КОМПЬЮТЕРНЫЕ НАУКИ И ИНФОРМАТИКА

ВОЗМОЖНОСТИ ЭВОЛЮЦИОННЫХ МЕТОДОВ ДЛЯ ОРГАНИЗАЦИИ СТРУКТУРЫ ИГРОВОГО МИРА

САВКИНА Анастасия Васильевна
кандидат технических наук, доцент
ФИЛИППОВ Алексей Сергеевич
магистрант
ФГБОУ ВО «Мордовский государственный университет им. Н.П. Огарева»
г. Саранск, Россия

Требования к реалистичности игрового мира является актуальной проблемой организации не только персонажей, но и структуры игрового мира с учетом обеспечения наибольшей приспособленности и правильного поведения персонажа в условиях окружающей среды.
Ключевые слова: NPC, геном, игровой объект, классы, структура.

Для описания организации поведения NPC (Non-player character – «персонаж не-игрока», т. е. «управляемый не игроком»), когда его поведение определяется программно, важную роль играет его взаимодействие с игровым миром. В компьютерных и настольных ролевых играх термином «NPC» обозначаются персонажи, общающиеся с игроком, независимо от их отношения к игровому персонажу. Такие персонажи служат важным средством создания игровой атмосферы, мотивируют игроков совершать те или иные действия и являются основным источником информации об игровом мире и сюжете игры.

Поскольку для исследования возможности организации поведения был выбран клеточный генетический алгоритм [1], то основное игровое поле представляет собой двумерную сетку с квадратными ячейками, каждая из которых находится в том или ином состоянии. В данной реализации каждая клетка может находиться в следующих состояниях воздух, трава, живое существо.

Таким образом, по своей сути игровое поле является двумерным массивом, что позволяет удобно работать с ним с программной точки зрения, так как координаты клетки будут в точности совпадать с индексами массива. При этом нам необходимо хранить и обрабатывать не только состояние клетки, но и определять конкретного NPC или иной игровой объект, расположенный на ней. Поэтому была создана структура данных игрового поля CellState.

```
public struct CellState
{
   public string Status;
   public string Hash;
}
```

В данной структуре в параметре Status хранится информация о состоянии клетки, а в параметре Hash уникальный идентификатор NPC или травы. Сами же экземпляры игровых объектов хранятся в хеш-таблицах, где ключом является Hash. Это позволяет быстро получать доступ к объектам без необходимости перебора всех элементов массива игрового мира. В свою очередь такой подход позволяет обновлять информацию только в тех элементах массива, которые были изменены за единицу игрового времени, что также избавляет от необходимости его полного перебора. Все это положительно сказывается на быстродействии.

Учитывая тот факт, что каждую единицу игрового времени необходимо каждому персонажу анализировать окружающую его обстановку необходимо обеспечить им доступ к хранящейся в массиве и хеш-таблицах информации. В языке программирования C#, который используется в Unity, массивы и

хеш-таблицы передаются по ссылке, а не по значению. Таким образом, это дает возможность передать их в каждый экземпляр класса NPC без накладных расходов на оперативную память. Отслеживание игровой логики как правило происходит в событиях Update, FixedUpdate и LateUpdate.

Каждый игровой объект, расположенный на сцене, является экземпляром своего класса объектов, который в свою очередь является наследником класса MonoBehaviour и, следовательно, имеет все функции событий, представленные на рисунке. Таким образом, каждый игровой объект способен обрабатывать события, происходящие с ним, что избавляет от написания единой функции обработки игрового мира, позволяя создавать лишь игрового менеджера, который будет отслеживать только глобальные изменения.

В результате мы можем передать NPC ссылки на массив, содержащий информацию об игровом поле, а также хештаблицы и они смогут самостоятельно обрабатывать затрагивающие их события, в то время как игровой менеджер будет заниматься лишь инициализацией игрового мира, созданием и удалением персонажей и т. д.

Исходя из поставленной задачи необходимо обеспечить набольшую приспособленность NPC к условиям игрового мира. Приспособленность определяется параметрами игрового персонажа, однако в нашей ситуации необходимо обеспечить еще и правильную модель поведения [2]. Поэтому целесообразно связать с целевой функцией как параметры, так и склонности к принятию того или иного решения. Поскольку каждый из параметров персонажа задается числовым значением, будет оптимальным решением определить геном как массив целых чисел, где каждый из элементов будет отвечать за тот или иной параметр персонажа и склонности к решениям. Кроме того, это позволит компактно хранить информацию о широком разнообразии видовой популяции.

Вместе с тем необходимо абстрагировать геном от самого класса персонажа, а также обеспечить удобную работу с ним при создании персонажей и передаче информации о геноме. Это позволит создавать различных NPC на базе единого класса используя в качестве инициализатора геном. В результате чего было принято решение сделать отдельный класс для хранения генетической информации, с реализацией ряда служебных функций.

Реализация данного класса представлена ниже.

```
public class Genom
{
    public int[] DNK;

    public Genom(int TypeOfIndividual)
    {
        DNK = new int[16];
        DNK[0] = TypeOfIndividual;
        for (int i = 1; i < DNK.Length - 2; i++)
        {
            DNK[i] = Random.Range(1, 100);
        }
        DNK[14] = Random.Range(1, 5);
        DNK[15] = Random.Range(1, 3);
    }
    public Genom(int [] newDNK)
    {
        this.DNK = newDNK;
    }

    public string GetLifeType()
    {
        if (DNK[0] == 0)
        {
            return "PREDATOR";
        }
        else if (DNK[0] == 1)
        {
            return "COW";
        }
        else
        {
            return "SCAVENGER";
        }
    }
    public string GetDNK()
    {
        string result = "";
        foreach (int i in DNK)
        {
            result = $"{result} {i}";
        }
        return result;
    }
}
```

В представленной реализации определены 3 функции (включая 2 перегрузки конструктора).

Здесь экземпляры NPC генерируются на основе единого класса, используя в качестве параметров инициализации геном, поэтому необходимо обеспечить четкое представление о том представителем какого класса будет создаваемый NPC, определив данный параметр генетической информации. Именно этот параметр и поступает на вход конструктора (Genom(int TypeOfIndividual)) при создании первой популяции. Дальнейшая генерация параметров происходит автоматически на основе случайных значений из заданного диапазона. Перегрузка конструктора реализована для создания экземпляра класса Genom на основе четко определенных параметров.

Функция GetLifeType() служит для определения класса создаваемого NPC.

Функция GetDNK() возвращает массив содержащий геном для его передачи в другие функции программы.

Поскольку в создаваемой игре NPC имеют однотипную структуру и возможности взаимодействия с игровым миром они все создаются на основе класса IndividualLife. Основная структура класса и его методы представлены ниже.

```
public class IndividualLife: MonoBehaviour
{
    public Tilemap World;
    public TileBase PredatorTile;
    public TileBase CowTile;
    public TileBase ScavengerTile;
    public CellState[,] WorldInformation;
    public Hashtable LifeInformationInWorld;
    public Hashtable GrassInformationIn-
World;
    private AStar pathFinder;
    public string Hash;
    public int x_position;
    public int y_position;
    private int weekDay;
    public List<PersonalInformation>
FreeCells = new List<PersonalInformation>();
    public Memory myMemory;
    public Genom MyDNK;
    public LifeStats MyStats;
    public void Init(string NewHash, int Life-
Type, CellState[,] WorldInfo, Tilemap World-
Tilemap)
    {
        MyDNK = new Genom(LifeType);
        MyStats = new LifeStats(MyDNK);
        Hash = NewHash;
        WorldInformation = WorldInfo;
        World = WorldTilemap;
        pathFinder = new
AStar(WorldInformation);
        weekDay = 0;
    }
    public void Init(string NewHash, int[]
DNK, CellState[,] WorldInfo, Tilemap World-
Tilemap)
    {
        MyDNK = new Genom(DNK);
        MyStats = new LifeStats(MyDNK);
        Hash = NewHash;
        WorldInformation = WorldInfo;
        World = WorldTilemap;
        pathFinder = new
AStar(WorldInformation);
        weekDay = 0;
    }
    void Look() {  }
    void Vait() {  }
    void GoToTarget(string FinalyStatus) {  }
    void Eat() {  }
    void Attack(PersonalInformation Target)
{  }
    void CreateB(string MyHash, string Paren-
tHash) {  }
    void ThinkAndDoIt() {  }
    public void LiveOneDay() {  }
    public void Die() {  }
    void GoToPoint(int new_x, int new_y)
{  }
    void UseEnergy(int delta) {  }
}
```

Основными полями данного класса являются Hash, myMemory, MyDNK и MyStats.

В поле Hash хранится уникальный идентификатор каждого NPC. Этот идентификатор служит ключом хеш-таблицы, а также указано в поле Hash в состоянии игровой ячейки.

Поле myMemory является экземпляром класса Memory и служит для хранения информации о сородичах, врагах, еде и выбранной NPC цели. Поле MyDNK хранит информацию о геноме. Поле MyStats хранит информацию о характеристиках NPC, таких как количество жизни, энергии, скорость передвижения и др.

При создании NPC игровой менеджер передает в конструктор Hash, либо тип NPC, либо готовый геном, а также ссылку на массив, содержащий информацию об игровом поле.

Также в классе IndividualLife реализованы функции и методы, обеспечивающие взаимодействие с игровым миром и направленные на совершение тех или иных действий. Например, осмотр мира вокруг себя (функция Look), перемещение (функция GoToTarget), атака (функция Attack), поедание пищи (функция Eat), рождение потомства (функция CreateB) и др.

Основная функция это LiveOneDay. Именно в ней обрабатывается все поведение персонажа в течение одной единицы игрового времени.

В качестве основы для алгоритма принятия решений был выбран алгоритм FSM, в котором разработчик обобщает все возможные ситуации, а затем программирует конкретную реакцию для каждой из них. В данном случае алгоритм был скорректирован с учетом применения эволюционных вычислений. Критерии для условий перехода были привязаны к геному персонажа, также добавлен механизм случайности принятия решения на основе склонностей. Такой подход позволяет создать не столько индивидуальное поведение каждой особи, но и позволяет создать реалистичность поведения, т. к. даже одна и та же особь будет вести себя немного по-разному в одной и той же ситуации [3].

В результате, поскольку склонности привязаны к геному NPC они попадают под механизмы естественного отбора, что в свою очередь и позволяет наиболее полно сформировать подходящее поведение под реалии игрового мира, поскольку отбор происходит не только на основе значений характеристик персонажа (максимальное количество жизней, скорость бега и т. д.), но и на основе модели поведения.

СПИСОК ЛИТЕРАТУРЫ

1. *Гладков Л.А., Курейчик В.В., Курейчик В.М.* Генетические алгоритмы: учеб. пособие для студентов вузов / под ред. В.М. Курейчик. – М.: Физматлит, 2016. – 367 с.

2. *Паласиос Х.* Программирование искусственного интеллекта в играх. – М.: ДМК Пресс, 2017. – 272 с.

3. *Сапрыкина А.О.* Эволюционные операторы и принцип работы генетического алгоритма // Современные научные исследования и инновации. – 2022. – № 11. – URL:https://web.snauka.ru/issues/2022/11/99226 (дата обращения: 02.06.2023).

POSSIBILITIES OF EVOLUTIONARY METHODS FOR ORGANIZING THE STRUCTURE OF THE GAME WORLD

SAVKINA Anastasia Vasilievna
Candidate of Sciences in Technology, Associate Professor
PHILIPPOV Alexey Sergeyevich
Undergraduate Student
Mordovian State University named after N.P. Ogarev
Saransk, Russia

Requirements for the realism of the game world is an urgent problem of organizing not only the characters, but also the structure of the game world, taking into account ensuring the greatest fitness and correct behavior of the character in environmental conditions.

Keywords: NPC, genome, game object, classes, structure.