# Исследование методов моделирования принятия решений на примере ботов в компьютерной игре

# Введение

Принятие решений – это выбор одной из альтернатив. Оно может быть автоматизировано с помощью компьютерных программ. Интеллектуальные агенты, принимающие решения, опираются на различные математические модели и алгоритмы такие как алгоритмы поиска оптимального пути, нечеткая логика, конечные автоматы, теория вероятности, нейронные сети, генетические алгоритмы и другие.

Системы искусственно интеллекта разделяют на два вида: сильный ИИ и слабый ИИ. Система, обладающая сильным ИИ может мыслить, осознавать себя и реальность, обучаться и принимать решения, основанные на собственном понимании задачи. Системы со слабыми ИИ – это компьютерные программы, созданные для решения конкретной заранее известной задачи. Такие системы окружают нас повсюду: рекомендательные сервисы, системы распознавания голоса, изображений и видео, экспертные системы и базы знаний, роботы, системы «умный дом» и другие.

В данной работе производится обзор различных методов и алгоритмов принятия решений, анализируются их сильные и слабые стороны. На примере разработки бота для компьютерной игры показано, как рассмотренные алгоритмы могут быть применены для создания системы, принимающей решения. В общем случае бот – это компьютерная программа, выполняющая какие-либо действия через интерфейсы, предназначенные для людей. В частном случае компьютерных игр бот – это программа, имитирующая поведение живых игроков в многопользовательских играх. Это агент, который анализирует окружающую обстановку, принимает решения и выполняет действия, которые дают ему игровые преимущества и приводят к выигрышу в соответствии с заданными правилами игры.

Разработка ботов для видеоигр имеет ряд нюансов. Так как основное назначение игр – это развлечение игроков, нет смысла делать компьютерных игроков, которые в любых условиях играют лучше игроков живых, так как в этом случае игра быстро потеряет интерес. Кроме того, все решения бот должен производить в реальном времени, быстро реагируя на действия других игроков. Оба эти фактора приводят к тому, что зачастую при разработке ботов на первый план выходит способность имитировать реальных игроков и вычислительная эффективность алгоритмов, а не максимизация качества самих решений.

# Постановка задачи и обоснование актуальности

## Принятие решений

Принятие решений – процесс построения множества альтернативных возможностей и выбор одного или нескольких элементов из этого множества в соответствии с определенной системой ценностей и предпочтений. Этот процесс может быть автоматизирован с помощью вычислительной техники. Вычислительная система, принимающая решение и действующая в соответствии с ним, называется агентом. Если при этом агент стремится достичь наилучшего результата, такой агент называется рациональным [Рассел и Норвиг].



Рисунок 1. Модель агента.

На рисунке 1 представлена модель агента. Агент получает информацию о внешней среде и обновляет свое внутреннее состояние. Затем на основании имеющейся информации и определенной системы правил он производит выбор решения и выполняет связанное с ним воздействие на среду.

В простейшем случае внутреннее состояние может отсутствовать. Такой агент называется простым рефлексным агентом. В этом случае действия, производимые агентом, являются простыми реакциями на внешнюю среду и не зависят от прошлого состояния среды и прошлых действий агента. Такая стратегия оправдана, если датчики всегда предоставляют информацию, которой достаточно для принятия решения.

В случае частичной наблюдаемости среды агент должен поддерживать внутреннее состояние, которое зависит от истории наблюдений и отражает некоторые из ненаблюдаемых аспектов текущего состояния среды. Для корректного обновления внутреннего состояния нужно обладать дополнительной информацией о том, как среда изменяется независимо от агента, и о том, как те или иные действия агента изменяют среду. Эта информация, заложенная в агенте, называется моделью мира, а такой агент – рефлексным агентом, основанным на модели.

Процесс принятия решений можно представить в виде отображения [Шампандар]:

,

где C – контекст, множество возможных состояний мира, S – множество стратегий принятия решений, D – множество существующих решений. Оно показывает, что в соответствии с выбранной стратегией из множества S, реакцией агента на окружение является определенное действие из множества D.

Возможно альтернативное отображение, моделирующее принятие решений:

,

где R – множество вещественных чисел. Каждой комбинации контекста, стратегии и возможного решения ставится в соответствие значение пригодности решения для представленных условий. Оптимальным решением при этом является то, которое при равных состояниях мира и выбранных стратегиях имеет максимальную пригодность.

## Сильный и слабый искусственный интеллект

Философы и инженеры разделяют два подхода к определению искусственного интеллекта: сильный ИИ и слабый ИИ. Впервые такое разделение ввел Джон Сирл в [Сирл]. Сильный искусственный интеллект имеет черты, свойственные человеческому. Система, обладающая сильным ИИ может мыслить, осознавать себя и реальность, обучаться и принимать решения, основанные на собственном понимании задачи. В настоящее время таких систем не существует, и ведутся споры о возможности их существования.

Слабый искусственный интеллект – это программно-аппаратная платформа, разработанная для решения конкретной заранее известной задачи. Такие системы широко используются в наше время: распознавание изображений, видео и голоса, экспертные системы и базы знаний, системы «умный дом», промышленные контроллеры, системы управления роботами и другие системы, основная задача которых – принятие решений.

## Компьютерные игры

В данной работе рассматриваются методы и алгоритмы принятия решений на примере ботов в компьютерной игре. Компьютерные игры или видеоигры – это компьютерные программы для организации игрового процесса. Назначение видеоигр схоже с назначением произведений литературы и кинематографа: они могут использоваться для развлечения, образования или передачи авторской идеи.

Игровая индустрия или индустрия интерактивных развлечений – это один активно развивающихся секторов экономики. Согласно прогнозам аналитиков, оборот игровой индустрии к 2018 году достигнет 96 миллиардов долларов [Video Game Market Overview. DFC Intelligence. Апрель 2016]. Бюджеты крупных игровых проектов сравнялись с бюджетами фильмов-блокбастеров [http://www.scotsman.com/lifestyle/gadgets-gaming/new-gta-v-release-tipped-to-rake-in-1bn-in-sales-1-3081943].

Разработка видеоигр – это технологически сложный процесс. Развитие видеоигр также способствует развитию многих компьютерных наук, таких как компьютерная графика, моделирование физических процессов, алгоритмы и структуры данных, искусственный интеллект и другие.

## Боты в компьютерных играх

В общем случае бот – это компьютерная программа, выполняющая какие-либо действия через интерфейсы, предназначенные для людей. В частном случае компьютерных игр бот – это программа, имитирующая поведение живых игроков в многопользовательских играх. Это агент, который анализирует окружающую обстановку, принимает решения и выполняет действия, которые дают ему игровые преимущества и приводят к выигрышу в соответствии с заданными правилами игры.

Благодаря большой вычислительной мощности современных компьютеров боты могут принимать более эффективные решения чем игроки и с более высокой скоростью. Однако, так как основной задачей видеоигр является развлечение игрока, боты должны имитировать поведение живого человека, играющего с ним на равных. В противном случае игра против оппонента, которого практически невозможно победить, быстро приведет к потере интереса.

## Правила игры

В данной работе рассматривается пример игры в жанре «шутер с видом сверху» (top-down shooter). В играх этого жанра игроки управляют вооруженными персонажами, сражаются с другими игроками или ботами и наблюдают за полем боя с высоты птичьего полета. Графика может быть как двухмерной, так и трехмерной, но игровая логика как правило рассчитывается в двухмерной координатной системе.

{добавить скриншоты из разрабатываемой игры, когда она появится}

Правила игры:

* Игроки обладают ограниченной дальностью и углом обзора.
* В игре есть стены и укрытия. За стеной не видно других игроков. Укрытия не мешают обзору, но все выстрелы могут с вероятностью 50% быть заблокированными укрытием, если стреляющий игрок стоит от него дальше, чем на 1 метр.
* Игроки могут делиться на команды.
* Игра ведется до поражения всех оппонентов. Выигрывает команда, в которой остались игроки.

В терминах теории игр рассматриваемая игра:

* симметричная;
* кооперативная;
* с нулевой суммой;
* непрерывная;
* параллельная;
* с неполной информацией.

Существует большое количество игр этого жанра, но во всех этих играх в качестве врагов игрока используются слабые монстры, которые нападают со всех сторон и пытаются одолеть игрока количеством. В них в качестве развлекательного фактора используется постоянное движение и стрельба по большому количеству целей. В данной работе предлагается рассмотреть симметричную многопользовательскую игру, в которой все участники, как живые, так и компьютерные, находятся в равных условиях. В этом случае победитель определяется навыками: быстротой реакции, качеством тактических решений, меткостью стрельбы.

## Цель и задачи работы

Целью данной работы является исследование методов моделирования принятия решений на примере ботов в компьютерной игре.

Задачи, решаемы в ходе работы:

* рассмотреть существующие методы и алгоритмы принятия решений;
* сформулировать задачи, которые должен решать бот;
* для каждой задачи выбрать методы и алгоритмы, подходящие для ее решения, обосновать выбор;
* реализовать выбранные алгоритмы;
* протестировать полученные реализации;
* сделать выводы об эффективности различных методов и алгоритмов для моделирования процесса принятия решений.

# Обзор методов и алгоритмов принятия решений

## Рулевое управление (steering behavior)

В [Reynolds] была предложена модель, описывающая поведение автономных персонажей и их взаимодействие друг с другом посредством векторов сил. Автономные персонажи – это агенты, использующиеся в анимации и в интерактивных системах, таких как видеоигры или симуляции. Эта модель дает описание параметров агентов и набор взаимодействий, которые можно комбинировать друг с другом для получения различных стратегий поведения. Различные комбинации стратегий позволяют моделировать движение транспортных средств, поведение охотника и жертвы, толпы людей, стаи птиц или косяка рыб.

Агент моделируется инерционной материальной точкой, которая обладает следующими параметрами:

* масса;
* позиция в пространстве;
* ориентация в пространстве;
* скорость;
* максимальная сила;
* максимальная скорость.

На каждом шаге симуляции параметры обновляются в соответствии с физическими законами:

сила = min(рулевое воздействие, максимальная сила);

ускорение = сила / масса;

скорость = min(скорость + ускорение, максимальная скорость);

позиция = позиция + скорость;

ориентация пересчитывается через вектор скорости.

Рулевое воздействие определяется стратегией движения.

### Движение к точке (seek)

Стратегия, согласно которой агент старается максимально сблизиться с целью. Для того, чтобы агент двигался к точке максимально эффективно, необходимо придать следующее воздействие (рисунок 2):

воздействие = текущая скорость – желаемая скорость.



Рисунок 2. Движение к точке.

### Бегство (flee)

Стратегия, согласно которой агент старается максимально отдалиться от цели. Идентично движению к точке, но с обратным направлением вектора желаемой скорости.

### Прибытие (arrive)

Аналогично движению к точке, но агент начинает тормозить на некотором расстоянии от цели, чтобы при ее достижении скорость была равна нулю. Для этого задаются торможение и порог – расстояние до цели, начиная с которого скорость должна падать. Если расстояние больше порогового, то скорость рассчитывается как обычно. Если меньше, то:

скорость = min(желаемая скорость \* расстояние до цели / порог, максимальная скорость).

Коэффициент «расстояние до цели / порог» дает линейное торможение при приближении к цели.

### Преследование (pursuit)

Стратегия, согласно которой агент старается максимально сблизиться с движущейся целью. Воздействие рассчитывается аналогично движению к точке, но целью движения является не сам преследуемый объект, а его прогнозируемая позиция в будущем с предположением, что объект движется прямо и не поворачивает. Так как в большинстве случаев предположение о прямолинейном движении не верно, появится ошибка, но за счет того, что воздействие пересчитывается на каждом шаге симуляции, она будет мала.

В качестве цели обычно используется позиция объекта через время T. Идеальным значением T является время до встречи с объектом. Однако, из-за непредсказуемого поведения объекта, эту величину сложно предсказать, поэтому используются приближенные оценки T, полученные на основании расстояния между объектами и из взаимном расположении (движутся навстречу друг другу или догоняют).

### Преследование со смещением (offset pursuit)

Аналогично стратегии преследования, но агент старается находиться на фиксированном смещении от цели. Используется для следования за лидером и поддержания строя.

### Уклонение (avoid)

Стратегия, согласно которой агент старается максимально отдалиться от движущейся цели. Стратегия аналогична стратегии преследования, но с обратным направлением вектора желаемой скорости. На рисунке 3 показаны маршруты преследования и уклонения.



Рисунок 3. Преследование и уклонение.

### Скитание (wander)

Стратегия, согласно которой агент движется по местности, имитируя движение без цели. На каждом шаге симуляции к вектору скорости добавляется случайное смещение в сторону, которое плавно отклоняет агента от прежнего курса (рисунок 4).



Рисунок 4. Скитание.

### Избегание препятствий (obstacle avoidance)

Стратегия, согласно которой агент старается избегать столкновений со статическими объектами, находящимися у него на пути. Перед агентом в направлении движения моделируется прямоугольник, ширина которого равна габаритам агента, а длина пропорциональна скорости движения. Если этот прямоугольник пересекается с препятствием, то к агенту прикладывается воздействие, отклоняющее его от этого препятствия (рисунок 5).



Рисунок 5. Избегание препятствий.

### Разделение (separation)

Групповая стратегия, согласно которой агенты стараются держаться на расстоянии друг от друга. Если расстояние между двумя агентами превышает заданное пороговое значение, то к ним прикладывается воздействие, отводящее их друг от друга. Группа агентов, следующих такой стратегии будет распределяться по местности, пока расстояние между ними не станет больше порогового.

### Выравнивание (alignment)

Групповая стратегия, согласно которой агенты стараются сохранять одинаковую ориентацию в пространстве. Для этого на каждом шаге симуляции вычисляется средний вектор направления агентов и принимается за желаемое направление. Вычитая из него текущее направление агента, получаем воздействие, которое нужно приложить, чтобы его повернуть.

### Сплочение (cohesion)

Групповая стратегия, согласно которой агенты стараются быть ближе друг к другу. Для этого вычисляется воздействие, которое двигает агентов к центру масс группы.

## Потенциальные поля / тепловые карты

## Поиск в ширину

## Алгоритм Дейкстры

Алгоритм Дейкстры – алгоритм на графах, изобретенный нидерландским ученым Эдсгером Дейкстрой в 1959 году. В общем случае находит кратчайшие пути от выбранного узла до всех остальных, результатом работы является дерево кратчайших путей [Кормен]. Алгоритм Дейкстры и подобные ему алгоритмы широко применяется в протоколах маршрутизации при обменах данными по сети, для решения логистических задач и для ориентирования систем искусственного интеллекта в пространстве. В последнем случае карта местности представляется в виде графа, часто называемого навигационной сеткой (navigation mesh, navmesh) (рисунок 6Рисунок 6).



Рисунок 6. Пример навигационной сетки, используемой в видеоиграх.

Рассмотрим пример работы алгоритма на графе, изображенном на рисунке 7. Найдем кратчайшие пути от вершины 1 до всех остальных. Каждой вершине сопоставляется метка со значением, равным длине пути от начальной вершины до нее. Перед началом работы алгоритма метке исходной вершины устанавливается значение ноль, а всем остальным – бесконечность или другое особое значение, обозначающее отсутствие пути (рисунок 8). В ходе работы алгоритма эти значения могут уменьшаться, если был найден более короткий путь до вершины.

На каждой итерации алгоритма вершина с минимальной меткой устанавливается в качестве текущей и помечается звездочкой, что означает, что до данной вершины найдено кратчайшее расстояние и уменьшаться оно больше не будет. В данном случае выбирается вершина 1, так как ее метка равно нулю, а остальные бесконечности. Для всех вершин, смежных с текущей, пересчитывается значение метки: новое значение равно сумме значения текущей вершины и веса ребра между этими вершинами. Если новое значение меньше текущего, то метка обновляется, а в отдельную структуру сохраняется текущая вершина.

dist[2] = min{∞, 0 + 2} = 2, prev[2] = 1

dist[7] = min{∞, 0 + 2.83} = 2.83, prev[7] = 1

Результат первой итерации представлен на рисунке 9.

|  |  |
| --- | --- |
| Граф для примера.png  Рисунок 7. Исходный взвешенный граф. | Граф для примера.png  Рисунок 8. Начальное состояние меток. |
| Граф для примера.png  Рисунок 9. Результат первой итерации. | Граф для примера.png  Рисунок 10. Результат второй итерации. |

Выбирается новая вершина с минимальным значением из тех, что еще не были посещены (т.е. не отмечены звездочкой), и пересчет меток повторяется для вершин смежных с ней. В данном примере такой вершиной является вершина 2.

dist[3] = min{∞, 2 + 2} = 4, prev[3] = 2

Результат второй итерации представлен на рисунке 10.

Обход вершин продолжается либо пока не будут посещены все вершины, либо пока не останутся вершины только со значением равным бесконечности. Второй вариант означает, что есть вершины, не достижимые из исходной. Результат обхода вершин из примера представлен на рисунке 11.

Для того, чтобы получить кратчайший путь из исходной вершины к другой, необходимо, используя структуру prev, восстановить путь, начиная с целевой вершины.



Рисунок 11. Итоговые значения меток.

Рассмотрим получение пути от вершины 1 до вершины 5:

prev[5] = 4;

prev[4] = 7;

prev[7] = 1.

Оптимальный путь из вершины 1 в вершину 5: 1-7-4-5.

Если требуется определить путь только до одной вершины, то в качестве критерия останова можно использовать достижение алгоритмом целевой вершины. При этом изменяется инициализация: нет смысла изначально помечать все вершины специальным символом или бесконечностью, так как некоторые из них могут быть не рассмотрены вовсе. Вместо этого, при пересчете вершины необходимо проверять, рассчитывается ли метка в первый раз, и инициализировать ее.

Рассмотрим псевдокод алгоритма Дейкстры [Дасгупта]:

procedure Dijkstra(graph, start, goal) {

processed = {} // просмотренные вершины

dist[start] = 0 // рассматриваемые вершины и их метки

prev[start] = null // предыдущие вершины

// обход вершин

while (dist не пусто) {

v = «извлечь из dist вершину с минимальным расстоянием от исходной вершины»

«добавить v в processed»

if (v == goal)

break // целевая вершина достигнута

for (e ∈ дуги, исходящие из v) {

v2 = e.getDestination()

if (processed содержит v2)

continue // вершина уже была посещена

d = dist[v] + e.getWeight()

if (dist не содержит v2 OR dist[v2] > d) {

dist[v2] = d

prev[v2] = v

}

}

}

}

Сложность алгоритма зависит от используемых реализаций структур данных [Кормен]. Проанализируем сложность выполнения псевдокода из раздела. Пусть V – это количество вершин графа, а E – количество ребер. Алгоритм посещает каждую вершину максимум один раз (ни разу, если она не достижима из исходной), при этом для каждой вершины алгоритм проверяет все исходящие дуги (ребра) и пересчитывает известное расстояние для смежных вершин. Можно сформулировать выражение для оценки сложности для неизвестных АТД:

TДейкстра(V, E) ~ V·(Tdist.getMin(V) + Tprocessed.set(V)) +

+ E·(2·Tdist.get(V) + Tdist.set(V) + Tprev.set(V))

Если выбрать для их реализации сбалансированные деревья, получим:

TДейкстра(V, E) ~ V·(O(log V) + O(log V)) + E·(O(log V) + O(log V) + O(log V))

~ O((V + E) · log V)

Если какой-либо из операторов имеет степень роста O(n), то по правилу умножения получаем:

TДейкстра(V, E) ~ O(V2)

Таким образом, степень роста алгоритма Дейкстры определяется самым медленным оператором используемых в нем структур данных.

## Алгоритм A\*

Алгоритм A\* (произносится «А звезда» или «А стар», от англ. A star) – алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины к другой. В 1964 году Нильс Нильсон изобрел эвристический подход к увеличению скорости алгоритма Дейкстры. Этот алгоритм был назван А1. В 1967 году Бертрам Рафаэль сделал значительные улучшения по этому алгоритму, но ему не удалось достичь оптимальности. Он назвал этот алгоритм A2. Тогда в 1968 году Питер Э. Харт представил аргументы, которые доказывали, что A2 был оптимальным при использовании последовательной эвристики лишь с незначительными изменениями. Таким образом, он обозначил новый алгоритм в синтаксисе звездочкой, он начинается на А и включает в себя все возможные номера версий.

Аналогично алгоритму Дейкстры A\* поочередно просматривает вершины и обновляет минимальное известное расстояние от исходной вершины до текущей. Отличие заключается в получении следующей вершины из очереди. Помимо информации о расстоянии от исходной вершины, A\* также использует информацию о положении целевой вершины. Каждая следующая вершина выбирается по минимуму суммы f(v) = g(v) + h(v), где g(v) – минимальное известное расстояние от исходной вершины до вершины v, h(v) – значение эвристической функции для вершины v.

Эвристика позволяет сократить количество вершин, просмотренных алгоритмом, предлагая направление для поиска, которое позволит приблизиться к цели, но при этом не гарантирует, что полученное приближение будет верным. Эвристика может использоваться для настройки и управления алгоритмом [Amit’s A\* Pages]:

* Если эвристическая функция всегда возвращает ноль, то алгоритм превращается в алгоритм Дейкстры, т.е. для выбора вершины используется только расстояние от исходной вершины.
* Если оценка всегда меньше либо равна реальной дальности от вершины до цели, то A\* гарантированно вернет оптимальный путь.
* Если эвристическая функция возвращает точное расстояние от вершины до цели, то алгоритм посетит только вершины, входящие в оптимальный путь.
* Если оценка завышена, т.е. ее значение для некоторых вершин превышает реальную длину пути, алгоритм не гарантирует, что найденный путь будет оптимальным. Однако, завышенная оценка отсеивает большое количество вершин, что позволяет увеличить быстродействие за счет потери качества.
* Если оценка много больше расстояния от исходной вершины, то поведение алгоритма близко к жадному поиску по первому наилучшему совпадению.

Эвристическая функция называется допустимой эвристической оценкой, если она не переоценивает расстояние между узлами.

Как правило, A\* используются на графах, основанных на физическом представлении, например, в прямоугольной системе координат. Используя геометрические свойства вершин графа, можно оценить минимально возможное расстояние между ними. Рассмотрим эвристические функции, которые обычно применяются с A\* [Buckland, Amit’s A\* Pages]:

* Манхэттенское расстояние. Метрика, введённая Германом Минковским. Согласно этой метрике, расстояние между двумя точками равно сумме модулей разностей их координат (рисунок 12). Также имеет название «расстояние городских кварталов». Используется, если перемещение по клеткам допускается только в четырех направлениях.

.



Рисунок 12. Манхэттенское расстояние между двумя узлами.

* Диагональное расстояние. Используется, если разрешено движение между диагональными клетками. Одна из разностей координат проходится по диагонали, другая – по прямой аналогично манхэттенскому расстоянию (рисунок 13).

где .



Рисунок 13. Диагональное расстояние между двумя узлами.

* Эвклидово расстояние. Представляет собой геометрическое расстояние между двумя точками в пространстве (рисунок 14). Гарантированно дает минимальную оценку расстояния.

.



Рисунок 14. Эвклидово расстояние между двумя узлами.

Рассмотрим псевдокод алгоритма A\*:

procedure AStar(graph, start, goal, heuristic) {

closed = {} // посещенные вершины

open = {start} // рассматриваемые вершины и их метки

prev[start] = null // структура для хранения предшествующих вершин

while (open не пусто) {

v = «извлечь из open вершину с минимальным значением f = g + h»

«добавить v в closed»

if (v == goal)

break // целевая вершина достигнута

for (e ∈ дуги, исходящие из v) {

v2 = e.getDestination()

if (closed содержит v2)

continue // вершина уже была посещена

newG = g[v] + e.getWeight()

if (open не содержит v2) {

h[v2] = heuristic(v2, goal)

g[v2] = newG

prev[v2] = v

} else {

if (g[v2] > newG) {

g[v2] = newG

prev[v2] = v

}

}

}

}

}

Временная сложность алгоритма зависит от эвристики. В худшем случае число вершин, исследуемых алгоритмом растет экспоненциально по сравнению с длиной оптимального пути: O(bd), где b – коэффициент ветвления (средняя степень вершины), d – длина оптимального пути. Сложность становится полиномиальной, если эвристическая функция удовлетворяет следующему условию:

, где – это оптимальная эвристика, то есть точное расстояние от x до цели [Рассел и Норвиг].

## Булева логика и системы, основанные на правилах

## Нечеткая логика

Нечеткая логика (fuzzy logic) является расширением классической математической логики [Zadeh]. В нечеткой логике каждая переменная может принимать вещественное значение в диапазоне [0; 1]: 0 соответствует значению «ложь», 1 – значению «истина», промежуточные значения – частичной истинности. Область значений переменных в классической логике является подмножеством области значений переменных в нечеткой логике.

Нечеткая логика применяется в условиях неопределенности, вызванных отсутствием четкого знания о среде. Такая неопределенность может возникнуть из-за аппаратурной погрешности датчиков, влияния шума, отсутствия теоретической или практической возможности вычислить нужную величину или из-за природы самой величины.

Операции в нечеткой логике могут быть определены разными способами в зависимости от решаемой задачи. При этом они должны быть применимы к переменным классической логики, т.е. при подстановке значений 0 и 1 они должны давать результаты, соответствующие таблицам истинности операций из классической логики.

### Отрицание

Отрицание нечеткой логической переменной x можно получить путем вычитания ее значения из единицы [Zadeh]:

¬x = 1 – x.

### Конъюнкция и дизъюнкция

Конъюнкция и дизъюнкция в нечеткой логике задаются с помощью треугольных нормы T и конормы S соответственно [Батыршин].

Функции T, S: [0; 1] × [0; 1] → [0; 1] называются треугольной нормой (t-нормой) и треугольной конормой (t-конормой), если они:

* 1. монотонны: T(x, y) ≤ T(y, z), S(x, y) ≤ S(y, z), если x ≤ y ≤ z;
  2. ассоциативны: T(T(x, y), z) = T(x, T(y, z)), S(S(x, y), z) = S(x, S(y, z));
  3. коммутативны: T(x, y) = T(y, x), S(x, y) = S(y, x);
  4. связаны соотношениями де Моргана: ¬T(x, y) = S(¬x, ¬y) и ¬S(x, y) = T(¬x, ¬y);
  5. удовлетворяют граничным условиям: T(0, 0) = T(0, 1) = T(1, 0) = 0, T(1, 1) = 1, S(0, 0) = 0, S(0, 1) = S(1, 0) = S(1, 1) = 1.

Примеры определений конъюнкции и дизъюнкции, удовлетворяющих перечисленным условиям [Батыршин, Штовба]:

Пересечение и объединение по Заде [Zadeh]:

x ∧ y = min(x, y);

x ∨ y = max(x, y).

Троичная логика Лысенко:

x ∧ y = max(a + b – 1, 0);

x ∨ y = min(a + b, 1).

Вероятностные пересечение и объединение:

x ∧ y = xy;

x ∨ y = x + y – xy.

### Импликация

Импликация определяет причинно-следственное отношение между посылками и заключениями логических правил. Перечислим несколько вариантов определения импликации [Тенетко].

Импликация Геделя (Godel):

x → y = .

Импликация Гогена (Goguen):

x → y = .

Импликация Клини-Дайнса (Kleene-Dienes):

x → y = max(1 - x, y).

### Нечеткие множества

На нечеткой логике строится теория нечетких множеств. Ее основная идея заключается в том, что в отличие от классической теории множеств, где элемент либо принадлежит множеству, либо нет, здесь каждый элемент имеет меру вхождения с областью значений [0; 1].

Пусть 𝔛 = {x} – множество всех рассматриваемых элементов, A ⊆ 𝔛 – нечеткое множество. Для A определена функция принадлежности (membership function):

μA: 𝔛 → [0; 1].

Значение μA(x) = 0 означает, что элемент x не входит в множество A, 1 – что входит полностью, промежуточные значения означают частичное вхождение.

Запись нечетких множеств обычно имеет вид [Bai]:

A = .

При этом знак + не является суммированием, а горизонтальная черта не является делением – такая запись принята в литературе и означает перечисление элементов вместе с их мерами вхождения.

### Операции над нечеткими множествами

Нечеткие множества являются расширением обычных множеств, поэтому операции над ними должны быть применимы так же и к обычным множествам.

Операции над нечеткими множествами определены следующим образом [Zadeh]:

A = B ⇔ ∀x μA(x) = μB(x),

A ⊂ B ⇔ ∀x μA(x) ≤ μB(x),

A ⋂ B: μA⋂B(x) = μA(x) ∧ μB(x) = T(μA(x), μB(x)),

A ⋃ B: μA⋃B(x) = μA(x) ∨ μB(x) = S(μA(x), μB(x)).

На рисунке 15 показаны два примера нечетких множеств: A – множество чисел, примерно равных 2, B – множество чисел, примерно равных 3.



Рисунок 15. Примеры нечетких множеств.

На рисунках 16, 17 и 18, 19 показаны соответственно объединения и пересечения этих множеств с использованием разных определений дизъюнкции и конъюнкции. Выбор конкретного определения зависит от решаемых задач. На примере представленных множеств объединение с использованием произведения функций принадлежности является более подходящим, так как дает более плоское плато у объединенной функции. В этом случае число 2.5 в большей степени принадлежит множеству чисел примерно равных 2 и примерно равных 3, чем 1.5 или 3.5, что более логично.



Рисунок 16. Объединение множеств с использованием нормы, определенной через максимум функций принадлежности.



Рисунок 17. Объединение множеств с использованием нормы, определенной через сумму функций принадлежности.



Рисунок 18. Пересечение множеств с использованием конормы, определенной через минимум функций принадлежности.



Рисунок 19. Пересечение множеств с использованием конормы, определенной через произведение функций принадлежности.

### Лингвистические переменные

Лингвистической переменной называется переменная, значениями которой являются слова или предложения естественного или искусственного языка [Zadeh 1976]. Например, «возраст» - лингвистическая переменная, если она принимает лингвистические, а не числовые значения, т.е. значения «молодой», «не молодой», «очень молодой», «старый», «не очень старый» и другие.

Нечеткая лингвистическая переменная (или просто лингвистическая переменная) – это набор нечетких множеств с одинаковой областью определения, каждое из которых определяет одно лингвистическое значение [Buckland]. Такие лингвистические значения также называются термами [Штовба]. Примеры лингвистических переменных:

Возраст = { молодой, не очень молодой, не очень старый, старый };

Температура = { холодно, прохладно, нормально, тепло, жарко };

Угловое отклонение = { сильно влево, влево, немного влево, нет, немного вправо, вправо, сильно вправо },

На рисунке 20 изображено графическое представление переменной «угловое отклонение».



Рисунок 20. Графическое представление лингвистической переменной «угловое отклонение».

Строгих правил для определения лингвистических переменных и их термов нет, но существуют рекомендации, позволяющие упростить разработку, улучшить качество и скорость вычислений [Buckland].

* Функции принадлежности термов должны в совокупности иметь ненулевые значения на всей области определения переменной.
* Сумма значений функций принадлежности всех термов для каждой точки области определения переменной должна быть примерно равна 1 для получения результата без разрывов.
* Форма функции принадлежности в общем случае может быть любой, но рекомендуется выбирать кусочно-линейные функции (треугольные и трапециевидные) для простоты расчетов.
* Для каждой точки области определения должно быть не более двух термов с ненулевой мерой для простоты расчетов.

### Дефаззификация

Результатом нечеткого вывода является набор термов – нечетких множеств. Для получения четкого значения, которое затем можно использовать в качестве управляющего воздействия, необходимо произвести дефаззификацию.

Рассмотрим основные методы дефаззификации [Bai, Buckland]:

Центр максимумов (mean of maximum):

x\* = ,

где a, b – самая левая и самая правая точки, в которых функция принимает максимальное значение. На рисунке 21 изображен пример дефаззификации с помощью центра максимумов. Данный метод позволяет быстро рассчитать результат, но он не учитывает влияние множеств со степенью достоверности меньше максимальной.



Рисунок 21. Пример дефаззификации с помощью центра максимумов.

Центр масс (center of gravity, centroid):

x\* = для непрерывных величин,

x\* = для дискретных величин.

Один из самых распространенных методов. Дает точный результат, но сложно рассчитывается. Пример результата изображен на рисунке 22 Для ускорения расчетов можно увеличить шаг суммирования (рисунок 23).



Рисунок 22. Пример дефаззификации с помощью центра масс.



Рисунок 23. Пример разбиения области определения на дискретные интервалы.

Взвешенная сумма (weighted average):

x\* = ,

где di – i-й терм выходной лингвистической переменной, mom(di) – средний максимум i-го терма выходной переменной.

Выбор метода дефаззификации, как и выбор нормы и конормы, зависит от решаемой задачи и обычно подбирается экспериментально.

### Преимущества и недостатки нечеткой логики

Нечеткая логика применяется, когда нет четкого знания об окружающей среде. Это может быть вызвано недостатками датчиков, отсутствием модели окружающего мира или сложностью ее применения. Нечеткая логика позволяет свести сложный математический аппарат предметной области к набору эмпирических правил вида «если X, то Y», основываясь на экспертном мнении и экспериментах.

Основным недостатком нечеткой логики является отсутствие математической строгости и неоднозначность определений. Например, разными способами могут быть определены: конъюнкция, дизъюнкция, импликация, набор и форма правил вывода, набор входных и выходных переменных, термы переменных, функции принадлежности термов, методы дефаззификация и другое. Нет возможности заранее рассчитать, как то или иное определение повлияет на результат, поэтому зачастую разработка нечеткой системы управления сводится к перебору параметров и методов, пока результат не будет удовлетворять требованиям. При этом увеличение количества входных переменных и их термов приводит к комбинаторному взрыву количества правил вывода, требуемых для описания системы. Существуют методы для уменьшения количества правил вывода, но они также влияют на результат и требуют экспериментов для определения их применимости.

## Системы, основанные на правилах (экспертные системы)

## Конечные автоматы

## Недетерминированные конечные автоматы

[Шампандар]

## Вероятностные конечные автоматы

[Шампандар]

## Нечеткие конечные автоматы

[Шампандар]

## Списки действий

## Деревья поведения

[Buckland]

## Планировщики

[Buckland]

## Теория полезности

[Dave Mark]

## Вероятностный подход и проверка статистических гипотез

Когда агент не обладает полными знаниями об окружающей среде, он должен строить модель мира и прогнозировать его изменения. Так как изменение среды в большинстве случаев является случайным процессом, наличие модели позволяет строить предположения о текущем состоянии окружающего мира только с определенной степенью уверенности. Степень уверенности может быть описана с помощью случайных переменных, для которых можно применить теорию вероятности и математическую статистику [Рассел и Норвиг, Ивановский].

Каждый отдельный параметр окружающей среды представляет собой случайную переменную. В зависимости от области определения выделяют три вида переменных:

* Булевы переменные. Могут принимать значения «истина» или «ложь».
* Дискретные переменные. Могут принимать значения из конечного множества значений.
* Непрерывные переменные. Могут принимать значения из бесконечного множества действительных чисел или его подмножества.

Для каждого возможного значения ai переменной A задается вероятность P(A=ai) того, что переменная имеет это значение. Запись P(ai) аналогична записи P(A=ai). Так же возможна векторная запись:

P(A) = <P(a1), P(a2), …, P(an)>.

Выражение P(x) часто читается как «вероятность события x».

Для непрерывных переменных задается плотность вероятности f(a), а вероятность того, что переменная будет иметь некоторое значение в окрестности ai, рассчитывается как интеграл плотности:

.

Вероятность P(A) в отсутствии любой другой информации называется безусловной или априорной. При наличии информации о других переменных априорная вероятность более не применима, так как переменные могут влиять друг на друга. В этом случае используется вероятность P(A|B) – вероятность события A при условии того, что произошло событие B. Такая вероятность называется условной или апостериорной.

Апостериорная вероятность рассчитывается по правилу (теореме) Байеса:

Оно позволяет пересчитать вероятность A, взяв в расчет как ранее известную информацию об A, так и данные о новых наблюдениях – событиях B. Если априорная вероятность P(A) не известна, ее заменяют некоторой оценкой, обычно априорное распределение принимают равномерным.

Выражение P(B|A) в теореме Байеса называется функцией правдоподобия. Это функция от параметра A, которая показывает, насколько были бы правдоподобны наблюдения B при условии, что имело место событие A.

Принятие решений с использованием вероятностных методов сводится к задаче проверки статистических гипотез – одного из классов задач математической статистики. Статистическая гипотеза – предположение о виде распределения и свойствах случайной величины, которое можно подтвердить или опровергнуть с помощью методов математической статистики [Ивановский, Васин].

Рассмотрим в качестве примера агента, участвующего в некоторой карточной стратегической игре с неполной информацией. Случайной величиной является набор карт в руках оппонента, а также в некоторых случаях выбранная им стратегия. Рациональный агент должен принимать решения, которые увеличивают его шансы на выигрыш. Гипотезой является предположение о конкретном значении случайной величины. Выдвигается множество гипотез H={hi}n, по одной для каждого возможного значения. Если априорная вероятность значений не известна, то берется равномерное распределение P(hi) = const. После того, как оппонент делает ход, мы наблюдаем событие и рассчитываем апостериорные вероятности гипотез по теореме Байеса. Агент выбирает одну гипотезу из множества в соответствии с некоторым критерием, принимает ее за текущее состояние окружающего мира и выполняет ход, который согласно установленным в него правилам является оптимальной реакцией на такое состояние окружающей среды.

При этом всегда существует вероятность принятие ошибочного решения при нехватке информации или недостаточном количестве наблюдений. Для количественной оценки ущерба (потерь), связанного с принятием решений, вводится функция потерь Π(hi, γj), которая задает потери при принятии решения hi, при условии, что правильным решением было γj.

### Критерий минимума среднего риска

Качество решения можно характеризовать математическим ожиданием функции потерь:

,

где – средний риск, – вероятность того, что одновременно принята гипотеза hi и правильным решением является γj.

Правило, при котором выбирается гипотеза с минимальным средний риском, называется байесовским правилом, байесовским критерием или критерием минимума среднего риска.

### Минимаксный критерий

Для использования критерия минимума среднего риска требуется большое количество дополнительной априорной информации о вероятностях правильности различных гипотез. Если такой информации нет, используется минимаксный критерий – выбирается гипотеза, у которой максимальная возможная ошибка минимальна. Он позволяет уменьшить потери в худшем случае, но не учитывает вероятность его возникновения.

### Критерий минимума полной вероятности ошибки

Если все ошибочные решения одинаково нежелательны, можно задать функцию потерь постоянной и не учитывать при принятии решения:

.

Тогда средний риск принимает вид:

Получаем частный случай байесовского критерия, который также называют критерием минимума полной вероятности ошибки или критерием идеального наблюдателя.

### Критерий максимума апостериорной вероятности

При постоянной функции потерь все ошибки имеют одинаковую стоимость, поэтому при принятии решения можно исходить только из свойств самих гипотез (их апостериорных вероятностей) и наблюдаемых событий. В этом случае решение принимается в пользу гипотезы с максимальной апостериорной вероятностью P(hi|E).

По теореме Байеса:

где E – наблюдаемые события и текущее состояние мира, P(E) – априорная вероятность текущего состояния (одинакова для всех гипотез, поэтому информация о ней не требуется при принятии решения).

### Критерий максимального правдоподобия

Если не известна информация об априорных вероятностях гипотез, то их распределение принимают за равномерное, т.е. P(hi)=const. Тогда получаем:

В этом случае критерий максимума апостериорной вероятности сводится к критерию максимальной правдоподобности.

### Преимущества и недостатки вероятностного подхода

Вероятностный подход применим к случаям, когда модель окружающей среды хорошо изучена, т.е. есть информация об априорных вероятностях, функциях правдоподобия, функциях потерь и других. К таким случаям можно отнести пример с карточной игрой, рассмотренный ранее. Зная карты у себя на руках и карты на столе, агент может рассчитать вероятности того или иного набора карт в руке оппонента и принять оптимальное решение в соответствии с заданным критерием. Другим примером является обнаружение и различение радиосигналов цифровым приемником [Васин]. Априорными данными является излученный сигнал – закодированный символ из известного конечного алфавита, апостериорными – принятый сигнал, являющийся суммой излученного сигнала с учетом затухания и шума. В отличие от примера с карточной игрой, в данном случае нельзя рассчитать точные вероятности того или иного события, но можно произвести оценки, основанные на статистической модели канала передачи и предположении о распределении шума.

В отсутствии подробной модели, приходится делать упрощения: априорные параметры принимать за равномерное распределение, потери считать равнозначными, делать предположения о функциях правдоподобия. Это приводит к уменьшению качества принятых решений. Статистические методы основаны на наблюдениях, поэтому чтобы сделать правильный выбор из набора гипотез, необходимо увеличивать количество наблюдений, а это не всегда возможно.

## Нейронные сети

[Шампандар]

## Генетические алгоритмы

[Шампандар]

# Выбор методов и алгоритмов для разработки

## Задачи, решаемые ботами, и подходящие для их реализации методы и алгоритмы

### Поиск пути

{Курсовик по алгоритмам и структурам данных}

### Прицеливание и стрельба

[Шампандар]

### Тактические решения

[Terrain Reasoning, Шампандар]

### Многоагентное взаимодействие

# Алгоритмизация и реализация

## Выбор и обоснование языка программирования и библиотек

Рассматриваемые методы и алгоритмы принятия решений не зависят от используемых технологий и языка программирования. В данной работе используется игровой движок Unity. Это один из самых популярных игровых движков {добавить ссылку на статистику}, позволяющий быстро создавать прототипы и полноценные игры. В качестве языка игровой логики в нем используются C# и javascript. В данной работе будет использоваться C#, так как он обладает более удобной объектной моделью и поддерживает сильную типизацию, которая позволяет уменьшить количество ошибок в ходе разработки.

## Интерфейс управления персонажем

Для управления персонажем используется следующий интерфейс:



Компонент, реализующий этот интерфейс, может принимать команды как от компонента, отвечающего за управление с помощью устройств ввода, так и от компонента, отвечающего за искусственный интеллект.

# Список использованных источников

Рассел С., Норвиг П. Искусственный интеллект: современный подход, 2-е изд. : Пер. с англ. – М.: Издательский дом «Вильямс», 2006.

Шампандар Алекс Дж. Искусственный интеллект в компьютерных играх. Как обучить виртуальные персонажи реагировать на внешние воздействия. : Пер. с англ. – М.: Издательский дом «Вильямс», 2007.

Сирл Д. Разум мозга — компьютерная программа? – В МИРЕ НАУКИ. (Scientific American. Издание на русском языке). 1990. № 3

Кормен T., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е издание. –М.: ООО «И.Д. Вильямс», 2013. – 1328 с. : ил.

Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы; Пер. с англ. под ред. А. Шеня. –М.: МЦНМО, 2014. – 320 с.

Amit Patel. Amit’s A\* Pages / Stanford University – URL: <http://theory.stanford.edu/~amitp/GameProgramming/index.html> – (дата обращения: 10.01.2016).

Buckland M. Programming game AI by example. – Wordware Publiching Inc. 2005.

Zadeh L. A. Fuzzy sets. / Information and control 8. 1965.

Батыршин И. З., Недосекин А. О. Нечеткие гибридные системы. Теория и практика. / Под ред. Н.Г. Ярушкиной. – М.: ФИЗМАТЛИТ, 2007.

Штовба С. Д. Введение в теорию нечетких множеств и нечеткую логику [Электронный ресурс] / http://matlab.exponenta.ru/fuzzylogic/book1/1.php, дата обращения: 20.08.2016.

Тенетко М. И., Пескова О. Ю. Проведение эксперимента по обоснованию выбора нечёткой импликации, пригодной для решения задач классификации рисков и выработки наилучших рекомендаций по рискам. – Известия Южного федерального университета. Технические науки, выпуск № 12 (137) / том 137 / 2012.

Bai Y., Wang D. Advanced Fuzzy Logic Technologies in Industrial Applications (Advances in Industrial Control). – Springer, 2006.

Заде Л. Понятие лингвистической переменной и его применение к принятию приближенных решений. – М.: Мир, 1976.

Ивановский Р. И. Теория вероятностей и математическая статистика. Основы, прикладные аспекты с примерами и задачами в среде Mathcad. – СПб.: БХВ-Петербург, 2008.

Васин В. А., Калмыков В. В. Радиосистемы передачи информации: Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2005.

Mark D. Behavioral Mathematics for Game AI. – Course Technology, a part of Cengage Learning, 2009.

Palacios J. Unity 5.x Game AI Programming Cookbook. – Packt Publishing, 2016.

Reynolds, C. W. Steering Behaviors For Autonomous Characters. In the proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California. Pages 763-782.

William van der Sterren. Terrain Reasoning for 3D Action Games. GDC 2001.