Дата поступления рукописи в редакцию: 29.06.2025 г.
Дата принятия рукописи в печать: 05.07.2025 г.
DOI: 10.24412/3034-4042-2025-3-80-86

БОРОВИКОВА Анастасия Валентиновна,
Магистрант,
Российская академия народного хозяйства и государственной службы при Президенте Российской Федерации (РАНХиГС),
e-mail: Borovikova.an.v@gmail.com

ГИЛЕВСКИЙ Александр Сергеевич,
Магистрант,
Российская академия народного хозяйства и государственной службы при Президенте Российской Федерации (РАНХиГС),
e-mail: alex.gilevskiy@yandex.ru

ОВЧИННИКОВА Ксения Романовна,
кандидат экономических наук,
Российская академия народного хозяйства и государственной службы при Президенте Российской Федерации (РАНХиГС),
e-mail: Borovikova.an.v@gmail.com

МЕТОДЫ И АЛГОРИТМЫ, ИСПОЛЬЗУЕМЫЕ ДЛЯ ПРИНЯТИЯ РЕШЕНИЙ ИГРОВЫМ ИСКУССТВЕННЫМ ИНТЕЛЛЕКТОМ В КОМПЬЮТЕРНЫХ ИГРАХ

Аннотация. *Статья посвящена обзору методов и алгоритмов, используемых для принятия решений игровым искусственным интеллектом (ИИ) в компьютерных играх. Описаны принципы работы конечных автоматов, деревьев поведения и систем на основе полезности. Рассмотрены их преимущества и недостатки, а также приведены примеры реализации данных методов в различных игровых сценариях.*

Ключевые слова: *искусственный интеллект, разработка компьютерных игр, принятие решений, неигровые персонажи, конечные автоматы, деревья поведения, системы на основе полезности.*

BOROVIKOVA Anastasia Valentinovna,
Master's Student The Russian Presidential Academy of National Economy and Public Administration (RANEPA) Россия, г. Москва

GILEVSKIY Aleksandr Sergeevich,
Master's Student The Russian Presidential Academy of National Economy and Public Administration (RANEPA)

OVCHINNIKOVA Kseniya Romanovna,
Associate Professor The Russian Presidential Academy of National Economy and Public Administration (RANEPA)

METHODS AND ALGORITHMS USED FOR DECISION-MAKING BY GAME ARTIFICIAL INTELLIGENCE IN COMPUTER GAMES

Annotation. *The article provides an overview of the methods and algorithms used for decision-making by game artificial intelligence (AI) in video games. It describes the principles of finite state machines, behavior trees, and utility-based systems. The advantages and disadvantages of these approaches are analyzed, and examples of their implementation in various game scenarios are presented.*

Key words: *artificial intelligence, game development, decision-making, non-player characters, finite state machines, behavior trees, utility-based systems.*

Введение

Разработчики компьютерных игр часто сталкиваются с задачами реализации поведения неигровых персонажей, генерации уникального игрового контента (уровней, событий, сценариев), адаптации сложности под уровень игрока, а также создания других игровых механик, направленных на повышение реалистичности и динамичности игрового процесса и, как следствие, вовлеченности игроков [1].

Для решения подобных задач в игровой индустрии сформировалось отдельное направление – игровой искусственный интеллект (ИИ), объединяющее алгоритмы из теории управления, робототехники, компьютерной графики и информатики [2]. В отличие от классического ИИ, основанного на нейронных сетях, при реализации игрового ИИ чаще всего применяются детерминированные подходы, позволяющие создавать иллюзию поведения, основанного на заданных разработчиком правилах.

В настоящее время наиболее широко распространены три подхода к построению игрового ИИ: конечные автоматы, деревья поведения и системы на основе полезности [3]. Каждый из них представляет собой уникальную модель принятия решений, отличающуюся по сложности реализации, степени гибкости, адаптируемости к игровому процессу и требованиям к производительности.

Данная статья посвящена обзору методов реализации игрового ИИ, рассмотрению характерных достоинств и недостатков каждого подхода, а также описанию примеров их применения в различных игровых сценариях.

Конечные автоматы

Конечные автоматы (Finite State Machines, FSM) представляют собой модель, которая позволяет определить состояние объекта в конкретный момент времени и описать его возможные переходы между этими состояниями. В данной модели количество состояний ограничено, что делает ее особенно удобной для анализа поведения различных объектов в игровых сценариях.

Конечные автоматы состоят из следующих компонентов:

1. Состояния – описывают текущее поведение объекта. Например, для врага в игре это может быть «патрулирование», «преследование», «атака», «отступление».
2. Переходы – логические условия, при выполнении которых происходит смена состояний. Переход может быть вызван событием (например, замечен игрок) или изменением параметров (например, здоровье ниже 50%).
3. Действия – операции, которые необходимо выполнить в каждом состоянии, например, находясь в состоянии «патрулирование» неигровой персонаж перемещается случайным образом и ищет игрока.

Принцип работы FSM заключается в том, что в каждый момент времени агент находится только в одном активном состоянии и реагирует на события или условия в соответствии с текущим состоянием. При срабатывании условия перехода происходит переключение на новое состояние, где начинается выполнение соответствующих действий [4].

Далее представлен пример конечного автомата (*рис. 1*).

Рисунок 1. Конечный автомат

Данный FSM описывает поведение боевой турели, которое начинается со сканирования области, переходит в прицеливание при обнаружении противника, а затем в атаку, если цель в зоне поражения. Если патроны заканчиваются, турель переходит в перезарядку и возобновляет работу после заполнения магазина. Если противник скрылся из виду или был уничтожен турель возвращается в режим сканирования.

Преимущества конечных автоматов:

– простота реализации и визуализации. Логика поведения легко представляется в виде состояний и переходов между ними. Это удобно как для программистов, так и для гейм-дизайнеров;
– предсказуемость поведения. При заданных условиях результат всегда одинаков – это облегчает отладку и настройку баланса;
– низкие вычислительные затраты. FSM не требует сложных вычислений и хорошо подходит для игр с ограничениями по производительности.
– Недостатки конечных автоматов:
– проблемы масштабирования. При увеличении количества состояний (N), сильно увеличивается количество переходов (NxN) и система быстро усложняется [5];
– ограниченная адаптивность. FSM плохо подходит для сценариев с высокой степенью вариативности.

Когда использовать конечные автоматы?

– для простых неигровых персонажей и объектов с линейным поведением (лифт, турели, патрульные);
– в прототипах для быстрой проверки игровой механики;
– для мобильных и веб-игр, где важна производительность и экономия памяти;
– в гибридных системах, например, где FSM используется для определения высокоуровневого состояния (мирный режим, боевой режим, отступление), а дерево поведения для реализации конкретных действий в этом состоянии.

Деревья поведения

Дерево поведений (Behavior Tree, BT) представляет собой ориентированный ациклический граф с множеством составных узлов, где каждый узел соединен друг с другом и имеет фиксированное направление, которое не зацикливается и не повторяется. На текущий момент данный метод является одним из популярных подходов для реализации игрового ИИ. В частности, в Unreal Engine от Epic и в Unity 6 инструменты для создания деревьев поведения встроены в игровой движок.

Обход дерева начинается с корневого узла и выполняется сверху вниз, слева направо. Каждый узел, в зависимости от типа, возвращает один из трёх статусов: успех (success), сбой (failure) или выполняется (running). Обычно дерево проверяется каждый игровой кадр, но реализация может предусматривать возврат к последнему выполняемому узлу [6].

Для хранения и обмена информацией между узлами дерева часто используется структура данных под названием черная доска (blackboard). Черная доска в общем случае является системой памяти агента, к которой могут получить доступ не только системы поведения типа дерева поведения, но и любые системы ИИ [7].

Существует несколько типов узлов:

1. Корневой узел (Root node) – является точкой входа для графа, у него нет родителей, но всегда есть дочерние узлы, именно с него начинается выполнение.
2. Конечные узлы (Leaf nodes) – содержат фактические команды, которые управляют поведением ИИ. Данные узлы не имеют дочерних узлов. Могут возвращать три статуса: успех, неудача, выполняется (если действие имеет продолжительный характер). Конечные узлы обычно бывают двух типов:

– действия – выполняют команды, соответствующие моделируемому поведению на самом низком уровне (бег к позиции, стрельба по врагу или взаимодействие с объектом);
– условия – используется для проверки условий (например, «Враг в зоне видимости?», «Здоровье <50%»).

3. Композитные узлы (Composite) – служебные узлы, которые управляют потоком перемещения по дереву и определяют переход от одного конечного узла к другому. Существует несколько типов композитных узлов:

– последовательность (sequence) – выполняет дочерние узлы по порядку до первого сбоя;
– переключатель (selector) – выполняет дочерние узлы по порядку до первого успеха;
– параллельный узел (parallel) – запускает несколько узлов одновременно; результат зависит от заданных условий успеха;
– случайная последовательность/селектор (random sequence / selector) – аналогично обычным, но порядок обхода выбирается случайным образом.

4. Узлы-декораторы (Decorator nodes) – модифицируют поведение одного дочернего узла. Наиболее распространённые:

– инвертор (inverter) – меняет успех на сбой и наоборот;
– успех (succeeder) – всегда возвращает успех;
– повторитель (repeater) – повторяет выполнение дочернего узла;
– повторять до успеха / до сбоя (repeat until successful / fail) – повторяет выполнение до заданного результата.

Далее представлен пример дерева поведения (*рис. 2*).

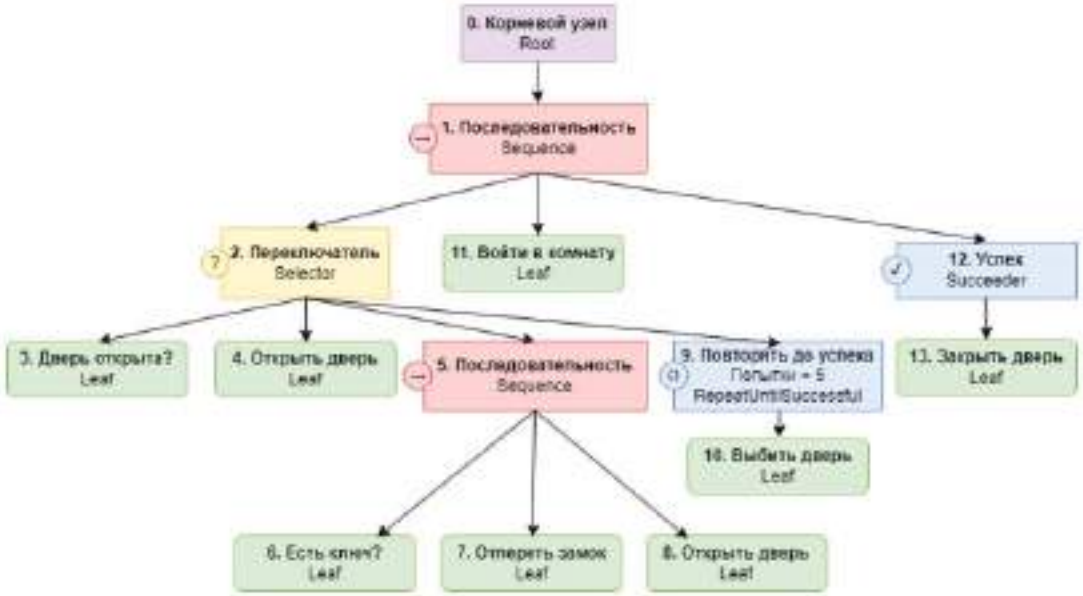

Рисунок 2. Дерево поведения

Данное дерево поведения будет использоваться при необходимости неигровому персонажу (NPC) пройти в комнату через дверь. Поведение начинается с узла «1. Последовательность», который определяет порядок действий NPC. Первый дочерний узел «2. Переключатель» – проверяет различные способы открытия двери, последовательно обходя дочерние узлы до первого успешного результата. Если же дверь изначально открыта или NPC удастся открыть её одним из доступных способов, то узел «2. Переключатель» вернет статус «успех» узлу «1. Последовательность», после чего выполняются действия «11. Войти в комнату» и «13. Закрыть дверь». Декоратор «12. Успех» добавлен специально, чтобы избежать статуса «сбой» при выполнении узла «13. Закрыть дверь», в случае, когда дверь была выбита.

Преимущества деревьев поведения:

– гибкость и масштабируемость. Деревья можно легко реорганизовывать, добавлять новые ветки поведения без переписывания всей логики;
– поддержка разнообразных стратегий поведения. Через композитные узлы можно реализовать широкий спектр логики;
– повторное использование компонентов. Отдельные узлы (действия, условия) можно использовать в разных частях дерева и даже для разных неигровых персонажей.

Недостатки деревьев поведения:

– сложность проектирования. Большие деревья поведения могут становиться трудно читаемыми и поддерживаемыми без хорошей структуры и комментариев;
– проблемы производительности. Рекурсивный обход дерева требует больше ресурсов, чем конечные автоматы.
– Когда использовать деревья поведения?
– для сложных неигровых персонажей, использующих тактику или различные режимы боя (союзники или враги в тактических шутерах, боссы с фазами боя);
– в проектах с частыми изменениями логики поведения. Иерархическая структура деревьев поведения упрощает модификацию отдельных частей поведения без необходимости переписывать всю систему ИИ.

Системы на основе полезности

Системы на основе полезности (Utility-Based Systems) представляют собой еще один подход к моделированию поведения агентов в компьютерных играх [8]. В отличие от конечных автоматов, деревьев поведений и других подходов, которые основываются на заранее определенных правилах и структурах, системы на основе полезности моделируют поведение агентов на основе динамического вычисления полезности различных действий в зависимости от текущего состояния игры. Это позволяет агентам принимать более адаптивные решения, приближенные к человеческой логике.

Общая схема процесса принятия решений на основе полезности состоит из следующих этапов:

1. Сбор данных. На этом этапе определяется список возможных действий, а также набор параметров (состояние здоровье, расстояние до цели, наличие ресурсов и т.д.), влияющих на оценку полезности каждого из них.
2. Оценка полезности действий. Для каждого действия вычисляется значение полезности на основе текущих параметров. Входные значения предварительно нормализуются в диапазоне от 0 до 1. Расчет полезности осуществляется с помощью совокупности простых математических функций, описывающих влияние каждого параметра на итоговую оценку. Во многих игровых движках (например, Unreal Engine и Unity) преобразование параметров реализуется через настраиваемые кривые полезности. Это даёт разработчикам возможность гибко адаптировать поведение агентов, изменяя форму кривых в визуальном редакторе без необходимости переписывания кода.
3. Выбор действия. На основе вычисленных значений полезности выбирается действие с наибольшим значением. Это обеспечивает выполнение агентом наиболее выгодного действия в текущих обстоятельствах.

Для примера рассмотрен выбор действий из категории «Восстановить здоровье», данная категория будет использоваться, когда количество очков здоровья неигрового персонажа меньше 50%. Ниже описаны доступные действия:

1. Выпить зелье здоровья. Действие мгновенно восстанавливает все здоровье, но тратит зелье здоровья.
2. Применить заклинание восстановления здоровья. Действие восстанавливает все здоровье в течение 3 секунд, тратит 10% маны и накладывает эффект перезарядки заклинания на 10 секунд.
3. Бежать в безопасную зону. При нахождении в данной зоне здоровье восстанавливается автоматически.

Затем были определены основные параметры, влияющие на расчет полезности этих действий, и вычислены нормализованные значения полезностей (*табл. 1*).

Таблица 1. Параметры и расчет полезности

Действие	Параметр	Текущее значение	Влияние на полезность	Полезность
Выпить зелье	Текущее здоровье	30%	Чем меньше здоровья, тем выше значение.	0.7
	Наличие зелья	1 зелье	1 - если есть 0 - если нет	1
	Расстояние до врага	5 метров	Чем ближе враг, тем выше значение	0.9
	Расстояние до безопасной зоны	10 метров	Чем дальше зона, тем выше значение	0.1
Применить заклинание	Мана	50%	Чем больше маны, тем выше значение 0 - если маны меньше 10%	0.6
	Доступность заклинания	Доступно	1 - если доступно 0 - если в откате	1
	Расстояние до врага	5 метров	Чем ближе враг, тем ниже значение	0.1
	Расстояние до безопасной зоны	10 метров	Чем дальше зона, тем выше значение	0.1
Бежать в безопасную зону	Текущее здоровье	30%	Чем меньше здоровья, тем ниже значение	0.3
	Расстояние до зоны	10 метров	Чем ближе зона, тем выше значение	0.9
	Расстояние до врага	5 метров	Чем ближе враг, тем ниже значение	0.1

Для расчета общей полезности каждого действия в данном примере использовалось среднее геометрическое, чтобы избежать зависимости полезности от количества параметров.

$$U = \left(\prod_{i=1}^{n} u_i\right)^{\frac{1}{n}}$$

где:

- U – итоговое значение полезности действия;
- ui∈[0,1] – нормализованные значения полезности каждого параметра;
- n – количество параметров

Далее показана итоговая полезность и пояснение логики выбора поведения (*табл. 2*).

Таблица 2. Логика выбора поведения

Действие	Полезность	Логика выбора
Выпить зелье	0.5	**Оптимальный выбор** - сочетание низкого здоровья, наличия зелья и близости угрозы делает это действие наиболее рациональным.
Бежать в безопасную зону	0.30	**Резервный вариант** - полезность снижена из-за уже критического здоровья и близости врага. Будет выбрано, если зелье недоступно.
Применить заклинание	0.28	**Наименее приоритетно** - несмотря на достаточный запас маны, близость врага делает заклинание слишком рискованным.

Преимущества системы на основе полезности:

- высокая вариативность. При одинаковом наборе действий персонаж ведёт себя по-разному и принимает решения в зависимости от текущей ситуации, а не по жестко заданным правилам;
- хорошая масштабируемость. Новые параметры и действия можно добавлять без необходимости перестраивать всю систему.
- Недостатки системы на основе полезности:
- сложность настройки функций полезности. Каждое поведение требует определения числовой функции, оценивающей его «полезность» в текущем контексте. Подбор весов и коэффициентов часто осуществляется вручную и требует множества итераций;
- трудности с интерпретацией и отладкой. В отличие от конечных автоматов или деревьев поведения, поведение ИИ может быть менее предсказуемым, особенно при большом числе факторов. Это усложняет анализ принятых решений и выявление ошибок в логике;
- увеличенная нагрузка на производительность. В особенно сложных случаях, когда поведение зависит от десятков факторов, расчет функций полезности в реальном времени может стать ресурсоемким процессом, особенно при большом числе NPC.

Когда использовать систему на основе полезности?

- для сложных неигровых персонажей, демонстрирующих адаптивное поведение (например, персонажи в ролевых играх и симуляторах), где необходимо учитывать несколько факторов одновременно при выборе действий;
- в проектах с высокими требованиями к вариативности решений (например, в стратегических играх), где система должна выбирать оптимальные действия на основе анализа множества факторов.

Заключение

В данной статье были рассмотрены три наиболее распространенных подхода к реализации игрового ИИ: конечные автоматы, деревья поведения и системы на основе полезности. Каждый из методов обладает характерными особенностями, преимуществами и недостатками, которые определяют область их применения в игровых проектах различного масштаба и сложности.

Конечные автоматы выделяются простотой, предсказуемостью и высокой производительностью, что делает их подходящими для реализации простого и строго определённого поведения. Деревья поведения предоставляют более гибкий и иерархически организованный способ описания логики, хорошо масштабируются и широко поддерживаются современными игровыми движками. Системы на основе полезности обеспечивают наибольшую адаптивность и вариативность, позволяя агентам принимать решения в зависимости от игрового контекста.

Выбор подходящего метода зависит от множества факторов, таких как жанр и тип игры, сложность поведения, технические ограничения и требования к адаптивности. На практике часто используются гибридные архитектуры, сочетающие в себе сильные стороны различных подходов.

Список литературы:

[1] Warpefelt H. The Non-Player Character: Exploring the believability of NPC presentation and behavior: PhD thesis. – Stockholm: Stockholm University, 2016. – 118 p.

[2] Лебедев В. Не совсем человек: искусственный интеллект в играх // Skillbox. URL: https://skillbox.ru/media/gamedev/iskusstvennyy-intellekt-v-igrakh/ (дата обращения: 06.04.2025).

[3] Lapeyrade S. Reasoning with Ontologies for Non-player Character's Decision-Making in Games / S. Lapeyrade // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2022. – Vol. 18(1) – 303-306 p.

[4] Artificial Intelligence. Lecture 03 – Finite State Machines // Edirlei Soares de Lima - Research Website. URL: https://edirlei.com/aulas/game-ai-2022/GAME_AI_Lecture_03_Finite_State_Machines_2022.html (дата обращения: 05.04.2025).

[5] Буковшин, В. А. Интеллектуальные системы в компьютерных играх. Перспективы развития искусственного интеллекта в игровой индустрии / В. А. Буковшин, С. Г. Воскобойников // Современные материалы, техника и технологии. – 2017. – № 3 (11). – С. 21-36.

[6] Simpson C. Behavior trees for AI: How they work // Game Developer. URL: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (дата обращения: 06.04.2025).

[7] Охинченко И. Игровой ИИ. Behavior Tree // Teletype. URL: https://teletype.in/@jazzyjohn/g_8-tnAlkDe

[8] Utility Intelligence // CarlosLab. URL: https://carloslab-ai.github.io/UtilityIntelligence/ (дата обращения: 02.06.2025).

Spisok literatury:

[1] Warpefelt H. The Non-Player Character: Exploring the believability of NPC presentation and behavior: PhD thesis. – Stockholm: Stockholm University, 2016. – 118 p.

[2] Lebedev V. Ne sovsem chelovek: iskusstvennyi intellekt v igrakh // Skillbox. URL: https://skillbox.ru/media/gamedev/iskusstvennyy-intellekt-v-igrakh/ (data obrashcheniia: 06.04.2025).

[3] Lapeyrade S. Reasoning with Ontologies for Non-player Character's Decision-Making in Games / S. Lapeyrade // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2022. – Vol. 18(1) – 303-306 p.

[4] Artificial Intelligence. Lecture 03 – Finite State Machines // Edirlei Soares de Lima - Research Website. URL: https://edirlei.com/aulas/game-ai-2022/GAME_AI_Lecture_03_Finite_State_Machines_2022.html (data obrashcheniia: 05.04.2025).

[5] Bukovshin, V. A. Intellektual'nye sistemy v komp'iuternykh igrakh. Perspektivy razvitiia iskusstvennogo intellekta v igrovoi industrii / V. A. Bukovshin, S. G. Voskoboinikov // Sovremennye materialy, tekhnika i tekhnologii. – 2017. – № 3 (11). – S. 21-36.

[6] Simpson C. Behavior trees for AI: How they work // Game Developer. URL: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (data obrashcheniia: 06.04.2025).

[7] Okhinchenko I. Igrovoi II. Behavior Tree // Teletype. URL: https://teletype.in/@jazzyjohn/g_8-tnAlkDe

[8] Utility Intelligence // CarlosLab. URL: https://carloslab-ai.github.io/UtilityIntelligence/ (data obrashcheniia: 02.06.2025).