Разработка шаблона поведения игрового бота с элементами искусственного интеллекта на основе цепей Маркова

О.В. Игнатьева, О.Г. Ведерникова

Ростовский государственный университет путей сообщения

Аннотация: Предметом данной статьи является разработка шаблона поведения с элементами ИИ для бота-противника в однопользовательской игре Steal Tower, суть которой - собрать ресурсы для постройки башни быстрее противников. Для создания иллюзии того, что против игрока играют такие же люди, разработана имитационная стохастическая модель на основе метода Монте Карло для цепей Маркова. На основе результатов ее испытаний определены сбалансированные параметры системы, которые внедрены в поведенческий шаблон бота, реализованный с помощью перечисления Enum AIStates, состоящего из пяти состояний: Idle (бездействие), GoTo (движение) и GoToWarehouse (возврат к складу), Win (победа), Loose (подсчет очков). В каждом из них разработаны функции для оптимального поведения бота, приведенные в статье. Так, для состояния GoTo, созданы функции, анализирующие выгоду от разного типа поведения: украсть или собрать, дойти до склада или до башни.
Ключевые слова: игровой интеллект, поведенческий шаблон, эмуляция живого мира, сценарий поведения бота, структура состояний, цепи Маркова, метод Монте Карло, имитационная модель, среда Unity, язык C#.

Введение. Разработка компьютерных игр с элементами искусственного интеллекта – это быстро развивающееся направление информационных технологий. Несмотря на кажущуюся несерьезность, гейм-индустрия превратилась в многомиллиардный рынок, который сопоставим по объему с бюджетом индустриальных компаний и уже сравним с ВВП небольшой европейской страны.

Проблема разработки игрового интеллекта (ИИ) для компьютерных игр возникает на этапе подготовки сценариев поведения для виртуальных противников игрока. ИИ в разных компьютерных играх имеет свою специфику. Например, ИИ для игры в шахматы имеет сложную структуру и мощный математический аппарат, но он не сможет поддержать беседу с человеком, в отличие от чат-бота, у которого ИИ простой структуры с облегченными алгоритмами [1].

Игровой интеллект или шаблон поведения бота в компьютерных играх, как правило, это набор программных методик, определяющих, насколько

натурально реагируют боты на действия игрока, как ведут себя противники при встрече с игроком, и на то, как глубоко сможет погрузиться человек в игру. В играх может использоваться ИИ с машинным обучением, при котором противники могут обучаться не только на своих ошибках, но и на ошибках игрока [2].

Цель разработки. Предметом данной статьи является разработка шаблона поведения противника в однопользовательской игре Steal Tower (Украсть башню). Основной персонаж в игре Steal Tower, разработанной студией, перемещается по игровому полю, собирая ресурсы или материалы в склады для постройки башни (рис. 1).

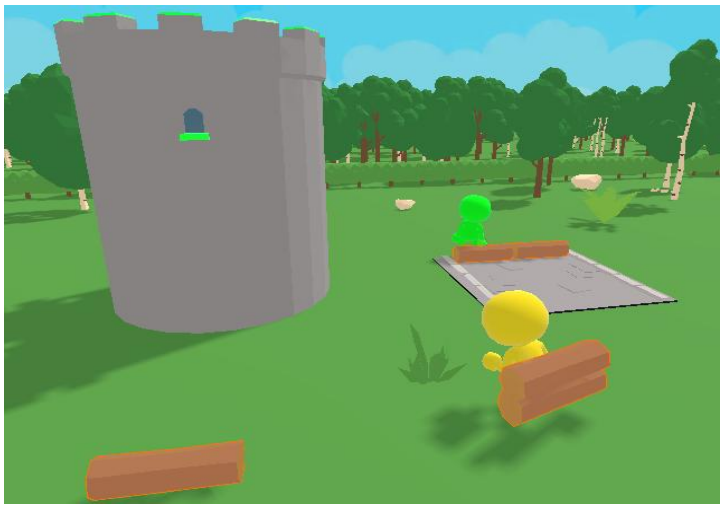

Рис. 1. Вид игрового процесса игры Steal Tower

Игроку мешают враги-боты, обладающие игровым интеллектом, которые, как и игрок могут собирать ресурсы, а также воровать их из склада игрока на свою базу. Для победы в игре персонаж должен успеть построить башню быстрее врагов, тогда происходит переход на новый уровень, на котором строиться более сложная башня.

Во время проектирования игры Steal Tower возникла необходимость создания шаблона поведения и передвижения противников по игровой поверхности с добавлением соревновательного элемента, который обеспечил бы возможность соревноваться в сборе ресурсов с игроком. Задача данного

игрового интеллекта состоит не в том, чтобы научить бота с помощью нейронной сети обыгрывать игроков, находить оптимально выигрышную стратегию, а имитировать реалистичное человекоподобное поведение, чтобы дать возможность выиграть пользователю, стимулировать интерес и азарт у игрока [3, 4]. Исходя из этого требования, было выделено пять состояний бота-противника: GoTo (движение), Idle (бездействие), GoToWarehouse (возврат к складу), Win (победа), Loose (подсчет очков). Второе состояние (Idle), обозначающее бездействие, создано для замедления бота-противника, для повышения вероятности выигрыша игрока. Переходы между этими состояниями происходят по следующему алгоритму. Из состояния S1 (GoTo) бот может перейти в состояние S2 (Idle) в 35% случаев, переходит в состояние S3 (GoToWarehouse) если инвентарь заполнен и его нужно выгрузить на складе (количество собранных бревен равно три), в состояние S5 (Loose) – если кто-то из соперников успел собрать башню и набрать больше очков, остаётся в этом же состоянии S1 (GoTo) пока бот строит кратчайший маршрут к ресурсам и собирает бревна. Из состояния S2 (Idle) бот может перейти в состояние S1(GoTo) если количество бревен в инвентаре меньше 3, в состояние S3 (GoToWarehouse) – если склад ближе, чем ближайшие ресурсы, остаётся в этом же состоянии S2 (Idle) около 1 секунды для задержки времени. Из состояния S3 (GoToWarehouse) бот может перейти в состояние S2 (Idle) в 35% случаев, остаётся в этом же состоянии S3 около 2 секунд пока разгружает бревна на складе, переходит в состояние S4(Win) если склад полон у бота раньше, чем у соперников, в состояние S5 (Loose) – если кто-то из соперников успел собрать башню и набрать больше очков.

Постановка задачи заключается в построении математической модели поведения бота, исходя из описания, приведенного выше и дальнейшей гармонизации параметров модели поведения, таких, как вероятности переходов между состояниями. В дальнейшем, в соответствии с построенной

математической моделью, нужно разработать программные алгоритмы, внедренные в игру Steal Tower. Гармонизация параметров необходима для поддержания азарта у игрока [4, 5]. Для этого необходимо произвести программную реализацию стратегии по математической модели методом Монте-Карло [6]. Для такого стохастического моделирования была разработана цепь Маркова, реализующая переходы системы между множеством состояний $S=\{s_1, s_2, s_3, s_4, s_5\}$. Граф цепи Маркова представлен на рисунке 2.

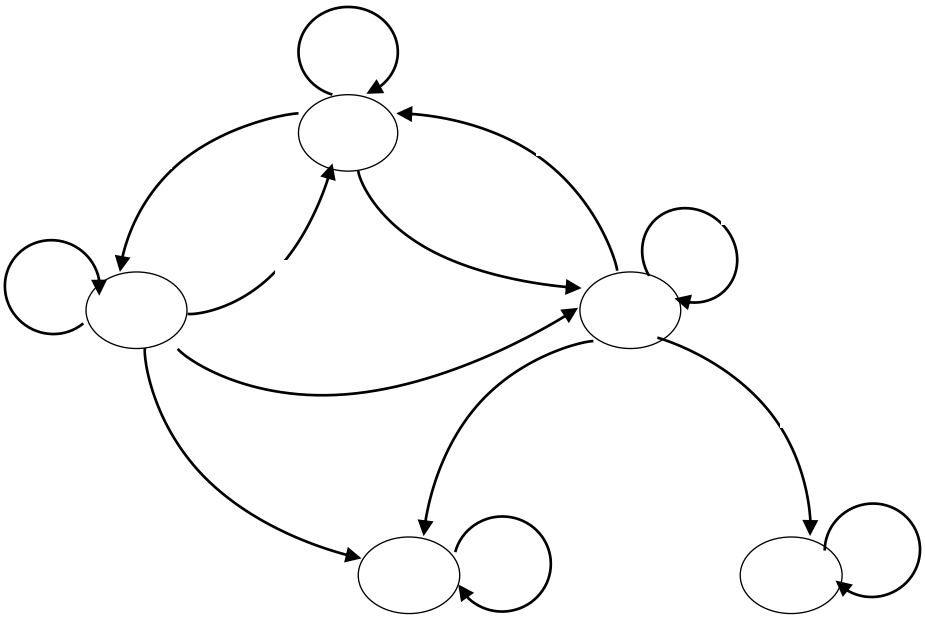

Рис.2. Цепь Маркова, моделирующая поведение бота в виде графа

Как видно из графа, поглощающими состояниями являются S4(Win) и S5 (Loose).

Матрица вероятностей переходов $P=\{p_{ij}\}$, где $\sum_{j=1}^{n} p_{ij} = 1$

$$P = \begin{pmatrix} p_{11} & p_{12}=0{,}35 & p_{13} & 0 & p_{15} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} \\ p_{31} & p_{32}=0{,}35 & p_{33} & p_{34} & p_{35} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Вектор начальных вероятностей $q_0=\{q^0_1, q^0_2, \ldots q^0_n\}$, в нашем случае равен $q_0=\{0, 1, 0, 0, 0\}$, так как процесс начинается с состояния бездействия.

Рассматриваемый случайный процесс можно считать Марковским, так как он обладает свойством отсутствия последействия, то есть на любом шаге t_k вероятность любого состояния системы в будущем (при $t>t_k$) зависит только от ее состояния в настоящем (при $t=t_k$) и не зависит от предыстории процесса. Также поток обладает свойствами ординарности и стационарности с дискретным временем и дискретным пространством состояний. Исходя из этого, можно воспользоваться формулой:

$$q_t = q_0 \cdot P^t$$

где q_t вектор распределения вероятностей на шаге t, q_0 вектор начальных вероятностей, P матрица вероятностей переходов. Другими словами, для вычисления q_t распределения вероятностей на любом шаге t нужно умножить q_0 на матрицу переходов P возведенную в степень t [7].

Так, например, при матрице вероятностей переходов P, равной:

$$P = \begin{pmatrix} 0{,}2 & 0{,}35 & 0{,}35 & 0 & 0{,}1 \\ 0{,}35 & 0{,}4 & 0{,}25 & 0 & 0 \\ 0 & 0{,}35 & 0{,}3 & 0{,}2 & 0{,}15 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

и начальных вероятностях $q_0=\{0, 1, 0, 0, 0\}$, получаем, что после первого шага вектор распределения вероятностей будет $q_1=\{0{,}35; 0{,}4; 0{,}25; 0; 0\}$. На шестом шаге t=6 вектор распределения вероятностей будет $q_6=\{0.109518;0.206168; 0.160801; 0.241591; 0.281922\}$

В некоторых случаях процесс может стабилизироваться, согласно эргодической теореме Хинчина, стационарные вероятности при $t\to\infty$ сходятся к стационарному распределению цепи Маркова:

$$\lim_{t\to\infty} q_0 \cdot P^t = \hat{Q}$$

Представленная стохастическая модель поведения бота позволяет определить не только вероятности, с которыми бот окажется в состоянии s_i спустя некое количество шагов, но и через сколько шагов перейдет в поглощающее состояние Win или Loose, как скоро игра закончится, и как часто будет побеждать противник бота, то есть игрок [6, 7].

Для программной реализации математической модели методом Монте-Карло каждый такт генерировалось случайное число в интервале от 0 до 1. Если система на k-ом шаге находится в состоянии S_i, то интервал [0; 1] разбивается на отрезки длиной p_{ij}, j=1..n, то есть на отрезки [0; p_{i1}] , [p_{i1}; p_{i1}+p_{i2}], [p_{i1}+p_{i2}; p_{i1}+p_{i2}+p_{i3}], … , [$\sum_{r=1}^{j-1} p_{ir}$; $\sum_{r=1}^{j} p_{ir}$], … [$\sum_{r=1}^{n} p_{ir}$; 1]. Если случайное число попадает в [$\sum_{r=1}^{j-1} p_{ir}$; $\sum_{r=1}^{j} p_{ir}$], то происходит переход из S_i в состояние S_j. Такие действия повторяются в цикле до достижения поглощающего состояния S_4 или S_5 .

После программной реализации и проведенных испытаний стохастической модели была проведена гармонизация параметров p_{ij} для поддержания азарта у игрока.

Материалы и методы

Основная разработка функционала игры Steal Tower проводилась в среде Unity 3D. В Unity используется визуальная среда разработки и модульная система компонентов [8]. Написание скриптов в Unity производится на языке C#. Также Unity моделирует физику твёрдых тел, которая рассчитывается с помощью движка PhysX [9].

Для программной реализации поведения ботов с ИИ использовались компоненты NavMesh, NavMesh Agent и Off-mesh Link [8], описывающие все поверхности, которые находятся на игровом поле в Unity 3D. Обращаясь к

NavMesh, основному компоненту, описывающему поверхности, искусственный интеллект, может просчитывать свой маршрут и составлять самый короткий путь для бота [9]. До того, как персонаж прикоснется к NavMesh, на игровых локациях присутствуют трамплины, показанные на рисунке 3.

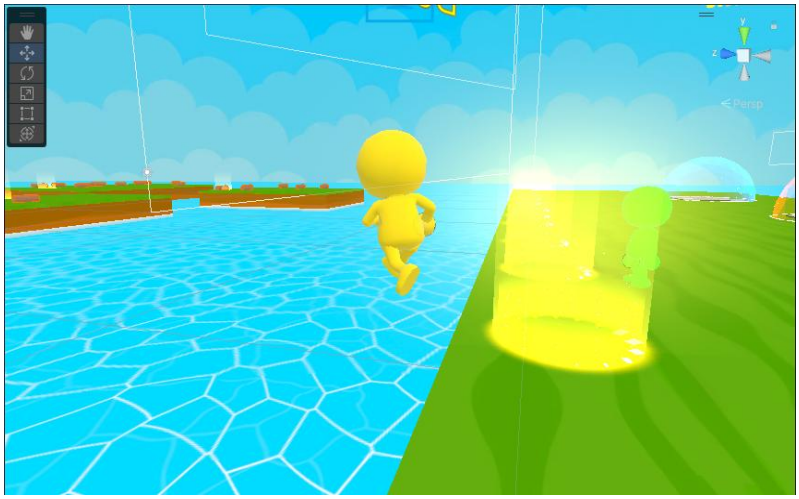

Рис. 3. Прыжок персонажа на трамплине NavMesh

С помощью трамплинов игрок может перепрыгивать препятствия, сокращая время перемещения между позициями. Однако, бот просто так не будет использовать трамплины, потому что не может просчитать, какой путь будет короче, прыгнуть или пробежать. Для решения этой проблемы в Unity есть компонент Off-mesh Link. Этот компонент необходим для создания перемещения по локации не только по горизонтали, но и по вертикали [8]. Перемещение через портал, подъем по лестнице или прыжок с трамплина – все это требует использования компонента Off-mesh Link.

Перейдем к компоненту NavMesh Agent, этот компонент используется непосредственно на игровых персонажах, которые должны перемещаться по игровому полю. Как правило, таких персонажей с компонентом NavMesh Agent вкратце называют агентами.

Также, для создания иллюзии того, что против игрока играют такие же люди, используется поведенческий шаблон, разработанный на основе цепей

Маркова, который в коде реализован с помощью перечисления Enum с идентификатором AIStates описание, которого представлено на рисунке 4.

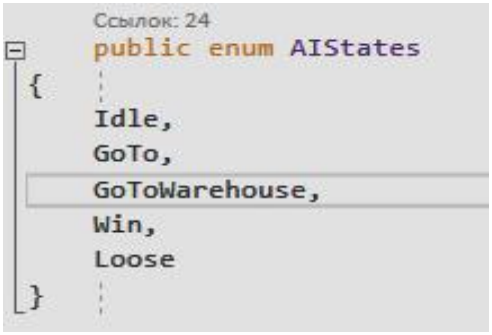

Рис. 4. Описание перечисления AIStates для состояния противников

Противник имеет пять состояний Idle, GoTo, GoToWarehouse, Win, Loose как и в цепи Маркова. Два последних из них - Win и Loose воспроизводятся только тогда, когда срабатывают определенные триггеры. Для Win это победа игрока, для Loose это получение определенного количества очков.

В состояние Idle (бездействие) бот переключается случайно с определенной вероятностью 0,35, которая была установлена исходя из проведенных испытаний стохастической модели. Настройка вероятностей для перехода в состояние бездействия Idle представлено на рисунке 5.

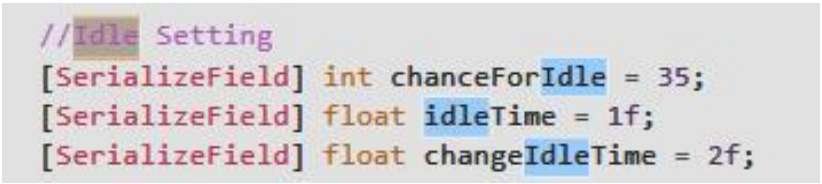

Рис. 5. Настройка вероятностей перехода в состояние Idle

Также противник-бот переходит в состояние Idle, если выполняет какое-либо из действий, описание которых приведено на рисунке 6, в соответствии с математической моделью цепи Маркова.

Искусственно замедлять противника необходимо, потому что искусственный интеллект уже знает местонахождение всех предметов для сбора. Без данного ограничения ИИ будет молниеносно реагировать на

любое изменение игровой локации, так, как бы не мог реагировать игрок [10, 11]. Именно для создания иллюзии состязания с реальными людьми создано данное состояние.

```
private void OnTriggerEnter(Collider other)
{
    GameObject entity = other.gameObject;
    if (entity.tag == Tags.materialObj)
    {
        if (aiController != null)
        {
            if (materialCount != 3)
            {
                aiController.IdleTime();
            }

        }
```

Рис. 6. Скрипт для переключения бота в состояние Idle

Второе состояние противника-бота, которое выполняется большую часть времени – это состояние GoTo. Благодаря ему противник передвигается по игровому полю, выстраивает маршрут к объекту для собирания ресурсов, а также вычисляет какой объект из множества на данный момент находится ближе всего.

В игре присутствует несколько, а именно четыре объекта, с которыми игрок и противник может взаимодействовать. Это вражеские базы или склады Warehouse, представленные на рисунке 7.

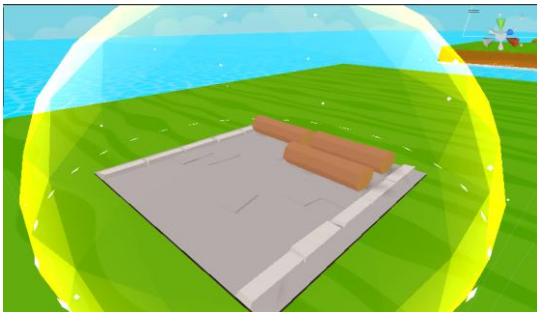

Рис. 7. Изображение базы или склада (Warehouse)

Из складов боты и игрок могут украсть очки у оппонента. Объекты, которые приносят очки – это ресурсы или материалы Material, выполненные

в виде бревен (рис. 8), ускорители Booster, которые на некоторое время увеличивают скорость передвижения.

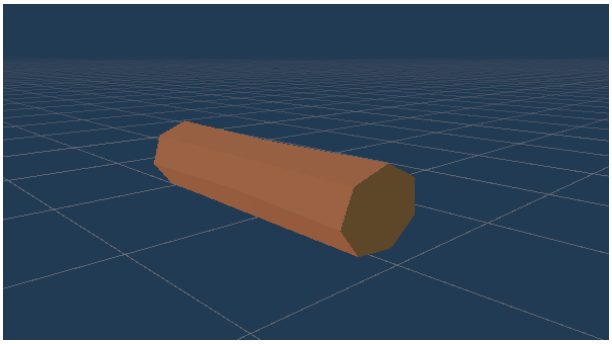

Рис. 8. Изображение ресурса Material

А также трамплины Tramplin, с помощью которых можно преодолевать препятствия.

Во время состояния GoTo игровой интеллект анализирует, что лучше будет сделать, украсть материалы у противника или же будет быстрее просто пройти и собрать материалы, которые разбросаны по полю. Бот может анализировать это, благодаря функции Vector3.Distance Unity.

Таким образом, можно определить какое расстояние на данный момент между ботом и другими объектами на локации. Но для того, чтоб их сравнить, нужно все полученные расстояния занести в буферные переменные и в отдельной функции произвести сравнение, то есть найти минимальное. В результате работы этой функции будет понятно, какой из объектов находится ближе всего к боту. Когда найдена минимальная дистанция к объекту, тогда применяется функция SetDestination, которая взаимодействует с NavMesh Agent. Обращаясь к агенту, функция SetDestination передает объект GameObject, к которому следует идти и выстроить маршрут (рис. 9).

Третье состояние противника-бота – это состояние GoToWarehouse. Бот переходит в него, когда собирает определенное количество ресурсов и ему нужно вернуться на свою базу, для освобождения инвентаря – пространства на спине для переноса материалов (рис. 10).

```
void SetDestination(GameObject destination)
{
    if (destination == null)
    {
        state = AIStates.Idle;
        return;
    }
    if (gameObject.GetComponent<NavMeshAgent>().enabled == true)
    {
        agent.SetDestination(destination.transform.position);

    }
}
```

Рис. 9. Функция SetDestination для выстраивания маршрута к объекту

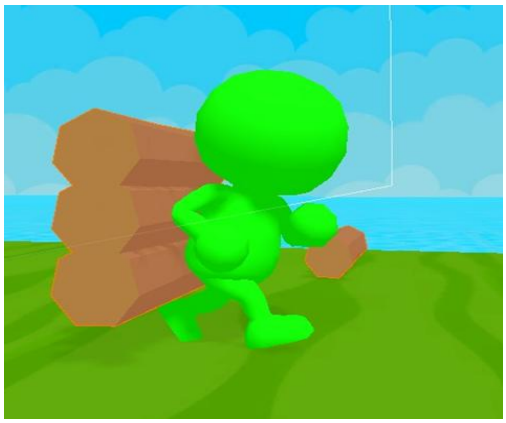

Рис. 10. Слоты инвентаря бота, заполненные ресурсами

Это происходит только тогда, когда слоты инвентаря полностью заполнены ресурсами-бревнами. Так как боту всегда известно, где находится его база, то возвращение происходит довольно просто. Можно использовать всего одну функцию, которая уже встроена в NavMeshAgent. Вызывая встроенную функцию SetDestination, бот отправляется к собственной базе Warehouse (рис. 11).

```
ссылка: 1
void GoToWarehouse(GameObject warehosue)
{
    agent.SetDestination(warehosue.transform.position);
}
```

Рис. 11. Функция для выстраивания маршрута к базе

Заключение

Разработан сценарий поведения противника с использованием игрового интеллекта в однопользовательской игре Steal Tower на основе цепей Маркова. Рассчитаны вероятности, с которыми бот окажется в состоянии s_i спустя заданное количество шагов, через какое количество шагов перейдет в поглощающее состояние. Определены сбалансированные параметры системы для эмуляции естественного поведения бота в целях поддержания интереса у игрока [12], исходя из экспериментальных испытаний имитационной модели на основе метода Монте-Карло для разработанной цепи Маркова. ИИ реализован в виде набора программных методик, которые кроме вычисления кратчайшего маршрута, выполняют искусственное замедление противника путем вероятностного перехода в бездействие, повторяют действия аналогичные действиям игрока со сбалансированными вероятностями переходов. Благодаря разработанному ИИ игра стала более реалистичной и захватывающей. Данный игровой интеллект имеет широкий функционал дополнительных настроек и полностью адаптивен к расширению функционала, так как для его написания была использована структура состояний.

Развитие игры может идти в направлении приобретения жанрового аспекта RPG игр в виде классов «вор», «силач». «Вор» не может подобрать с поля больше одного бревна, но может украсть из вражеского склада более трех. «Силач» сможет собирать большое количество материалов, но способности воровать, у него нет. Дальнейшее улучшение игры возможно также в направлении добавления функций для иллюзии живого мира в игре, в сфере разнообразия типов башен.

Литература

1. Chernov A.V., Butakova M.A., Guda A.N., Vereskun V.D., Kartashov O.O.Knowledge representation method for intelligent situation awareness system

design // Advances in Intelligent Systems and Computing. 2019. V. 875. pp. 225-235.

2. Игнатьева О.В., Корниенко И.А. Автоматизация процесса авторизации с помощью технологий искусственного интеллекта // Сборник научных трудов «Транспорт: наука, образование, производство». Рост.гос.ун-т путей сообщения. Ростов н/Д, 2023. С. 81-85.

3. Куликова Я.В., Качалов Д.Л. Метод определения эмоционального состояния человека при помощи чат-бота // Инженерный вестник Дона. 2022. № 9. URL: ivdon.ru/ru/magazine/archive/%20n9y2022/7893.

4. Schreiber Ian, Romero Brenda. Game Balance. // CRC Press Taylor & Francis Group, 2021. p.886.

5. McLaughlin T., Cutler L., Coleman D. Character rigging, deformations, and simulations in film and game production // ACM SIGGRAPH 2011 Courses. – SIGGRAPH'11. – New York, NY, USA: ACM, 2011.– Pp. 5:1–5:18.

6. Зятева О.А., Питухин Е.А., Воронов Р.В., Щеголева Л.В. Алгоритм непараметрической оценки плотности распределения вероятностей переходов для марковской модели динамики численности сотрудников вуза // Инженерный вестник Дона. 2023. №8. URL: ivdon.ru/ru/magazine/archive/n8y2023/8624.

7. Баркалов С.А., Моисеев С.И., Серебрякова Е.А. Модель управления запасами в строительной сфере, основанная на марковских случайных процессах // Инженерный вестник Дона. 2023. №2. URL: ivdon.ru/ru/magazine/archive/n2y2023/8216.

8. Sung K., Gregory S. Basic Math for Game Development with Unity 3D. - New York: Springer Science + Business Media, 2019. 414 p.

9. Best 3D Animation Software. 2022. URL: youtube.com/watch?v=EMfdtog1NGY (Accessed: 05.05.2022).

10. Ведерникова О.Г., Москат Н.А. Расширение и использование

редактора визуального программирования для разработки виртуальных тренажеров // Инженерный вестник Дона. 2021. № 1. URL: ivdon.ru/ru/magazine/archive/n1y2021/6783.

11. Аникеев, А.С. Генерация и моделирование 2D ландшафта по контрольным значениям с использованием Unity на языке программирования // Инженерный вестник Дона. 2020. №4. URL: ivdon.ru/ru/magazine/archive/N4y2020/6433.

12. Москат Н.А, Ведерникова О. Г. Разработка самообучающегося бота-собеседника с возможностью расчета арифметического выражения // Инженерный вестник Дона. 2023. №2. URL: ivdon.ru/ru/magazine/archive/n2y2023/8216.

References

1. Chernov A.V., Butakova M.A., Guda A.N., Vereskun V.D., Kartashov O.O. Advances in Intelligent Systems and Computing. 2019. V. 875. pp. 225-235.

2. Ignatieva O.V., Kornienko I.A. Sbornik nauchnykh trudov «Transport: nauka, obrazovanie, proizvodstvo». Rost.gos.un-t putey soobshcheniya. Rostov n/D. 2023. pp. 81-85.

3. Kulikova Ya.V., Kachalov D.L. Inzhenernyj vestnik Dona. 2022. №9. URL: ivdon.ru/ru/magazine/archive/%20n9y2022/7893.

4. Schreiber Ian, Romero Brenda. Game Balance. CRC Press Taylor & Francis Group. 2021. p.886.

5. McLaughlin T., Cutler L., Coleman D. Character rigging, deformations, and simulations in film and game production. ACM SIGGRAPH 2011 Courses. SIGGRAPH'11. New York, NY, USA: ACM, 2011. Pp. 5:1–5:18.

6. Zyateva O.A., Pitukhin E.A. Inzhenernyj vestnik Dona. 2023. №8. URL: ivdon.ru/ru/magazine/archive/n8y2023/8624.

7. Barkalov S.A., Moiseev S.I., Serebryakova E.A. Inzhenernyj vestnik

Dona. 2023. №2. URL: ivdon.ru/ru/magazine/archive/n2y2023/8216.

8. Sung K., Gregory S. Basic Math for Game Development with Unity 3D. New York: Springer Science + Business Media. 2019. 414 p.

9. Best 3D Animation Software. 2022. URL: youtube.com/watch?v=EMfdtog1NGY (Accessed: 05.05.2022).

10. Vedernikova O.G, Moskat N.A. Inzhenernyj vestnik Dona. 2021. №1. URL: ivdon.ru/ru/magazine/archive/n1y2021/6783.

11. Anikeev, A.S. Inzhenernyj vestnik Dona 2020. №4. URL: ivdon.ru/ru/magazine/archive/N4y2020/6433.

12. Moskat N.A., Vedernikova O.G. Inzhenernyj vestnik Dona. 2023. №2. URL: ivdon.ru/ru/magazine/archive/n2y2023/8216.

Дата поступления: 11.02.2024

Дата публикации: 26.03.2024