УДК 004.946
DOI: 10.26102/2310-6018/2026.54.3.006

Метод реализации псевдореалистичного перемещения неигровых персонажей в открытых виртуальных мирах

К.И. Шутов✉, А.А. Лобанов

МИРЭА – Российский технологический университет, Москва, Российская Федерация

Резюме. На современном рынке игр с открытым миром существует запрос на правдоподобное, но управляемое поведение неигровых персонажей (NPS) при ограниченных вычислительных ресурсах. Существующие решения предлагают две крайности. Либо они перегружают систему попыткой полной симуляции, либо дают предсказуемые скриптовые паттерны. Цель работы – предложить метод псевдореалистичного перемещения NPC, заполняющий разрыв между этими крайностями. Метод основан на проверке достижимости появления NPC: используется матрица кратчайших расстояний между областями игрового мира. При входе игрока в область алгоритм выбирает тех неигровых персонажей, которые физически могли туда добраться с учетом прошедшего времени, заданной скорости и доступных маршрутов, что интерпретирует встречу как следствие скрытого перемещения, а не мгновенного появления. Частота встреч регулируется системой приоритетов. Разработчик может управлять редкостью отдельных персонажей без детальной симуляции их маршрутов. Выбор кандидатов ускоряется переупорядочением почти отсортированного списка, уменьшая стоимость повторных выборок при близких входных условиях. Эксперименты на сгенерированных графах показали, что время выполнения основной части алгоритма на стороне клиента остается в пределах миллисекунд при численности до 1000 NPC. Метод обеспечивает правдоподобие и контролируемость встреч при низкой нагрузке и может быть интегрирован в существующие системы для настройки сложности и баланса.

Ключевые слова: гейм-дизайн, гейм-дев, видеоигры, алгоритм поиска пути, алгоритм сортировки, NPC, неигровой персонаж.

Для цитирования: Шутов К.И., Лобанов А.А. Метод реализации псевдореалистичного перемещения неигровых персонажей в открытых виртуальных мирах. *Моделирование, оптимизация и информационные технологии.* 2026;14(3). URL: https://moitvivt.ru/ru/journal/article?id=2190 DOI: 10.26102/2310-6018/2026.54.3.006

A method for implementing pseudo-realistic movement of non-player characters in open virtual worlds

K.I. Shutov✉, A.A. Lobanov

MIREA – Russian Technological University, Moscow, the Russian Federation

Abstract. The open-world game market increasingly demands NPC (non-player character) behaviour that feels believable yet remains designer-controllable under tight computational budgets. Common solutions tend to be extreme: either they attempt full simulation and overload the system, or they rely on predictable scripted patterns. This paper proposes a pseudo-realistic NPC movement method that bridges these extremes. The core idea is to verify spawn reachability using a matrix of shortest-path distances between world areas. When the player enters an area, the algorithm selects only those NPCs that could have physically reached it given elapsed time, movement speed and available routes, making an encounter consistent with hidden travel rather than instantaneous spawning. Encounter frequency is controlled via a priority scheme, allowing designers to tune event density and the rarity of specific characters without maintaining a detailed simulation. Candidate selection is further accelerated by reordering an almost-sorted list, reducing the cost of repeated queries under similar conditions. Experiments on synthetic graphs show that the core client-side runtime stays within milliseconds for up

to 1000 NPCs. The method delivers believability and control at low computational cost and can be integrated into existing engines to adjust difficulty and balance.

Keywords: game design, game development, video games, pathfinding algorithm, sorting algorithm, NPC, non-player character.

For citation: Shutov K.I., Lobanov A.A. A method for implementing pseudo-realistic movement of non-player characters in open virtual worlds. *Modeling, Optimization and Information Technology*. 2026;14(3). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2190 DOI: 10.26102/2310-6018/2026.54.3.006

Введение

В последнее время, очень популярны крупнобюджетные игры, в которых игроку предоставляется свобода передвижения и взаимодействия с окружающим пространством без строгих ограничений и заранее заданного маршрута. Для таких игр появилось свое название − игры с открытым миром. Такой мир часто включает нелинейный геймплей, исследование территорий, динамические события и возможность выполнения заданий в произвольном порядке. Разработчики современных видеоигры с открытым миром стремятся к максимальному реализму. Однако на этом пути возникает множество проблем. Одна из них − достоверность перемещения неигровых персонажей (NPC), пока остается нерешенной.

Существующие методы основаны либо на алгоритмах искусственного интеллекта, либо на жестких сценариях [1, 2]. В первом случае алгоритмы делают поведение NPC непредсказуемым для разработчиков и усложняют балансировку игры [3]. Во втором случае делают перемещение персонажей слишком предсказуемым для игрока и уменьшают ощущение реализма в игре [4].

В игровой индустрии для построения маршрутов до цели чаще всего используют алгоритмы A* его вариации [5]. Данные подходы хорошо работают на локальном уровне, перемещая NPC в одной области с игроком. Однако при применении в больших мирах они значительно увеличивают нагрузку на вычислительные ресурсы. Некоторые разработчики используют мгновенное перемещение NPC по заранее заданным маршрутам, однако это снижает реализм игрового мира [6]. Другой подход – процедурные системы симуляции. Данный подход используется в играх серии The Elders Scrolls, где NPC имеют расписание и потребности и, исходя из них, наиболее эффективно определяют действие [7, 8]. Однако эта симуляция все еще работает в зоне рядом с игроком, а когда он из нее выходит, NPC мгновенно перемещаются на новые позиции согласно расписанию. Подобные решения приводят к нелогичным ситуация, когда персонаж оказывается там, куда физически не мог добраться. В исследованиях отмечается, что для правдоподобного поведения важно не столько учитывать перемещение персонажей, сколько управлять их появлением так, чтобы оно соответствовало ожиданиям игрока [9].

Настоящая работа предлагает метод псевдореалистичного перемещения NPC, который минимизирует вычислительную нагрузку, сохраняя при этом контроль над частотой встреч игрока с персонажами. В данной статье под методом понимается формализованный способ решения задачи, основанный на системе правил, алгоритмов и математического обеспечения, который направлен на достижение поставленных целей.

Метод основан на динамическом управлении вероятностью появления NPC в зависимости от места и времени их последней встречи с игроком. В отличие от традиционных алгоритмов поиска пути, предлагаемый в данной работе метод позволяет создавать иллюзию реалистичного перемещения без постоянного отслеживания всех

NPC в режиме реального времени, что снижает затраты на вычислительные ресурсы устройств.

Материалы и методы

Для решения задачи псевдореалистичного перемещения неигровых персонажей определимся с самим термином «псевдореалистичный». В рамках настоящей работы под псевдореалистичным поведением понимается такое поведение NPC, которое воспринимается пользователем как правдоподобное и не нарушает его ментальную модель поведения. При этом данное поведение не является результатом имитации реальных процессов. Правдоподобность – это степень, в которой люди считают наблюдаемого и предназначенного для взаимодействия агента автономным [9]. Специалисты в области психологии поведения в данной ситуации допускают возможность использования примитивной логики перемещения, так как мозгу свойственно упрощать информацию об окружающем мире до простых моделей[1] [10]. Согласно известным исследованиям, мозг постоянно работает с допущениями. Это позволяет ему снизить поток единовременно обрабатываемой информации [11].

Учитывая эту особенность восприятия, предлагаемый метод псевдореалистичного перемещения NPC основывается на отслеживании места и времени последней встречи игрока с каждым NPC и использует эту информацию в момент принятия решения о создании NPC.

Перед «появлением» NPC алгоритм проверяет, может ли персонаж находиться в текущей точке игрового мира. Для этого оценивается расстояние и время, прошедшее с момента последней встречи с этим NPC, а также его заданная скорость перемещения. Если расчет показывает, что этот NPC не мог «добраться» до текущей области, он не будет введен в игровую сцену. Предлагаемый подход исключает возможность нереалистичного быстрого «перемещения» неигровых персонажей, что подсознательно повышает степень доверия пользователя к событиям в игре.

Для определения того, какой NPC должен быть создан в текущей области, разработанный метод использует систему приоритетов. Приоритет в данном контексте – переменная для каждого NPC, которая будет увеличиваться, пока игрок с ним не встречается, и уменьшается после встречи с игроком. Таким образом эта переменная контролирует частоту встреч с каждым NPC. Это гарантирует, что встречи в каждой области будут чередовать всех возможных персонажей, что также повышает достоверность игры в глазах игрока.

В Таблице 1 представлены все необходимые переменные, которые необходимо внедрить в NPC для разработки метода. Чтобы реализовать метод псевдореалистичного перемещения NPC с использованием предлагаемого алгоритма разработчики игры должны сначала создать места встреч в игровом мире, между которыми могут перемещаться NPC. Эти места обычно представляют собой такие области, как города, деревни, заведения, лагеря и другие сюжетно важные места игры. Каждая из этих областей – территория, на которой может появиться NPC.

Опишем процесс добавления областей разработчиками, между которыми будут перемещаться неигровые персонажи (Рисунок 1). Для начала на глобальной карте разработчики расставляют области «Встречи» с «путешествующими» NPC. В основном это будут города, деревни, заведения, перекрестки, лагеря и другие ключевые для сюжета игры локации. Под областью будем понимать территорию, в рамках которой будет создаваться неигровой персонаж. Затем, чтобы NPC не возникал прямо перед игроком необходимо на входах в данную область расставить зоны триггеров, чтобы,

[1] Шелл Дж. *Геймдизайн: как создать игру, в которую будут играть все.* Москва: Альпина Паблишер; 2021. 640 с.

после того как игрок через них проходит, вызывалась функция создания NPC. Затем нужно назначить триггерам области «Встречи», к которым они принадлежат.

После того как таким образом будут созданы все области в открытом мире, разработчики должны задать возможные пути перемещения между ними. Для этого они создают лист переменных расстояние между теми областями, между которыми есть прямой путь. Причем следует учесть, что путь в прямом и обратном направлении может отличаться по протяженности. Последовательность составления такого листа показана на Рисунке 1.

Таблица 1 – Переменные сущности «Путешествующий NPC»
Table 1 – Variables of the "Travelling NPC" entity

Название	Описание	Смысловое назначение
ID	Уникальный идентификатор	Отличительное значение каждого персонажа, по которому они и определяются
Скорость	Переменная, использующая в расчетах	Показывает скорость перемещения персонажа по открытому миру
Приоритет	При создании NPC, выбирается тот персонаж, у которого самый высокий положительный приоритет	Показывает с кем наиболее вероятно встретится игрок
Локальный приоритет	Если при выборе NPC приоритет одинаковый, то создается тот, у кого локальный приоритет выше	Второстепенный показатель вероятности встречи с игроком
Последняя посещенная область	ID области. Переменная, использующая в расчетах	Область последней встречи NPC с игроком
Время последней встречи	Игровое время. Переменная, использующая в расчетах	Время последней встречи NPC с игроком
Отсрочка встречи	Значение, которое присваивается Приоритету, после встречи с игроком	Коэффициент того, на сколько часто игрок будет встречаться с персонажем
Приближение встречи	Значение, на которое увеличивается Приоритет, после встречи другого NPC с игроком	Коэффициент того, на сколько часто игрок будет встречаться с персонажем

После того как разработчики определили пути перемещения, запускается функция определения расстояния между точками (Рисунок 2). Это необходимо для создания из листа переменных с расстоянием путей двумерного массива расстояний между каждыми из областей.

Функция работает с использованием цикла, реализующего алгоритм Дейкстры для каждой области игрового мира. Входные данные соответствуют необходимым условием для использования этого алгоритма [5]. Используя этот алгоритм для каждой точки игрового мира, разработчики могут определить расстояние между точками различных локаций.

Полученный двумерный массив хранит расстояния между областями в игровом мире, где номер строки и столбца соответствует уникальному идентификатору каждой области (Рисунок 3). Соответственно на пересечении этих строк и столбцов хранится расстояние между двумя областями.

Запуск алгоритма, который подберет необходимого неигрового персонажа для создания в области, происходит после того, как игрок переместится в зону триггера

(Рисунок 4). Далее идет работа алгоритма с уже отсортированным по приоритету массивом персонажей.

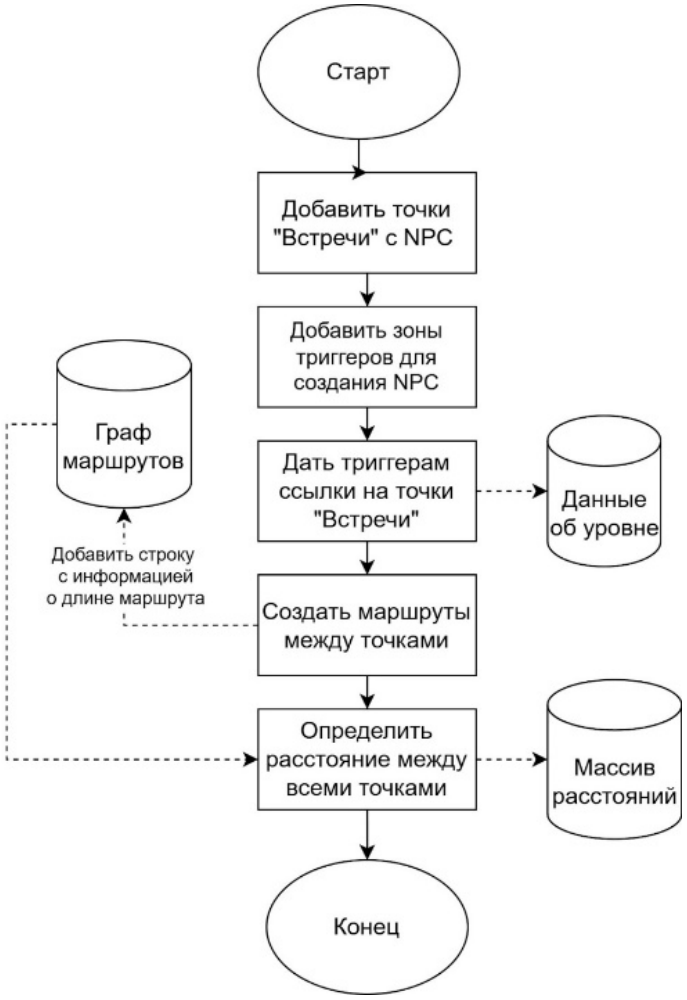
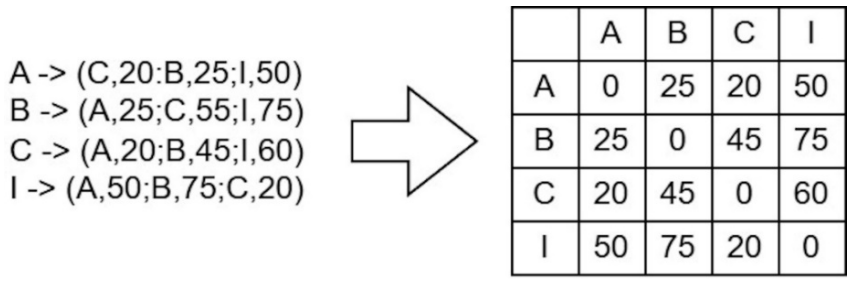


Рисунок 1 – Диаграмма создания графа перемещения NPC
Figure 1 – Diagram of constructing the NPC movement graph

Рисунок 2 – Определение расстояние между областями
Figure 2 – Computing distances between areas



	A	B	C	I
A	0	25	20	50
B	25	0	45	75
C	20	45	0	60
I	50	75	20	0

Рисунок 3 – Преобразование данных
Figure 3 – Data transformation

Рисунок 4 – Алгоритм выбора NPC для создания
Figure 4 – Algorithm for selecting NPCs to spawn

Алгоритм перебирает массив и проверяет для каждого персонажа, является ли приоритет NPC положительным (Рисунок 5). Если он отрицательный, то алгоритм прекращает работу, так как все остальные NPC также будут иметь отрицательный приоритет. Предлагаемый подход позволяет настроить гейм-дизайнерам полное отсутствие встреч с NPC в областях, если того требует игровая ситуация.

Если приоритет NPC положительный, разработанный алгоритм проверяет, имел ли этот персонаж возможность добраться до текущей области, используя предлагаемое выражение (1). Выражение (1) учитывает расстояние между текущей областью и областью последней встречи с данным NPC, а также время, прошедшее с момента последней встречи с этим NPC. Используя это выражение, алгоритм определяет, было ли у NPC достаточно времени, чтобы добраться до текущей области. Если у NPC есть возможность находиться в текущей области, он создается в игровом мире. Если нет,

алгоритм переходит к следующему NPC в массиве и повторяет процесс до тех пор, пока не будет создан NPC или пока не будут проверен весь массив.

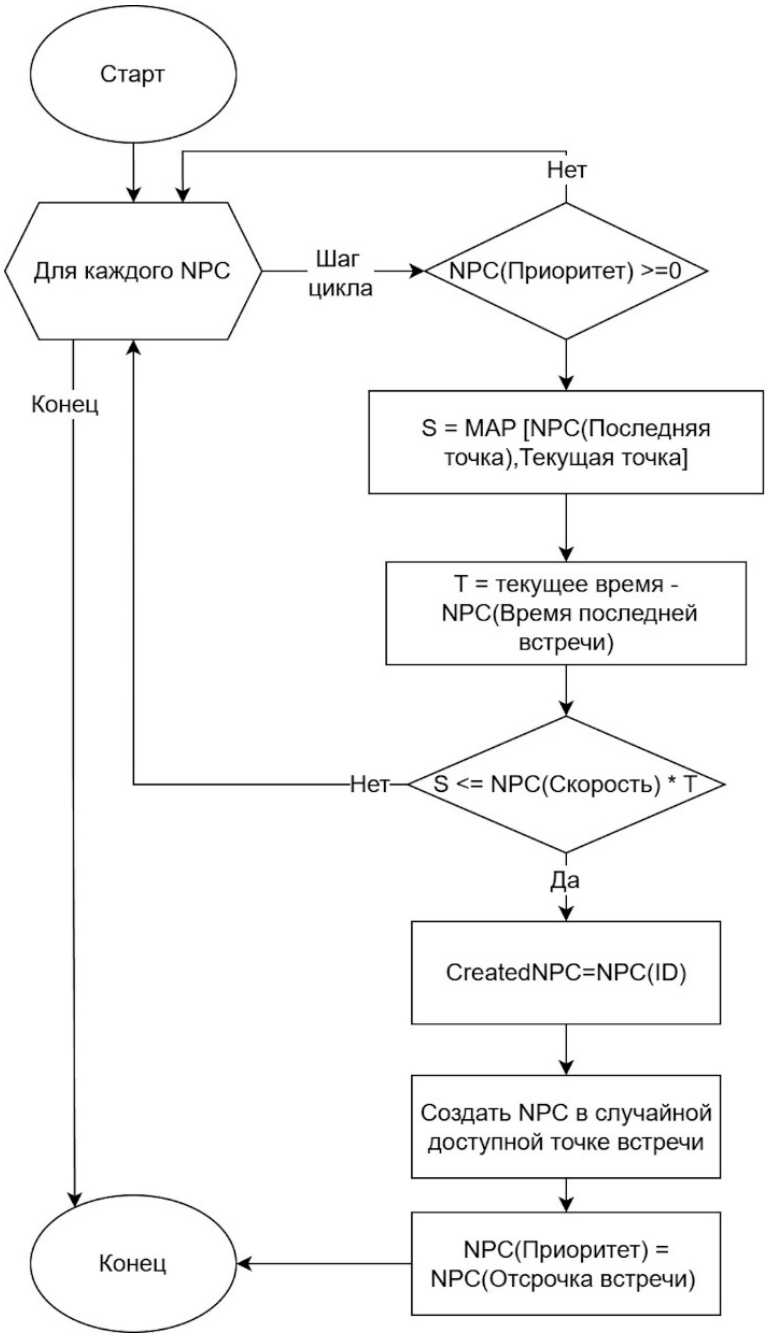


Рисунок 5 – Создание персонажа
Figure 5 – Character spawning

$$S \leq V_{NPC} \cdot \Delta t, \quad (1)$$

где S – расстояние между последний областью, в котором видели персонажа и текущей (2); V_{NPC} – скорость перемещения NPC в виртуальной среде; Δt – время, в течение которого игрок не видел этого персонажа (3).

Для определения расстояние между искомыми областями алгоритм смотрит в ячейку массива, координаты которой равный ID областям. Там хранится искомое расстояние:

$$S = M_{c,l}, \tag{2}$$

где M – двумерный массив расстояний между всеми областями, в которых могут появиться NPC; c – ID текущей области; l – ID локации, в которой персонаж был встречен в последний раз.

Время, прошедшее с последней встречи с неигровым персонажем, рассчитывается простым вычитанием игрового времени последней встречи из текущего игрового времени:

$$\Delta t = t_c - t_{NPC}, \tag{3}$$

где t_c – текущее игровое время; t_{NPC} – время последней встречи с конкретным NPC.

Все выражения можно объединить в одно неравенство:

$$M_{c,l} \leq V_{NPC} \cdot (t_c - t_{NPC}). \tag{4}$$

Если данное выражение истинно, то это означает, что данный неигровой персонаж может добраться до текущей точки игровой карты и тогда в данной области создается этот персонаж, после чего цикл поиска подходящего персонажа завершается. В противном случае цикл переходит к следующему неигровому персонажу.

Каждый раз, когда создается персонаж, его приоритет меняется, отсрочивая следующую встречу:

$$P = P - P_s, \tag{5}$$

где P – текущий приоритет персонажа; P_s – отсрочка встречи.

В завершение алгоритма у каждого не созданного персонажа также меняется приоритет (Рисунок 6), но уже в большую сторону, чтобы приблизить следующую встречу:

$$P = P + P_a, \tag{6}$$

где P_a – приближение встречи.

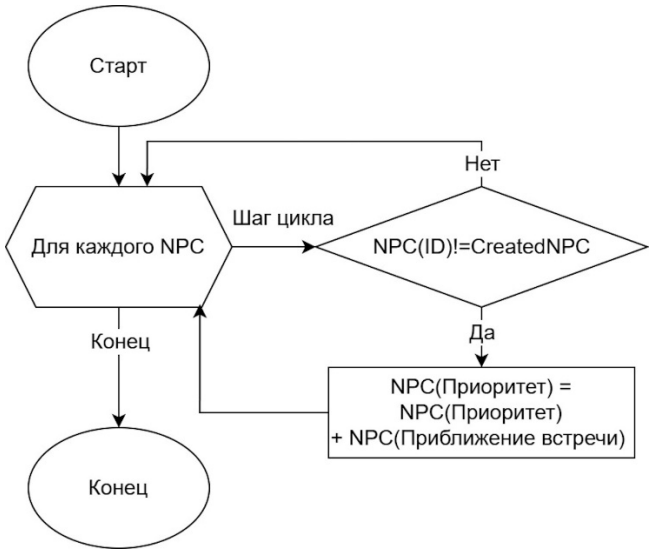


Рисунок 6 – Обновление приоритетов
Figure 6 – Priority updating

После завершения алгоритма массив с персонажами сортируется по приоритету. Таким образом, при следующем запуске цикла массив уже будет готов к работе алгоритма.

Результаты

Для определения оптимального алгоритма сортировки массива персонажей и их приоритета конкретно под данную специфичную задачу, были проведены эксперименты. Вычислительные эксперименты выполнены на сгенерированных данных с использованием программной симуляции, реализующей предложенный метод. Данных подход часто используется для экспериментального сравнения скорости работы алгоритмов [12, 13]. Симулятор формирует связанный взвешенный граф областей, вычисляет матрицу кратчайших расстояний повторными запусками алгоритма Дейкстры и моделирует множество NPC с параметрами из заданных диапазонов. Алгоритм пошагово выполняет выбор NPC согласно правилам приоритета и достижимости; после каждой «встречи» обновляются приоритеты и переупорядочивается список NPC несколькими видами сортировки, подходящими для почти отсортированного списка [13, 14].

1. Шейкерная сортировка.
2. Пирамидальная сортировка.
3. Сортировка вставками.
4. Сортировка выбором.
5. Гибридный алгоритм сортировки timsort [15, 16].

Для оценки масштабируемости варьировались ключевые параметры (Таблица 2). Во время тестирования автоматически собирались метрики времени отдельных этапов, а результаты сохранялись в CSV для последующей обработки и визуализации.

Таблица 2 – Набор параметров для тестирования
Table 2 – Parameter set for testing

Параметр	Значение
Число областей	От 50 до 1000, шаг 50
Число NPC	От 50 до 1000, шаг 50
Средняя степень вершины	От 1 до 10
Кол-во симуляций на одном наборе	200

Результаты среднего времени работы алгоритмов представлены в Таблице 3. Алгоритм «Сортировка выборам» исключен из дальнейшего сравнения, так как его временные значения значительно превышают остальные, что не позволит визуально сравнить остальные алгоритмы. На Рисунке 7 приведено среднее время переупорядочения списка NPC в зависимости от их количества. Результаты показывают, что наименьшее время сортировки у гибридного алгоритма timsort.

Таблица 3 – Среднее время работы алгоритмов сортировки
Table 3 – Average runtime of sorting algorithms

Алгоритм сортировки	Среднее время сортировки (мс)
Шейкерная сортировка	0,191
Пирамидальная сортировка	0,120
Сортировка вставками	0,156
Сортировка выбором	19,205
Гибридный алгоритм сортировки timsort	0,056

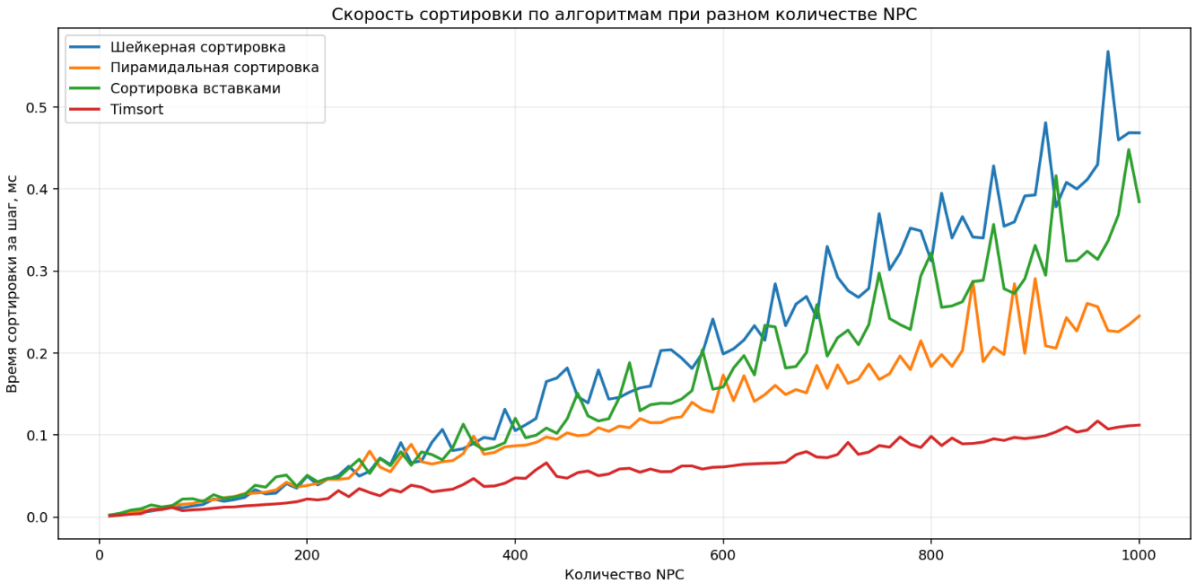


Рисунок 7 – Сравнение времени сортировки массива NPC разных алгоритмов
Figure 7 – Comparison of NPC array sorting time across different algorithms

Были построены графики суммарного времени преобразования данных в зависимости от числа областей (Рисунок 8) и среднего времени выполнения одного шага основной части алгоритма в зависимости от числа NPC (Рисунок 9). Красные кривые на графиках – сглаженные аппроксимации с экстраполяцией до 2000 по оси абсцисс.

Параллельно собирались данные о времени выполнения каждого шага алгоритма. Результат записывался в CSV для последующей обработки. Дополнительно выполнилась аппроксимация зависимостей по всей выборке, что отражено на графиках результатов (Рисунки 8 и 9). Код программной симуляции выложен на GitHub в открытом доступе, чтобы результаты можно было воспроизвести[2].

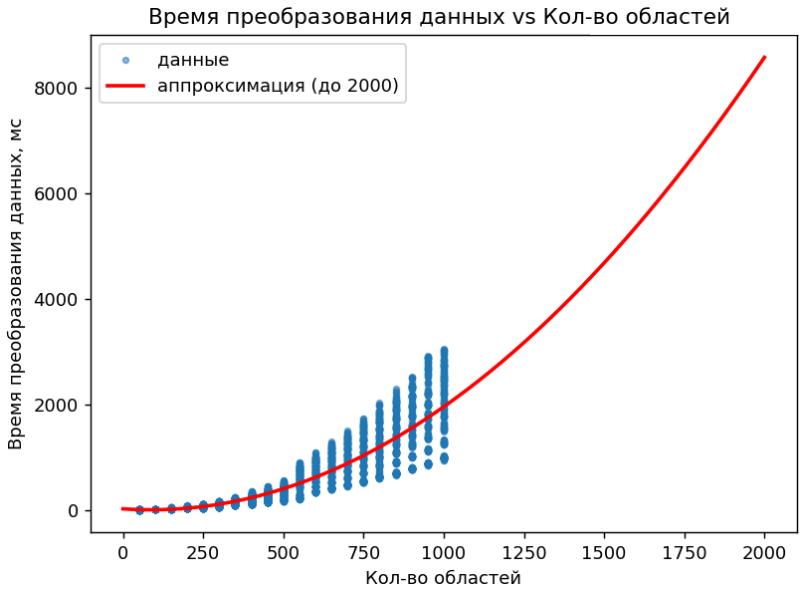


Рисунок 8 – Время преобразования данных в зависимости от количества областей
Figure 8 – Data transformation time vs. number of areas

[2] Kubirill. NPC-Simulation-Prototype. GitHub. URL: https://github.com/Kubirill/NPC-Simulation-Prototype (дата обращения: 28.10.2025).

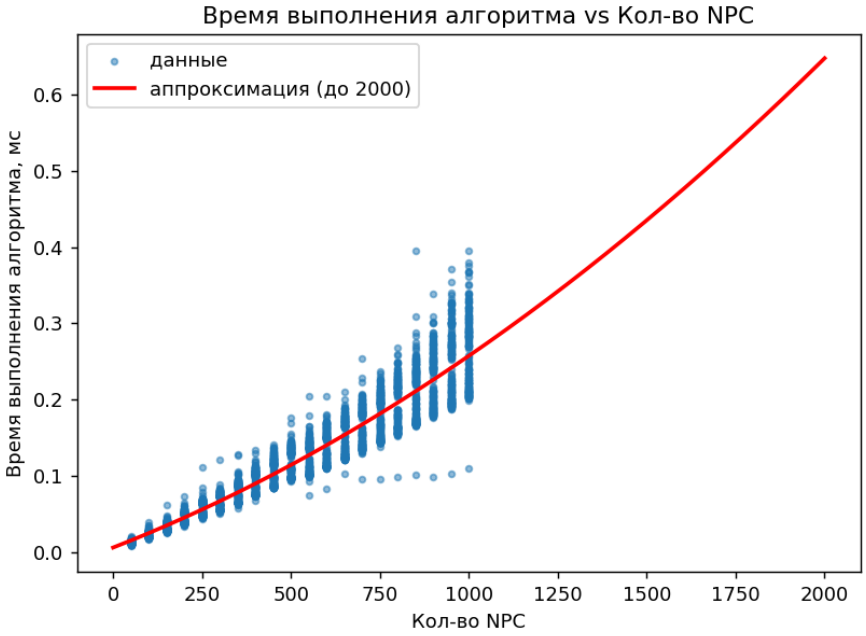


Рисунок 9 – Среднее время шага алгоритма в зависимости от количества NPC
Figure 9 – Average algorithm step time vs. number of NPCs

Обсуждение

Рисунок 8 демонстрирует рост времени построения матрицы расстояний по мере увеличения числа областей. Наблюдаемый квадратичный тренд согласуется с теоретической оценкой повторного запуска алгоритма Дейкстры на разрежённых графах: при постоянной средней степени. Вертикальный разброс точек при фиксированном числе областей отражает влияние плотности (средней степени вершины). Чем больше ребер на вершину, тем дольше один прогон и тем выше суммарное время подготовки. Этот этап – разовая операция на стороне разработчиков. Оценка порядка десятков секунд для ~2000 областей является приемлемой для процесса подготовки данных к разработке.

На Рисунке 9 представлено среднее время одного шага алгоритма выбора NPC в зависимости от их количества, который выполняется при посещении областей. Сортировка TimSort имеет оценку $O(nlogn)$ в худшем случае и адаптивные свойства на частично упорядоченных данных [16]. На практических данных массив NPC, как правило, «почти отсортирован», поэтому фактические значения существенно ниже теоретического предела. Разброс обусловлен параметрами модели и случайной динамикой встреч, влияющими на глубину перестановок. В диапазоне до 1000 NPC среднее время шага остается в пределах миллисекунды, что совместимо с временем обработки кадра современных приложений (например, при 120 FPS – это 8,3 мс).

Основная вычислительная нагрузка сосредоточена в подготовке матрицы расстояний на стороне разработчика, тогда как клиентская часть алгоритма масштабируется умеренно и обеспечивает стабильную работу при типичных размерах миров.

Ограничения метода

Предложенный метод ориентирован на макроуровень и не решает задачи локальной навигации на местности. Предлагается, что данный метод нужно комбинировать с локальным навигационным модулем поиска пути.

Предложенная реализация работает при статичной топологии мира и не поддерживает изменения на карте виртуального мира. Чтобы метод мог учитывать динамичное изменение мира, необходимо изменять граф маршрутов и пересчитывать расстояния между областями.

Подразумевается, что срабатывание алгоритма приводит к созданию только одного NPC. Чтобы нивелировать это ограничение, необходимо запустить алгоритм создания NPC несколько раз (Рисунок 5).

Предложенный метод работает только в однопользовательских проектах. Для многопользовательских требуются отдельные реализации согласования приоритета в зависимости от задач и целей проекта. Это выходит за рамки данной статьи и требует отдельного исследования.

Заключение

Предлагаемый в данной статье метод предлагает оригинальное архитектурное решение для видео игр, которое обеспечивает решение задачи создания псевдореалистичного перемещения NPC в открытых виртуальных мирах без использования сложных алгоритмов. Архитектура ориентирована на то, что обработка графа мира будет предварительно обрабатываться один раз, а метод вызывается только при пересечении триггерных зон. Благодаря этому алгоритм оказывает низкую вычислительную нагрузку, что подтверждается экспериментами.

Использование системы двойного приоритета для каждого NPC гарантируют встречи с персонажами с заданной разработчиком частотой. Это дает разработчикам инструмент контроля, который позволяет регулировать частоту встреч с разными NPC, не нарушая ожиданий игрока.

По итогу в основе метода лежит простая математика, основанная на ожиданиях игрока. Благодаря этому у игроков создается иллюзия реалистичности перемещения NPC.

Реализация предложенного метода улучшит игровой опыт и обеспечит предсказуемый игровой процесс для гейм-дизайнеров, что является приоритетной целью разработчиков игр.

Результаты экспериментов показывают, что метод способен обрабатывать до 1000 NPC, не оказывая заметной нагрузки на систему. Это позволяет не только использовать метод в текущих проектах, но и оставляет запас для игрового масштабирования в будущем. Также данный метод может быть использован не только в рамках игровых проектов, но также в обучающих тренажерах и симуляторах виртуальной реальности.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Uludağlı M.Ç., Oğuz K. Non-player character decision-making in computer games. *Artificial Intelligence Review.* 2023;56(12):14159–14191. https://doi.org/10.1007/s10462-023-10491-7
2. Da Silva G.A., de Souza Ribeiro M.W. Development of Non-Player Character with Believable Behavior: a systematic literature review. In: *2021: Companion Proceedings of the 20th Brazilian Symposium on Games and Digital Entertainment, 18–21 October 2021, Gramado, Brazil.* Porto Alegre: Sociedade Brasileira de Computação; 2021. P. 319–323. https://doi.org/10.5753/sbgames_estendido.2021.19660
3. Yang D., Kleinman E., Harteveld C. *GPT for Games: An Updated Scoping Review (2020–2024).* arXiv. URL: https://arxiv.org/abs/2411.00308 [Accessed 5th October 2025].

4. Yakan S.A. Analysis of Development of Artificial Intelligence in the Game Industry. *International Journal of Cyber and IT Service Management.* 2022;2(2):111–116. https://doi.org/10.34306/ijcitsm.v2i2.100
5. Cormen Th.H., Leiserson Ch.E., Rivest R.L., Stein C. *Introduction to Algorithms.* Cambridge: MIT Press; 2022. 1312 p.
6. Шутов К.И., Акатьев Я.А., Лобанов А.А. Анализ особенностей поведения неигровых персонажей в виртуальных мирах. *E-Scio.* 2023;(3):140–154.
7. Guimarães M., Santos P.A., Jhala A. *Emergent Social NPC Interactions in the Social NPCs Skyrim Mod and Beyond.* arXiv. URL: https://doi.org/10.48550/arXiv.2207.13398 [Accessed 3rd October 2025].
8. Park J.S., O'Brien J., Cai C.J., et al. Generative Agents: Interactive Simulacra of Human Behavior. In: *UIST '23: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, 29 October – 01 November 2023, San Francisco, CA, USA.* New York: ACM; 2023. https://doi.org/10.1145/3586183.3606763
9. Guo S., Adamo N., Mousas Ch. Developing a Scale for Measuring the Believability of Virtual Agents. In: *ICAT-EGVE 2023: 33rd International Conference on Artificial Reality and Telexistence, 28th Eurographics Symposium on Virtual Environments, 06–08 December 2023, Trinity College Dublin, Ireland.* Eurographics Association; 2023. P. 45–52. https://doi.org/10.2312/egve.20231312
10. Sprevak M., Smith R. An Introduction to Predictive Processing Models of Perception and Decision-Making. *Topics in Cognitive Science.* 2023. https://doi.org/10.1111/tops.12704
11. Clark A. *The Experience Machine: How Our Minds Predict and Shape Reality.* New York: Pantheon Books; 2023. 320 p.
12. Bai X., Coester Ch. Sorting with Predictions. In: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 10–16 December 2023, New Orleans, LA, USA.* 2023. URL: https://papers.neurips.cc/paper_files/paper/2023/file/544696ef4847c903376ed6ec58f3a703-Paper-Conference.pdf
13. Durrani O.Kh., AbdulHayan S. Performance Measurement of Popular Sorting Algorithms Implemented using Java and Python. In: *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), 16–18 November 2022, Maldives.* IEEE; 2022. P. 1–6. https://doi.org/10.1109/ICECCME55909.2022.9988424
14. Michail D. JHeaps: An open-source library of priority queues. *SoftwareX.* 2021;16. https://doi.org/10.1016/j.softx.2021.100869
15. Mubarak A., Iqbal S., Naeem T., Hussain Sh. 2 mm: A new technique for sorting data. *Theoretical Computer Science.* 2022;910:68–90. https://doi.org/10.1016/j.tcs.2022.01.037
16. Auger N., Jugé V., Nicaud C., Pivoteau C. On the Worst-Case Complexity of TimSort. In: *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018), 20–22 August 2018, Helsinki, Finland.* Schloss Dagstuhl – Leibniz-Zentrum für Informatik; 2018. https://doi.org/10.4230/LIPIcs.ESA.2018.4

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Шутов Кирилл Игоревич, ассистент кафедры игровой индустрии, МИРЭА – Российский технологический университет, Москва, Российская Федерация.
e-mail: kubirill@rambler.ru

Kirill I. Shutov, Assistant at the Department of Gaming Industry, MIREA – Russian Technological University, Moscow, the Russian Federation.

Лобанов Александр Анатольевич, кандидат технических наук, доцент, доцент кафедры инструментального и прикладного программного обеспечения, МИРЭА – Российский технологический университет», Москва, Российская Федерация.
e-mail: a.a.lobanoff@ya.ru

Aleksandr A. Lobanov, Candidate of Engineering Sciences, Docent, Associate Professor at the Department of Instrumental and Applied Software, MIREA – Russian Technological University, Moscow, the Russian Federation.

Статья поступила в редакцию 29.01.2026; одобрена после рецензирования 06.03.2026; принята к публикации 16.03.2026.

The article was submitted 29.01.2026; approved after reviewing 06.03.2026; accepted for publication 16.03.2026.