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

# Введение

# 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, применяемых в компьютерных играх

Рассмотрим методы и алгоритмы принятия решений, используемых в видеоиграх и других областях.

## Рулевое управление (steering behavior)

В [Reynolds] была предложена модель, описывающая поведение автономных персонажей и их взаимодействие друг с другом посредством вектором сил. Автономные персонажи – это агенты, использующиеся в анимации или в интерактивных системах, таких как видеоигры или симуляции. Эта модель предоставляет 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 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

## Нечеткая логика

{НИР за третий семестр}

## 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]

## Проверка статистических гипотез

{НИР за третий семестр}

## 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.

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.