

# Содержание

<b>Об авторе .....</b>	<b>8</b>
<b>О технических рецензентах .....</b>	<b>9</b>
<b>Предисловие .....</b>	<b>10</b>
<b>Глава 1. Интеллектуальные модели поведения:</b>	
<b>перемещение .....</b>	<b>16</b>
Введение .....	16
Создание шаблона моделей поведения .....	17
Преследование и уклонение.....	21
Достижение цели и уход от погони .....	23
Поворот объектов.....	26
Блуждание вокруг.....	29
Следование по маршруту .....	31
Уклонение от встреч с агентами.....	36
Уклонение от стен .....	39
Смешивание моделей поведения с весовыми коэффициентами .....	41
Смешивание моделей поведения по приоритету.....	43
Комбинирование моделей поведения с применением конвейера управления .....	45
Стрельба снарядами.....	49
Прогнозирование места падения снаряда .....	50
Нацеливание снаряда .....	52
Создание системы прыжков.....	53
<b>Глава 2. Навигация.....</b>	<b>60</b>
Введение.....	60
Представление игрового мира с помощью сетей .....	61
Представление игрового мира с помощью областей Дирихле.....	71
Представление игрового мира с помощью точек видимости .....	77
Представление игрового мира с помощью навигационного меша .....	81
Поиск выхода из лабиринта с помощью алгоритма DFS .....	84
Поиск кратчайшего пути в сети с помощью алгоритма BFS.....	86
Поиск кратчайшего пути с помощью алгоритма Дейкстры.....	88
Поиск оптимального пути с помощью алгоритма A* .....	91

Улучшенный алгоритм $A^*$ с меньшим использованием памяти – алгоритм IDA* .....	95
Планирование навигации на несколько кадров вперед: поиск с квантованием времени .....	98
Сглаживание маршрута .....	100
<b>Глава 3. Принятие решений .....</b>	<b>103</b>
Введение .....	103
Выбор с помощью дерева принятия решений .....	104
Работа конечного автомата .....	107
Усовершенствование конечного автомата: иерархические конечные автоматы .....	110
Комбинирование конечных автоматов и деревьев принятия решений .....	112
Реализация деревьев моделей поведения .....	113
Работа с нечеткой логикой .....	116
Представление состояний с помощью числовых значений: система Маркова .....	120
Принятие решений в моделях целенаправленного поведения .....	123
<b>Глава 4. Координирование и тактика .....</b>	<b>126</b>
Введение .....	126
Обработка формирований .....	127
Расширение алгоритма $A^*$ для координации: алгоритм $A^*mbush$ ....	132
Выбор удобных точек позиций .....	136
Анализ точек позиций по их высоте .....	138
Анализ точек позиций по обзорности и незаметности .....	140
Оценка точек позиций для принятия решения .....	142
Карты влияния .....	143
Улучшение карт влияния путем заполнения .....	147
Улучшение карт влияния с помощью фильтров свертки .....	152
Построение боевых кругов .....	155
<b>Глава 5. Органы чувств агентов .....</b>	<b>164</b>
Введение .....	164
Имитации зрения с применением коллайдера .....	165
Имитация слуха с применением коллайдера .....	167
Имитация обоняния с применением коллайдера .....	171
Имитации зрения с применением графа .....	175
Имитация слуха с применением графа .....	176
Имитация обоняния с применением графа .....	179
Реализация органов чувств в стелс-игре .....	181

<b>Глава 6. Настольные игры с искусственным интеллектом .....</b>	<b>189</b>
Введение .....	189
Класс игрового дерева .....	190
Введение в алгоритм Minimax .....	192
Алгоритм Negamax .....	194
Алгоритм AB Negamax .....	196
Алгоритм Negascouting .....	199
Подготовка .....	199
Реализация соперника для игры в крестики-нолики .....	201
Реализация соперника для игры в шашки .....	206
<b>Глава 7. Механизмы обучения .....</b>	<b>217</b>
Введение .....	217
Предугадывание действий с помощью алгоритма прогнозирования N-Gram .....	217
Усовершенствованный иерархический алгоритм N-Gram .....	220
Использование классификаторов Байеса .....	222
Использование деревьев принятия решений .....	225
Использование закрепления рефлекса .....	229
Обучение с помощью искусственных нейронных сетей .....	234
Создание непредсказуемых частиц с помощью алгоритма поиска гармонии .....	238
<b>Глава 8. Прочее .....</b>	<b>242</b>
Введение .....	242
Улучшенная обработка случайных чисел .....	242
Соперник для игры в воздушный хоккей .....	245
Соперник для настольного футбола .....	251
Программное создание лабиринтов .....	261
Реализация автопилота для автомобиля .....	264
Управление гонками с адаптивными ограничениями .....	265
<b>Предметный указатель .....</b>	<b>269</b>

# Об авторе

**Хорхе Паласиос** (Jorge Palacios) – профессиональный программист с более чем семилетним опытом. Последние четыре года занимался разработкой игр, работая на различных должностях, от разработчика инструментария до ведущего программиста. Специализируется на программировании искусственного интеллекта и игровой логики. В настоящее время использует в своей работе Unity и HTML5. Также является преподавателем разработки игр, лектором и организатором «геймджемов».

Больше узнать о нем можно на его личном сайте: <http://jorge.palacios.co>.

# О технических рецензентах

**Джек Донован** (Jack Donovan) – разработчик игр и инженер-программист, работающий с движком Unity3D начиная с третьей версии. Учился в колледже Шамплейн (г. Берлингтон, штат Вермонт), где получил степень бакалавра в области программирования игр.

В настоящее время работает над проектом виртуальной реальности IrisVR в Нью-Йорке, где занимается разработкой программного обеспечения, позволяющего архитекторам создавать модели виртуальной реальности по САД-моделям. До проекта IrisVR Джек работал в небольшой независимой студенческой команде, занимавшейся разработкой игр, написав при этом книгу «*OUYA Game Development By Example*».

**Лорен С. Ферро** (Lauren S. Ferro) – консультант по игрофикации (геймификации), дизайнер игр и схожих с играми приложений. Занимается проектированием, консультацией и разработкой стратегий для целого ряда проектов в области профессионального обучения, систем рекомендаций и общеобразовательных игр. Ведет активную исследовательскую работу в области геймификации, профилирования игроков и ориентированного на пользователей проектирования игр. Проводит семинары для специалистов и компаний, занимающихся разработкой игр и приложений с элементами игры, ориентированных на вкусы пользователей. Также является разработчиком методики Gamificards проектирования прототипов игр и приложений с элементами игры.

# Предисловие

Стоит подумать об искусственном интеллекте, и в уме возникает множество ассоциаций. От простых моделей поведения, например преследование или убегание от игрока, до игры с искусственным интеллектом в классические шахматы, методов машинного обучения или процедурной генерации контента.

Движок Unity сделал разработку игр намного демократичнее. Благодаря простоте использования, быстрому совершенствованию технологий, постоянно растущему сообществу и новым облачным услугам движок Unity превратился в один из важнейших программных продуктов для игровой индустрии.

С учетом вышесказанного основной целью написания книги была попытка помочь вам, читатель, через описание технологий Unity, знакомство с передовым опытом, изучение теории, разобраться в идеях и методах искусственного интеллекта, что обеспечит вам преимущества как в любительской, так и в профессиональной разработке.

Эта книга рецептов познакомит вас с инструментами создания искусственного интеллекта, например для реализации достойных противников, доведенных до совершенства, и даже для разработки собственного нестандартного движка искусственного интеллекта. Она станет вашим справочным пособием при разработке методов искусственного интеллекта в Unity.

Добро пожаловать в увлекательное путешествие, которое сочетает в себе ряд вещей, имеющих для меня, как профессионала и просто человека, большое значение: программирование, разработка игр, искусственный интеллект и обмен знаниями с другими разработчиками. Не могу не отметить, как я польщен и рад, что вы читаете написанный мной текст прямо сейчас, и я благодарен команде издательства Packt за предоставление такой возможности. Надеюсь, что этот материал поможет вам не только поднять на новый уровень навыки работы с Unity и искусственным интеллектом, но и содержит описание функций, которыми вы заинтересуете пользователей своих игр.

## О чем рассказывается в этой книге

*Глава 1 «Интеллектуальные модели поведения: перемещение»* описывает наиболее интересные алгоритмы перемещения, основанные на

принципах управления поведением, разработанных Крейгом Рейнольдсом (Craig Reynolds) совместно с Яном Миллингтоном (Ian Millington). Они стали основой большинства алгоритмов искусственного интеллекта в современных играх наряду с другими алгоритмами управления перемещением, такими как семейство алгоритмов определения маршрута.

*Глава 2 «Навигация»* описывает алгоритмы определения маршрута для сложных сценариев перемещения. Она рассматривает несколько способов представления игрового мира с помощью различных графовых структур и алгоритмы поиска путей в различных ситуациях.

*Глава 3 «Принятие решений»* демонстрирует различные методы принятия решений, достаточно гибкие, чтобы приспособливаться к разным видам игр, и достаточно надежные, чтобы позволить разрабатывать модульные системы принятия решений.

*Глава 4 «Координирование и тактика»* содержит ряд рецептов координации действий агентов, превращающих их в целостный организм, например в воинское подразделение, и методов принятия тактических решений на основании графов, таких как точки позиций и карты влияния.

*Глава 5 «Органы чувств агентов»* описывает различные подходы к моделированию мышления агентов. Для моделирования будут использованы уже известные нам инструменты, такие как коллаидеры и графы.

*Глава 6 «Настольные игры с искусственным интеллектом»* рассматривает семейство алгоритмов для разработки настольных игр с использованием технологий искусственного интеллекта.

*Глава 7 «Механизмы обучения»* посвящена области машинного обучения. Она послужит хорошим стартом для изучения и применения методов машинного обучения в играх.

*Глава 8 «Разное»* описывает применение новых методов и алгоритмов, рассматривавшихся в предыдущих главах, для создания моделей поведения, которые не вписываются ни в одну из перечисленных выше категорий.

## Что потребуется для работы с книгой

Примеры, представленные в книге, были протестированы с последней версией Unity (на момент завершения работы над книгой это была версия Unity 5.3.4f1). Однако, приступая к работе над книгой, я использовал Unity 5.1.2, поэтому ее можно считать минимальной рекомендуемой версией для опробования примеров.

## Кому адресована эта книга

Эта книга рассчитана на тех, кто уже знаком с Unity и хочет обзавестись инструментами для создания искусственного интеллекта и реализации игровой логики.

## Разделы

В этой книге вы найдете несколько часто встречающихся заголовков («Подготовка», «Как это реализовать», «Как это работает», «Дополнительная информация» и «Полезные ссылки»).

Разделы, начинающиеся с этих заголовков, содержат пояснения к реализации рецептов и используются следующим образом:

### **Подготовка**

Объясняет назначение рецепта и описывает настройку программного обеспечения и другие подготовительные действия, необходимые для осуществления рецепта.

### **Как это реализовать...**

Описывает шаги по реализации рецепта.

### **Как это работает...**

Обычно включает подробное объяснение того, что произошло в предыдущем разделе.

### **Дополнительная информация...**

Содержит дополнительную информацию о рецепте для лучшего понимания читателем.

### **Полезные ссылки**

Содержит ссылки на полезные материалы, содержащие информацию, связанную с рецептом.

## Соглашения

В этой книге используется несколько разных стилей оформления текста для выделения разных видов информации. Ниже приводятся примеры этих стилей и объясняется их назначение.



Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, фиктивные адреса URL, пользовательский ввод и ссылки в Twitter будут выглядеть так: «AgentBehaviour – это шаблонный класс для определения большинства моделей поведения в главе».

Блоки программного кода оформляются так:

```
using UnityEngine;
using System.Collections;
public class Steering
{
    public float angular;
    public Vector3 linear;
    public Steering ()
    {
        angular = 0.0f;
        linear = new Vector3();
    }
}
```

Когда потребуется привлечь ваше внимание к определенному фрагменту кода, соответствующие строки или элементы будут выделены жирным:

```
using UnityEngine;
using System.Collections;

public class Wander : Face
{
    public float offset;
    public float radius;
    public float rate;
}
```



Так оформляются предупреждения и важные примечания.



Так оформляются советы и рекомендации.

## Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте [www.dmkpress.com](http://www.dmkpress.com), зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com), при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и если вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу: [http://dmkpress.com/authors/publish\\_book/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу: [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

## Поддержка пользователей

Покупателям наших книг мы готовы предоставить дополнительные услуги, чтобы ваша покупка принесла вам максимальную пользу.

### Загрузка исходного кода примеров

Загрузить файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте [www.dmkpress.com](http://www.dmkpress.com) или [www.дмк.рф](http://www.дмк.рф) в разделе «Читателям – Файлы к книгам».

### Загрузка цветных иллюстраций к книге

Вы также можете получить файл в формате PDF с цветными иллюстрациями и диаграммами к этой книге. Цветные изображения помогут лучше понять содержание книги. Загрузить этот файл можно по адресу: [http://www.packtpub.com/sites/default/files/downloads/Unity5xGameAIProgrammingCookbook\\_ColorImages.pdf](http://www.packtpub.com/sites/default/files/downloads/Unity5xGameAIProgrammingCookbook_ColorImages.pdf).

### Список опечаток

Хотя мы приняли все возможные меры, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы нашли опечатку, пожалуйста, сообщите о ней главному редактору по адресу: [dmkpress@gmail.com](mailto:dmkpress@gmail.com), и мы исправим это в следующих тиражах.

## Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты [dmkpress@gmail.com](mailto:dmkpress@gmail.com) со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

## Вопросы

Вы можете присылать любые вопросы, касающиеся данной книги, по адресу [dm@dmk-press.ru](mailto:dm@dmk-press.ru) или [questions@packtpub.com](mailto:questions@packtpub.com). Мы постараемся решить возникшие проблемы.

# Глава 1

## Интеллектуальные МОДЕЛИ ПОВЕДЕНИЯ: перемещение

В этой главе рассматриваются алгоритмы искусственного интеллекта для перемещения и охватываются следующие рецепты:

- создание шаблона моделей поведения;
- преследование и уклонение;
- достижение цели и уход от погони;
- поворот объектов;
- блуждание вокруг;
- следование по маршруту;
- уклонение от встреч с агентами;
- уклонение от встреч со стенами;
- смешивание моделей поведения с применением весовых коэффициентов;
- смешивание моделей поведения с применением приоритетов;
- сочетание моделей поведения с применением конвейера управления;
- стрельба снарядами;
- прогнозирование эллипса рассеивания снарядов;
- нацеливание снаряда;
- создание системы прыжков.

### Введение

В настоящее время Unity является одним из самых популярных игровых движков, а также основным инструментом создания игр для индивидуальных разработчиков, не только благодаря доступной биз-

нес-модели, но и надежности редактора проектов, ежегодным улучшениям и, что самое главное, простоте использования и постоянно растущему сообществу разработчиков по всему миру.

Благодаря тому что движок Unity сам заботится о сложных закулисных процессах (отображение, физические процессы, интеграция и кросс-платформенное развертывание – лишь некоторые из них), разработчики могут сосредоточиться на создании систем искусственного интеллекта, оживляющих игры, обеспечивающих интерактивное взаимодействие в реальном времени.

Цель книги – познакомить вас с инструментами проектирования искусственного интеллекта, чтобы вы могли создавать достойных противников, доводя их до совершенства, и даже разработать собственный движок искусственного интеллекта.

Эта глава начинается с нескольких наиболее интересных алгоритмов, основанных на принципах управления перемещением, разработанных Крейгом Рейнольдсом (Craig Reynolds) совместно с Яном Миллингтоном (Ian Millington). На этих принципах базируется большинство алгоритмов искусственного интеллекта в современных играх наряду с другими алгоритмами управления перемещениями, такими как семейство алгоритмов определения маршрута.

## Создание шаблона моделей поведения

Перед созданием моделей поведения необходимо заложить программный фундамент не только для реализации интеллектуального перемещения, но и для модульной системы, позволяющий изменять и добавлять модели поведения. Здесь создаются пользовательские типы данных и базовые классы для большинства алгоритмов в этой главе.

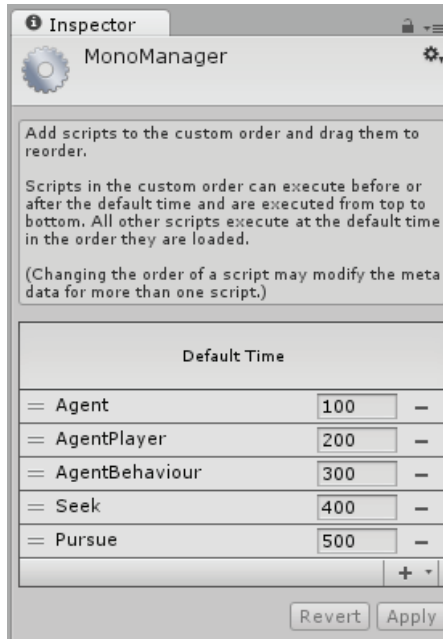
### Подготовка

Для начала вспомним порядок выполнения функций обновления:

- Update;
- LateUpdate.

Также вспомним о возможности управления порядком выполнения сценариев. В нашей модели поведения сценарии выполняются в следующем порядке:

- сценарии агентов;
- сценарии моделей поведения;
- модели поведения или сценарии, зависимые от предыдущих сценариев.



**Рис. 1.1** ❖ Пример определения порядка выполнения сценариев перемещения. Сценарий Pursue выполняется после сценария Seek, который выполняется после сценария AgentBehaviour

## Как это реализовать...

Нужно создать три класса: Steering, AgentBehaviour и Agent.

1. Класс Steering – пользовательский тип данных для хранения значений перемещения и поворота агента:

```
using UnityEngine;
using System.Collections;
public class Steering
{
    public float angular;
    public Vector3 linear;
    public Steering ()
    {
        angular = 0.0f;
        linear = new Vector3();
    }
}
```

2. Создадим класс `AgentBehaviour`, который станет шаблоном для большей части моделей поведения в этой главе:

```
using UnityEngine;
using System.Collections;
public class AgentBehaviour : MonoBehaviour
{
    public GameObject target;
    protected Agent agent;
    public virtual void Awake ()
    {
        agent = gameObject.GetComponent<Agent>();
    }
    public virtual void Update ()
    {
        agent.SetSteering(GetSteering());
    }
    public virtual Steering GetSteering ()
    {
        return new Steering();
    }
}
```

3. И наконец, класс `Agent` является основным компонентом, ответственным за реализацию моделей интеллектуального перемещения. Создадим файл с его скелетом:

```
using UnityEngine;
using System.Collections;
public class Agent : MonoBehaviour
{
    public float maxSpeed;
    public float maxAccel;
    public float orientation;
    public float rotation;
    public Vector3 velocity;
    protected Steering steering;
    void Start ()
    {
        velocity = Vector3.zero;
        steering = new Steering();
    }
    public void SetSteering (Steering steering)
    {
        this.steering = steering;
    }
}
```

4. Далее напишем функцию Update, обрабатывающую перемещение в соответствии с текущим значением:

```
public virtual void Update ()
{
    Vector3 displacement = velocity * Time.deltaTime;
    orientation += rotation * Time.deltaTime;
    // необходимо ограничить значение переменной orientation
    // диапазоном (0-360)
    if (orientation < 0.0f)
        orientation += 360.0f;
    else if (orientation > 360.0f)
        orientation -= 360.0f;
    transform.Translate(displacement, Space.World);
    transform.rotation = new Quaternion();
    transform.Rotate(Vector3.up, orientation);
}
```

5. В заключение реализуем функцию LateUpdate, которая подготовит управляющие воздействия для следующего кадра на основании текущего кадра:

```
public virtual void LateUpdate ()
{
    velocity += steering.linear * Time.deltaTime;
    rotation += steering.angular * Time.deltaTime;
    if (velocity.magnitude > maxSpeed)
    {
        velocity.Normalize();
        velocity = velocity * maxSpeed;
    }
    if (steering.angular == 0.0f)
    {
        rotation = 0.0f;
    }
    if (steering.linear.sqrMagnitude == 0.0f)
    {
        velocity = Vector3.zero;
    }
    steering = new Steering();
}
```

## Как это работает...

Идея заключается в том, чтобы поместить логику перемещения в функцию GetSteering() модели поведения, которая будет реализо-



вана позже, для упрощения класса агента, предназначенного только для основных расчетов.

Кроме того, это гарантирует установку управляющих значений агента до его использования благодаря настроенному порядку выполнения сценариев и функций в Unity.

## Дополнительная информация...

Этот подход на основе компонентов означает, что для обеспечения нужными моделями поведения к объекту класса `GameObject` требуется присоединить сценарий `Agent`.

## Полезные ссылки

Дополнительную информацию об игровом цикле в Unity и порядке выполнения функций и сценариев можно найти в официальной документации:

- <http://docs.unity3d.com/Manual/ExecutionOrder.html>;
- <http://docs.unity3d.com/Manual/class-ScriptExecution.html>.

# Преследование и уклонение

Модели преследования и уклонения отлично подходят для реализации в первую очередь, потому что опираются только на самые основные модели поведения и расширяют их возможностью предсказания следующего шага к цели.

## Подготовка

Нам потребуются две базовые модели поведения: `Seek` и `Flee`. Поместим их в порядке выполнения сценариев сразу после класса `Agent`.

Следующий код реализует модель `Seek`:

```
using UnityEngine;
using System.Collections;
public class Seek : AgentBehaviour
{
    public override Steering GetSteering()
    {
        Steering steering = new Steering();
        steering.linear = target.transform.position - transform.position;
        steering.linear.Normalize();
        steering.linear = steering.linear * agent.maxAccel;
```

```
        return steering;
    }
}
```

и модель Flee:

```
using UnityEngine;
using System.Collections;
public class Flee : AgentBehaviour
{
    public override Steering GetSteering()
    {
        Steering steering = new Steering();
        steering.linear = transform.position - target.transform.position;
        steering.linear.Normalize();
        steering.linear = steering.linear * agent.maxAccel;
        return steering;
    }
}
```

## Как это реализовать...

Классы Pursue и Evade, по сути, реализуют один и тот же алгоритм, отличаясь только разными базовыми классами, которые они наследуют:

1. Создадим класс Pursue, наследующий класс Seek, и добавим атрибуты для прогнозирования:

```
using UnityEngine;
using System.Collections;

public class Pursue : Seek
{
    public float maxPrediction;
    private GameObject targetAux;
    private Agent targetAgent;
}
```

2. Реализуем функцию Awake для настройки всего, что связано с реальной целью:

```
public override void Awake()
{
    base.Awake();
    targetAgent = target.GetComponent<Agent>();
    targetAux = target;
    target = new GameObject();
}
```

3. А также реализуем функцию `OnDestroy` для корректной обработки внутреннего объекта:

```
void OnDestroy ()
{
    Destroy(targetAux);
}
```

4. Наконец, реализуем функцию `GetSteering`:

```
public override Steering GetSteering()
{
    Vector3 direction = targetAux.transform.position -
                        transform.position;
    float distance = direction.magnitude;
    float speed = agent.velocity.magnitude;
    float prediction;
    if (speed <= distance / maxPrediction)
        prediction = maxPrediction;
    else
        prediction = distance / speed;
    target.transform.position = targetAux.transform.position;
    target.transform.position += targetAgent.velocity * prediction;
    return base.GetSteering();
}
```

5. Реализация модели поведения `Evade` выглядит точно так же, с той лишь разницей, что она наследует класс `Flee`:

```
public class Evade : Flee
{
    // все то же самое
}
```

## Как это работает...

Эти модели поведения базируются на моделях `Seek` и `Flee`, но вдобавок учитывают скорость цели, чтобы предсказать, куда двигаться дальше. Координаты целевой точки при этом сохраняются в дополнительном внутреннем объекте.

## Достижение цели и уход от погони

В основу этих моделей заложены те же принципы, что и для моделей `Seek` и `Flee`, плюс дополнительно определяется точка, в которой агент

может автоматически остановиться после выполнения определенного условия, либо при приближении к пункту назначения (достижении цели), либо при достаточном удалении от опасной точки (уход от погони).

## Подготовка

Для алгоритмов Arrive и Leave необходимо создать отдельные файлы и не забыть настроить порядок их выполнения.

## Как это реализовать...

При реализации этих моделей поведения используется один и тот же подход, но они содержат разные свойства и производят разные расчеты в начальной части функции GetSteering:

1. Во-первых, модель Arrive должна определять радиус остановки (достижение цели) и радиус замедления скорости:

```
using UnityEngine;
using System.Collections;

public class Arrive : AgentBehaviour
{
    public float targetRadius;
    public float slowRadius;
    public float timeToTarget = 0.1f;
}
```

2. Создадим функцию GetSteering:

```
public override Steering GetSteering()
{
    // дальнейшая реализация описывается ниже
}
```

3. Первая половина функции GetSteering вычисляет скорость в зависимости от расстояния до цели и радиуса замедления:

```
Steering steering = new Steering();
Vector3 direction = target.transform.position - transform.position;
float distance = direction.magnitude; float targetSpeed;
if (distance < targetRadius)
    return steering;
if (distance > slowRadius)
    targetSpeed = agent.maxSpeed;
else
    targetSpeed = agent.maxSpeed * distance / slowRadius;
```

Конец ознакомительного фрагмента.  
Приобрести книгу можно  
в интернет-магазине  
«Электронный универс»  
[e-Univers.ru](http://e-Univers.ru)