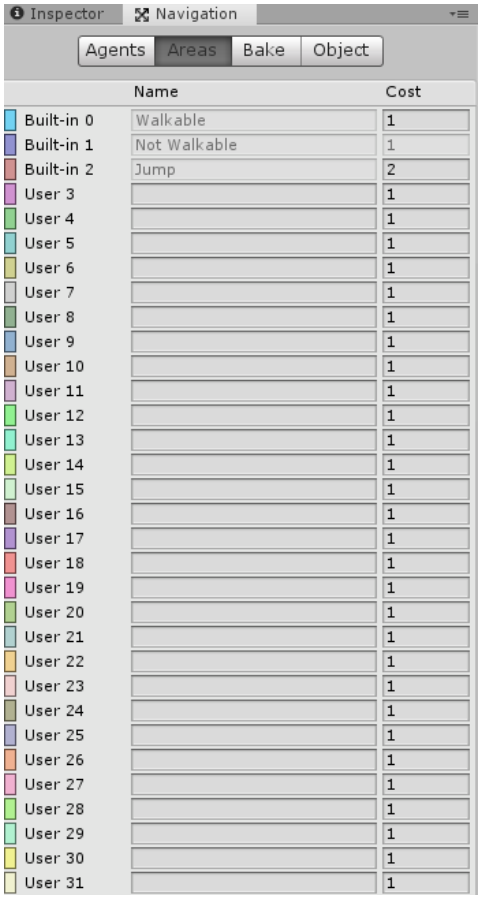


Рисунок 2.9. Типы клеток со своей стоимостью

На каждую клетку можно задать свой тип участка (walkable, not walkable, jump и т.д.) за свою цену. Под ценой понимаем – затратность прохождения по данной клетке. Это означает, что если у агента будет 3 почти одинаковых пути, то он выберет тот, который будет дешевле. А если расстояния этих путей будет сильно отличаться, но разница (расстояний) не будет

превышать затрачиваемую цену за «сложный участок», то он выберет более длинный путь. Цена появляется в зависимости от кол-ва объектов на данной клетке. Если на клетке 3 или более объектов, то цена прохода по клетке возрастает, при этом цена у соседних клеток остаётся минимальной.

На данном этапе у нас есть только один путь к выходу, но во время работы программы любую стену можно убрать и с имитировать различные помещения для имитации (Рисунок 2.10).

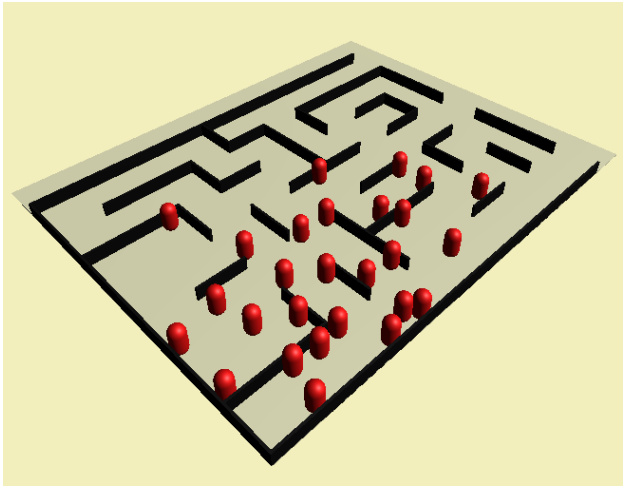


Рисунок 2.10. Возможность корректировки лабиринта

Теперь можно добавить цель, к которой будут двигаться агенты. Для этого необходимо для каждого агента добавить скрипт, который ждёт клика мышкой и идёт по кратчайшему для себя пути к этой цели.

Листинг 2. Метод, в котором агент получает цель:

```
void Update()
 {
 if (Input.GetMouseButtonDown(0))
 {
 RaycastHit hit;
 if (Physics.Raycast(mainCamera.ScreenPointToRay(Input.mousePosition), out hit))
 {
 agent.SetDestination(hit.point);
 }
 }
 }
```

Для разных типов агентов можно добавить свои обозначения для старта движения.

2.5 Интерфейс

Рисунок 2.11 является стартовым окном программы. В левой части расположено список объектов, созданных на данной сцене (Рисунок 2.11). В правой части расположены все доступные для изменения параметры выбранного в левой части объекта (Рисунок 2.12).

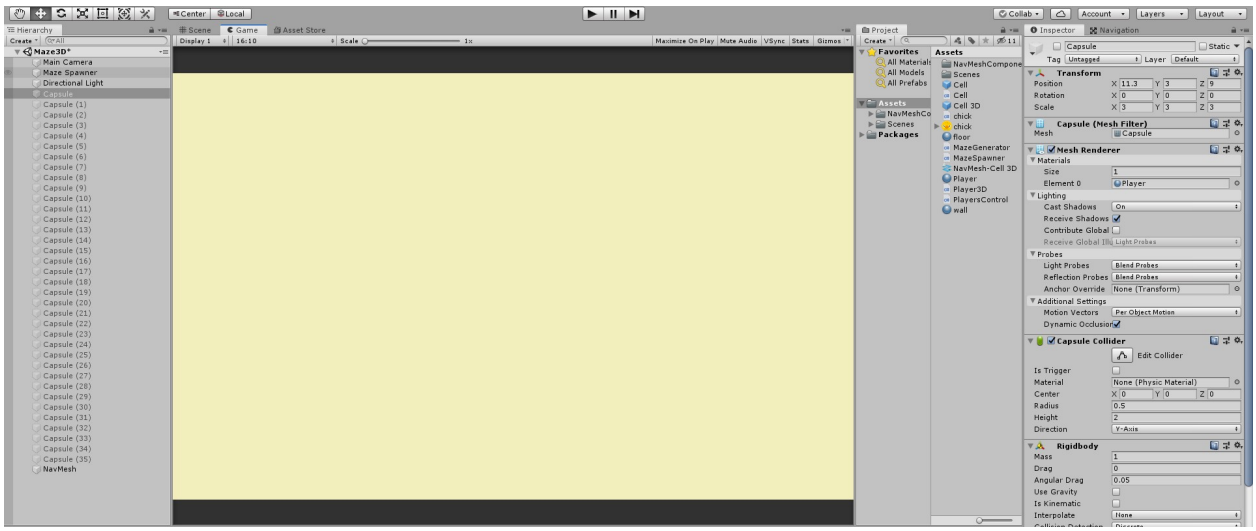


Рисунок 2.11 – Стартовое окно

Рисунок 2.12 – Список объектов на сцене

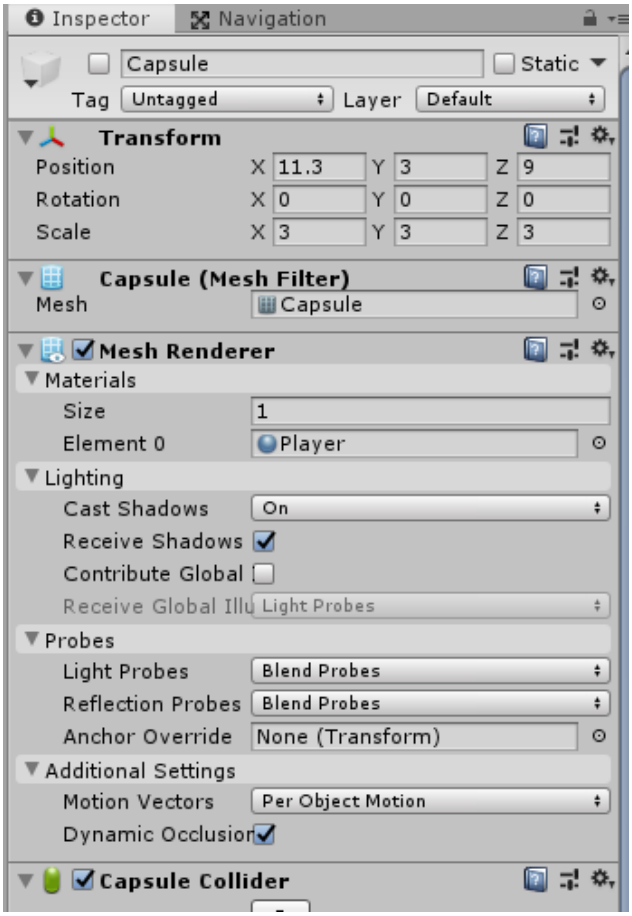


Рисунок 2.13 – Параметры выбранного объекта

В верхней части находится блок управления сценой, в котором можно запустить и поставить на паузу работу программы (Рисунок 2.14).

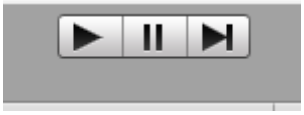


Рисунок 2.14 – Блок управления программой

А по центру находится сама отрисовка элементов программы, где можно отрыть редактируемую зону, под названием Scene, и зону Game, в которой будет отображена работа программы (Рисунок 2.15). На Scene можно управлять камерой, выставить под нужным углом, поставить свет и многие другие элементы.

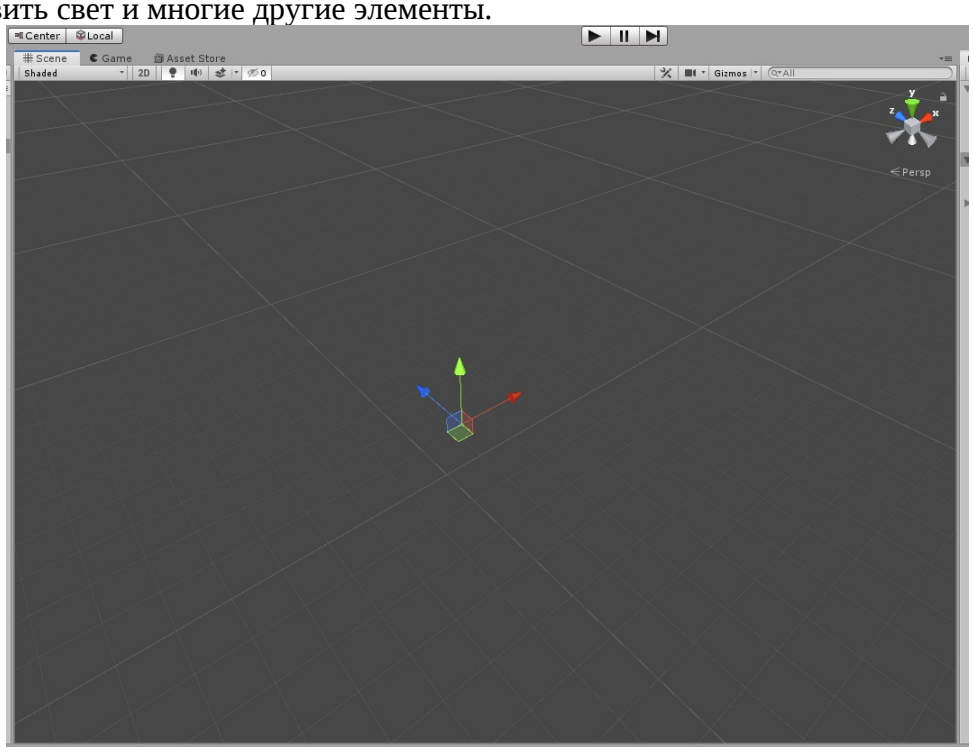


Рисунок 2.15 – Сцена, для отрисовки работы программы

2.6 Работа программы

Изначально на сцене процедурно генерируется сетка из клеток и из этой сетки делается лабиринт, с помощью алгоритма Recursive Backtracking. Далее в случайном порядке добавляются агенты на это поле. После этого, с помощью клика мыши, даем команду движения к выбранной цели и видим это на рисунке 2.16.

У каждого участника есть метод для нахождения кратчайшего пути с помощью алгоритма A*. Некоторые из агентов объединены в группы, с определенными параметрами, которые имитируют, например, людей определенной возрастной категории (старше 60 лет). Для имитации различных слоев населения в методе NavMeshAgent есть очень много различных параметров. Это позволяет разделить толпу на разные типы участников. Также некоторые люди могут находиться в какой-либо группе одни, так как людей, с похожим человеческими характеристиками, может не найтись.

Еще можно представить работу программы в произвольном помещении, которое легко изменяется во время работы программы (Рисунок 2.17).

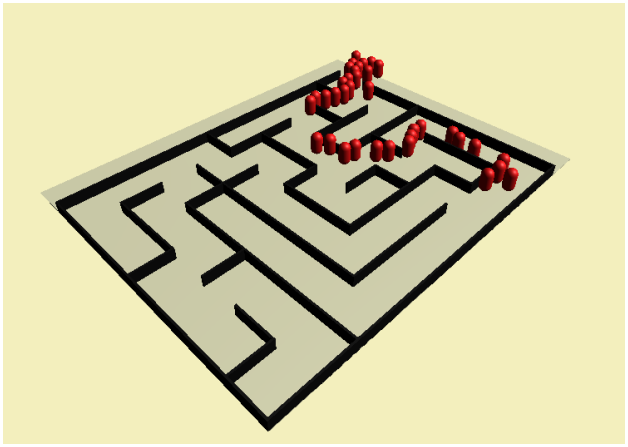


Рисунок 2.16. Работа программы в лабиринте

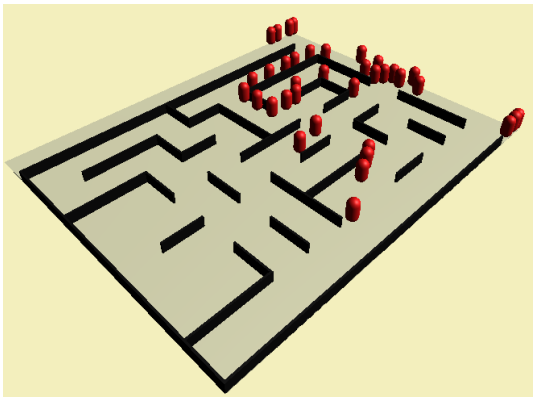


Рисунок 2.17. Работа программы в произвольном помещении

Как видно по рисункам 2.16 и 2.17, все участники находятся в движении, и никто из них не застревает. Каждый из агентов двигается к своей цели и со своей скоростью через определенные препятствия.

В целях проверки программы, были смоделировано поведение толпы с различными группами внутри толпы и их параметрами 100 раз, из которых только в 7-х случаях некоторые агенты не смогли дойти до цели.

Данная модель использует методы для имитации физики, как на клетках, так и на агентах. Генерируемая процедурным способом лабиринт с каждым запуском выдает уникальные помещения, которые можно изменить под нужный вид. Используется большое количество параметров для настройки всех объектов, что позволяет добиться более точных результатов. Также эта модель дает возможность изменять параметры во время работы, что делает ее гибкой и позволяет спрогнозировать очень редкие ситуации.

2.7 Проверка модели

В качестве примера имитационного моделирования толпы в помещении рассмотрим ситуацию в ночном клубе «Хромая лошадь», событие которого произошло 5 декабря 2009 года [7]. Несмотря на то, что стандартная вместимость клуба позволяет находиться в нем одновременно только 50 посетителям, в этот день в помещении находилось 283 человека (143 мужчины и 140 женщин). Учитывая, что руководство и персонал заведения покинули помещение клуба через черный ход, в расчетах используем 250 агентов, предполагая, что количество мужчин и женщин было одинаковым. В результате неосторожного применения пиротехники в 01:08 по местному времени в клубе начался пожар. По счастливой случайности в зале находился оператор, который снимал все дальнейшие события, не выключая камеру до самого выхода из клуба. Это дает возможность установить хронологию событий с точностью до секунд. Как показывает изучение видеоматериалов трагедии, пожар возник над сценой вблизи перегородки между комнатами № 5 и № 7 на рисунке 2.18. Всего с начала появления первой

искры, зафиксированного на видео, до обрушения потолка и гибели людей прошло 3 минуты и 22 секунды. Такой небольшой промежуток времени объясняется быстрым распространением материала по потолку, который был выложен высокогорючим материалом. Кроме того, печальную роль сыграл выход руководства клуба через черный выход, который привел к возникновению сильной тяги воздуха вдоль потолка и резкому усилению пожара.

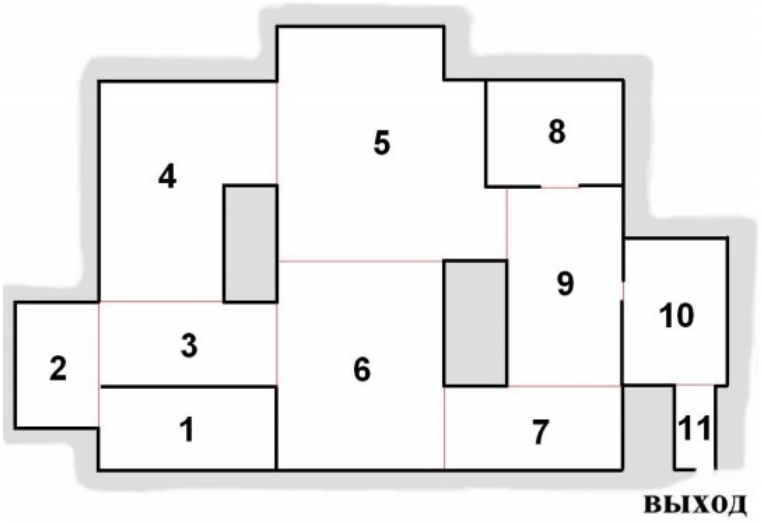


Рисунок 2.18. Схема клуба «Хромая лошадь»

Реализуем схему данного помещения в программе и расположим агентов (Рисунок 2.19).

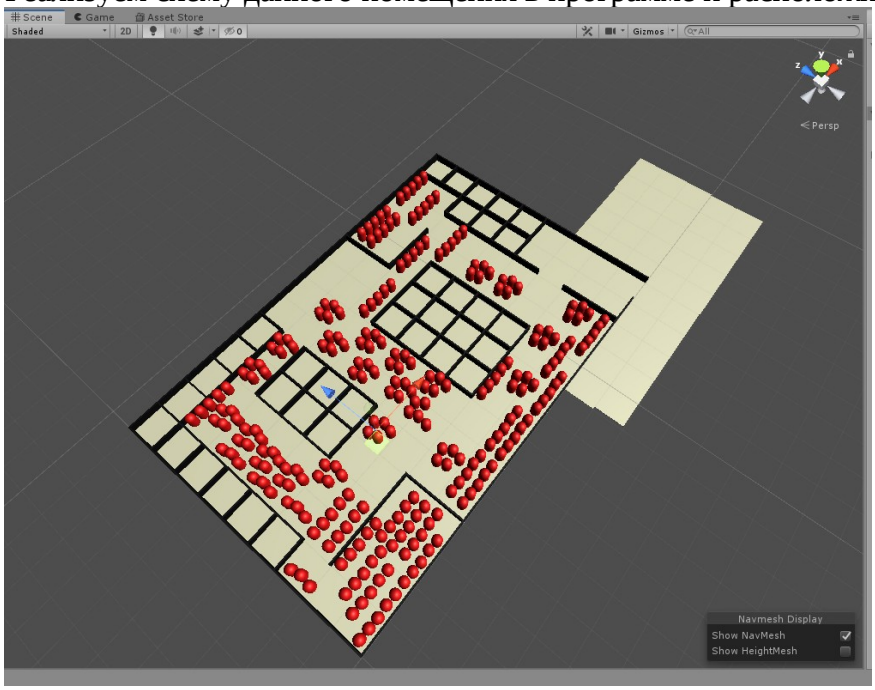


Рисунок 2.19. Реализована примерная схема помещения ночного клуба «Хромая лошадь»

Рассмотрим происходящее в программе через 5, 15 и 60 секунд после начала эвакуации, учитывая количество агентов равно 250. Препятствия в виде столов и стульев отображены не были, но учитывались замедления от состояния клеток, что способствует получению помех в движении, которые нужно обходить.

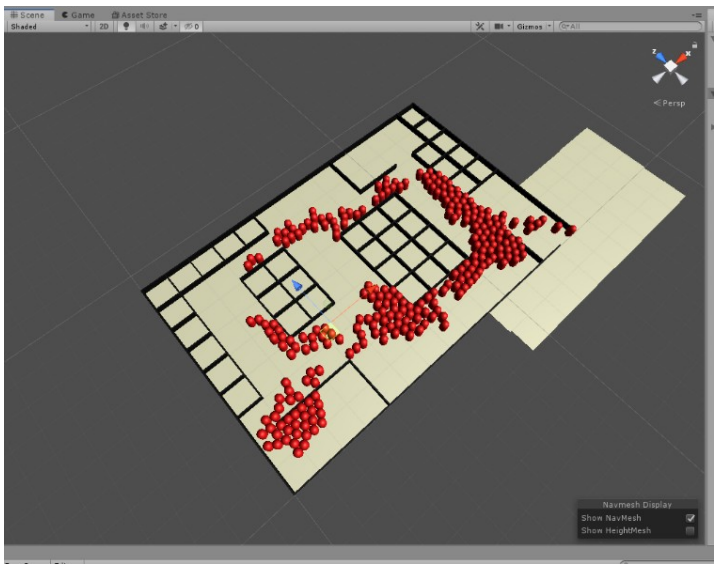


Рисунок 2.20. Пройденное время 5 секунд

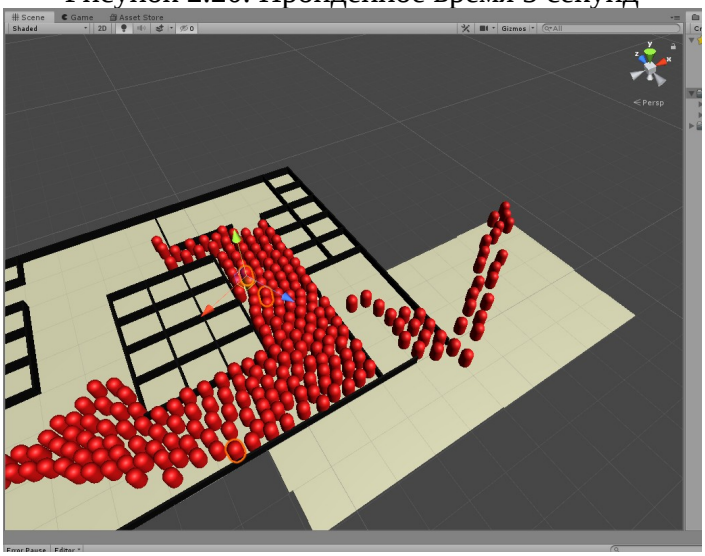


Рисунок 2.21. Пройденное время 15 секунд

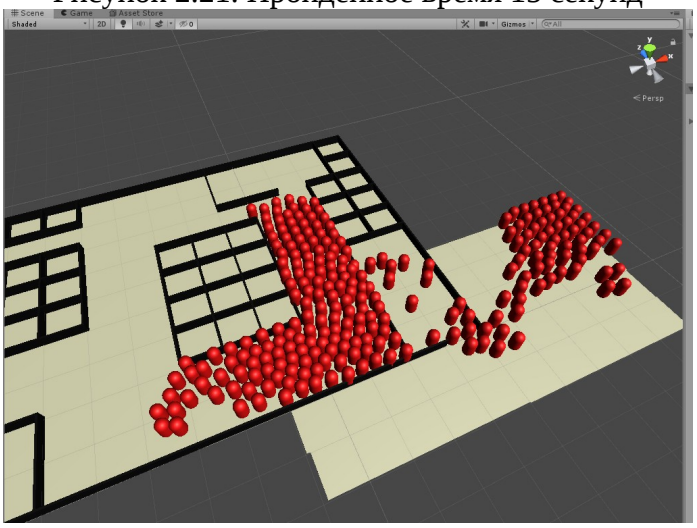


Рисунок 2.22. Пройденное время 60 секунд

После начала эвакуации люди направляются к выходу из клуба (рисунок 2.20). В этот момент выход уже начинается блокироваться из-за слишком большого количества людей, но при этом люди пока могут поочередно покидать помещение. Через некоторое время около выхода в гардероб образуется арка из людей, и он блокируется толпой. Система переходит к нестационарному типу движения потока людей. Время от времени арка обрушивается, и

очередная небольшая порция людей выходит из клуба (рисунок 2.22). После момента времени равному 100 секунд всех людей, оставшихся в зале клуба, можно считать погибшими.

На рисунке 2.23 представлен график зависимости количества вышедших из помещения клуба индивидов от времени. Хорошо видно, что график делится на две части, каждая из которых может быть аппроксимирована прямой. Пересечение прямых указывает нам на важный момент времени (примерно 13-ая секунда эволюции системы), после которого изменяется сценарий эвакуации людей. Этот излом кривой, условно говоря, можно назвать началом паники. Из графика видно, что именно паника является главной причиной гибели массы людей. Если экстраполировать прямую, аппроксимирующую первоначальный участок кривой, то оказывается, что через 100 секунд помещение могло покинуть порядка 250 человек!

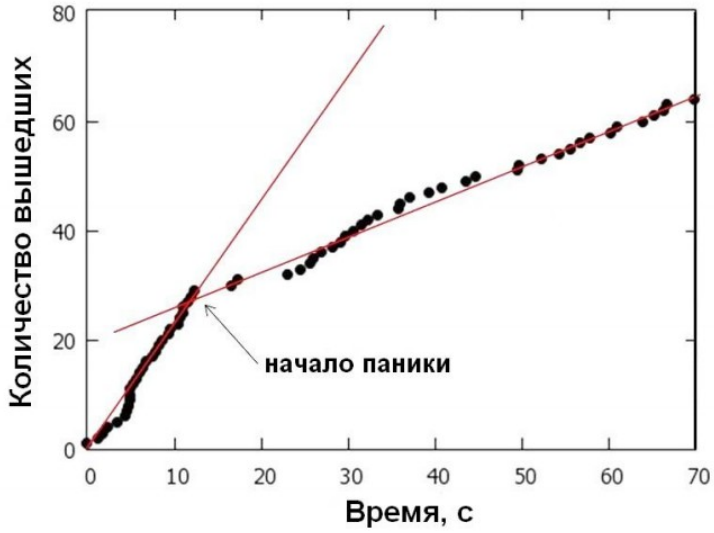


Рисунок 2.23. График зависимости количества вышедших из клуба индивидов от времени

Тем не менее, спустя около 13 секунд модель движения переходит к другому динамическому режиму, при котором через 100 секунд помещение покидает всего 80 человек, что хорошо согласуется с хроникой реальных событий в «Хромой лошади».

Заключение

В процессе работы были выполнены все поставленные задачи и достигнута цель.

Рассмотрены принципы работы клеточного автомата. Проведен анализ мультиагентного подхода к моделированию толпы.

Реализована модель поведения гетерогенной толпы с использованием клеточных автоматов, которые позволят более легко и эффективно приблизить поведение виртуальных толп к поведению реальных. Вся работа выполнена на языке программирования C# в среде разработки Microsoft Visual Studio с использованием редактора Unity3D.

Полученная в этой работе модель, может помочь в прогнозировании редких ситуаций, моделировать поведение толпы, разделяя ее на различные подгруппы. Также можно отметить использование методов имитации физики, которые широко используется в современных играх и показывает очень достойные результаты.

Была смоделирована реальная ситуация в ночном клубе «Хромая лошадь», пожар в котором произошёл 5 декабря 2009 года. И результаты были близки к реальным.

Список литературы и источников

1. Аптуков А.М., Моделирование групповой динамики толпы, паникующей в ограниченном пространстве / Аптуков А.М., Брацун Д.А. // Вестник Пермского университета. Серия: Математика. Механика. Информатика, 2009. – №3. – с. 18-23.
2. Аптуков А. М., Брацун Д. А., Люшнин А. В. Моделирование поведения паникующей толпы в многоуровневом разветвленном помещении // Компьютерные исследования и моделирование. — 2013. — Т. 5, № 3. — С. 491–508.
3. Максимова М. З., Барбин Н. М. Моделирование эвакуации людей различных возрастных групп // Компьютерные исследования и моделирование. — 2013. — Т. 5, № 3. — С. 483–490.
4. Акопов А. С. Имитационное моделирование: учебник и практикум для академического бакалавриата. — М.: Юрайт, 2014. — 389 с.
5. Beklaryan A. L. On the existence of solutions of the first boundary value problem for elliptic equations on unbounded domains // Russian Journal of Mathematical Physics. — 2012. — Vol. 19, no. 4. — Pp. 509–510
6. Таранцев А.А. Моделирование параметров людских потоков при эвакуации с использованием теории массового обслуживания.// Пожаровзрывобезопасность. – М: Пожнаука, 2002. – №6.
7. https://ru.wikipedia.org/wiki/Пожар_в_клубе_«Хромая_лошадь»

Рецензия на работу «Моделирование поведения толпы с использованием клеточных автоматов»
Автор: учащаяся 11 класса МБОУ «Гимназия № 53» г. Пензы
Симонова Софья
Научный руководитель: Артюхина Елена Владимировна

Работа Симоновой Софьи посвящена актуальной теме моделированию поведения толпы. Основная задача для моделирования толпы – это описать поведение многих сущностей в некоторых условиях. Благодаря важности параметров потока людей и транспорта появился особая геоинформационная система, которая состоит из симуляторов толпы, которые позволяют измерять, оптимизировать и визуализировать такие потоки

В своей работе Симонова Софья ставит целью реализовать модель поведения толпы, с использованием редактора Unity3D.

В ходе теоретических исследований были изучены проблемы поведения толпы, методы для создания моделей поведения толпы, клеточные автоматы, мультиагентные методы.

Реализована модель поведения гетерогенной толпы с использованием клеточных автоматов, которые позволят более легко и эффективно приблизить поведение виртуальных толп к поведению реальных. Вся работа выполнена на языке программирования C# в среде разработки Microsoft Visual Studio с использованием редактора Unity3D.

Полученная в этой работе модель, может помочь в прогнозировании редких экстремальных ситуаций. Также можно отметить использование методов имитации физики, которые широко используется в современных играх и показывает очень достойные результаты. Была смоделирована реальная ситуация в ночном клубе «Хромая лошадь», пожар в котором произошёл 5 декабря 2009 года. И результаты были близки к реальным.

Пояснительная записка к работе составлено грамотно и лаконично, в ней отражены основные моменты исследования и полученные результаты.

На мой взгляд, рецензируемая работа является законченной и имеет хорошие перспективы для дальнейшего развития. Несомненно, работа может быть представлена на научно-практической конференции школьников г. Пенза.

К.п.н., доцент кафедры «Информатика и методика обучения информатике и математике» Пензенского государственного университета Акимова И.В.

Личную подпись И.В. Акимовой
ЗАВЕРЯЮ
Специалист по кадрам
«28» 12 2020 г.