Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский Государственный университет
(национально исследовательский университет)»
Филиал ФГАОУ ВО «ЮУрГУ (НИУ)» в г. Златоусте
Факультет «Техники и технологии»
Кафедра «Математика и вычислительная техника»
Факультет Техники и технологии

Симуляция искусственной жизни

Научно исследовательская работа

Руководитель доцент, к.т.н
Соколова Е.В.
_____2018г.

Автор проекта
студент группы ФТТ-307
Б.А. Мурашов
_____2018г.

Проект защищен
с оценкой
_____2018г.

Златоуст 2018г

АННОТАЦИЯ

Мурашов Б.А. Симуляция искусственной жизни. – Златоуст: ЮУрГУ, МиВТ; 2018 г.,74 стр., 40 иллюстраций, библиографический список 34 ссылок, 6 прил.

Разработка приложения, которое осуществляет и поддерживает механизм естественного отбора в созданной среде на движке Unity. Главной идеей научно исследовательской работы является изучение теории по нейронным сетям, применение эволюционного алгоритма, анализ созданных аналогов и ранее разработанной версии приложения.

В работе рассматривается принцип и особенности работы, разрабатывается как среда, так и искусственные организмы, производится анализ и доработка недочетов предыдущий версии. В конце приводится анализ разрабатываемой второй версии приложения а так же рассматривается список доработок приложения.

СОДЕРЖАНИЕ

ПОСТАНОВКА ЦЕЛИ И ЗАДАЧИ

Целью исследовательской работы является разработка приложения симуляции искусственной жизни с использованием нейронных сетей, на движке Unity.

Задачи проекта:

1) изучение теории по нейронным сетям;
2) генетическому алгоритму;
3) движку Unity;
4) приведение и разбор первой версии приложения, данных полученных в результате расчетов и выявленных недостатков;
5) разработка второй версии программы на движке Unity, с учтенными недостатками первой версии.
6) анализ разработанной версии приложения и список доработок.

ВВЕДЕНИЕ

Искусственная жизнь — это не что иное, как попытка изучить саму жизнь, живые системы, их эволюцию, с помощью созданных человеком программ. Учёные, работающие по этому направлению, говорят, что их исследования направлены именно на практические приложения, такие как подвижные роботы, медицина, искусственный интеллект, и даже исследования социальных и экономических систем. Подобные проекты можно сравнить с закрытыми биологическими системами, созданными для изучения процессов, протекающих в природе.

ИЗУЧЕНЕ АНАЛОГОВ

1) PolyWorld (ПолиМир) Автор: Larry Yaeger, 1993[5]

Программа, написанная для развития искусственного интеллекта посредством естественного отбора и эволюционных алгоритмов.

Используется графический инструментарий Qt и OpenGL для отображения графической среды, в которой популяция трапециевидных агентов ищет пищу, союзников, размножаются и убивают друг друга (рисунок 1).

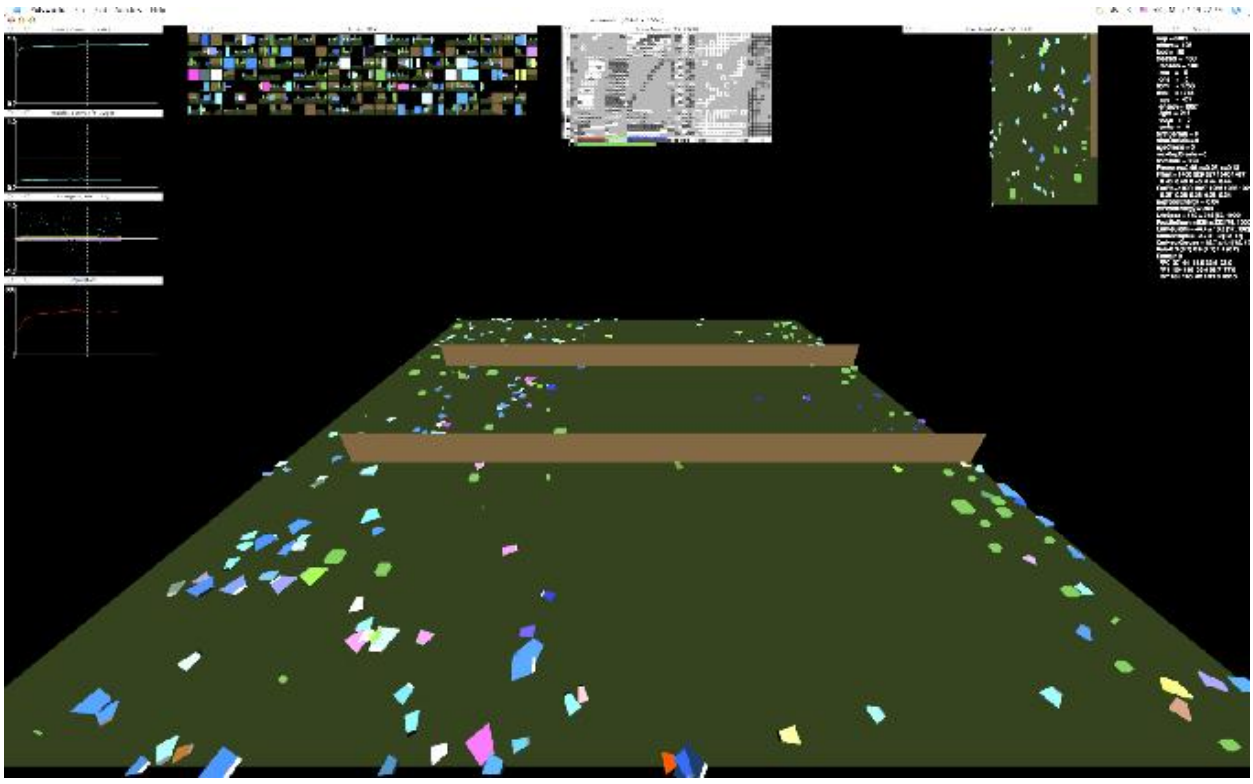

Рисунок 1- Polyworld 1994

Возникло множество интересных тактик, спонтанно возникших после продолжительной эволюции, таких как каннибализм, появления хищников и жертв.

Каждый бот принимает решения, основанные на работе нейронной сети. Геном случайным образом мутирует с заданной вероятностью, которые также изменяются в потомках организмов.

2) Interaction between learning and Evolution Авторы: D. Ackley, M. Littman, 1992 [5]

Общая схема поведения агентов:

- агенты живут в двумерном мире, разбитом на клетки;
- в клетках располагаться сами агенты, деревья, хищники.
- Агенты — это главная «единица» мира, а всё остальное разнообразным образом влияет на них.
- Хищники бьют агентов, причём сильнее, чем агенты хищников. Средством для защиты от хищников служит дерево, на которое может забраться агент, если там нет другого агента. Но и находиться постоянно на дереве опасно.
- Каждый агент имеет свою нейронную сеть, с помощью которой он и принимает решения о своих действиях.
- В процессе скрещивания двух агентов, их нейронные сети передаются потомкам — так осуществляется эволюция в модели.

Запускали модель поочередно: сначала модель только с обучением, потом модель только с эволюцией и, наконец, модель полную, и с обучением, и с эволюцией. При взаимодействии обучения и эволюции агенты не вымирали на протяжении миллиона тактов жизни, в то время как при одиночных моделях агенты вымирали в 5-7 раз быстрее.

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Генетический алгоритм — это алгоритм, используемый для решения задач оптимизации и моделирования путем случайного выбора, комбинации и изменения желаемых параметров с использованием механизмов, сходных с естественным отбором в природе. [2]

Это эволюционное вычисление, с помощью которого оптимизационные задачи решаются с использованием методов естественной эволюции, таких как наследование, мутации, выбор и пересечение. На рисунке 2 представлен график, демонстрирующий пример распределения решений, полученных с использованием генетического алгоритма.

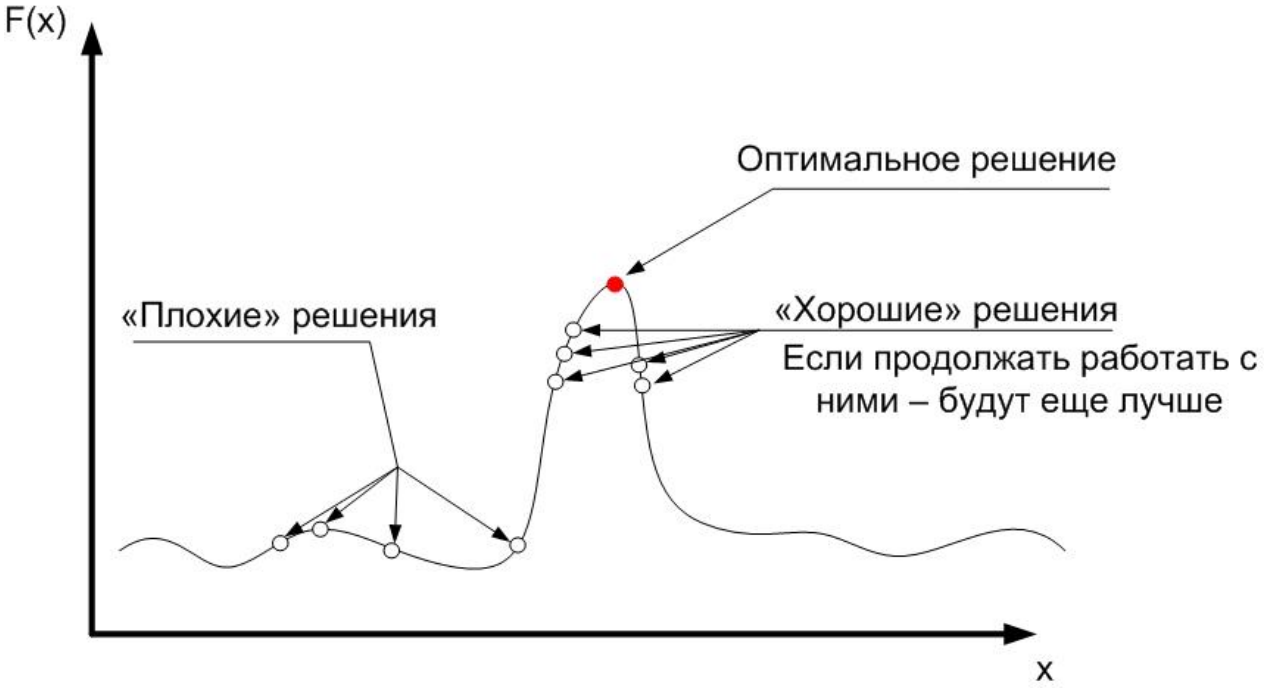

Рисунок 2 – График решений

НЕЙРОННЫЕ СЕТИ

Нейронная сеть — это математическая модель, а также ее реализация, построенная на принципе биологических нейронных сетей. [3]

Первой такой попыткой были нейронные сети В. Маккаллоха и В. Питтса. После разработки алгоритмов обучения полученные модели стали использоваться для практических целей: в задачах прогнозирования, распознавания образов, в задачах управления.

Это система связанных и взаимодействующих простых процессоров (искусственных нейронов). Каждый процессор использует сигналы, которые он получает, и сигналы, которые он передает другим процессорам. Будучи подключенным, к достаточно большой сети отдельные простые процессоры вместе могут выполнять довольно сложные задачи. На рисунке 3 представлена простейшая нейронная сеть.

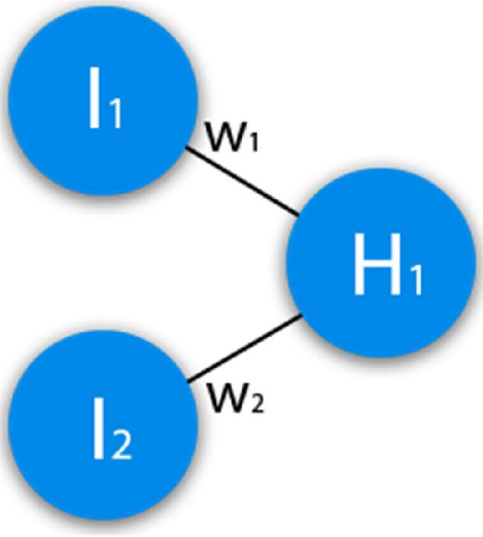

Рисунок 3 – Простейшая нейронная сеть

$$H_{1input} = (I_1 * W_1) + (I_2 * W_2)$$

$$H_{1output} = f_{activation}(H_{1input})$$

где l – входной слой, служит для записи входных значений в сеть;

W – слой синапсов, каждый синапс имеет «вес» (числовое значение), при расчетах значения с входного слоя умножаются на соответствующие им значение синапса и суммируется;

H – выходной слой, служит для возврата рассчитанного сетью нормализированного значения.

$F_{activation}$ – функция активации, нормализирует полученное значение в заданный диапазон значений (например, [0; 1]).

ПЕРВАЯ ВЕРСИЯ ПРОГРАММЫ

ЛОГИКА И ПРАВИЛА МИРА

Мир представлен ограниченной картой 80x80 клеток, на которой живут боты, каждая клетка имеет 4 характеристики: может содержать бота, органику (пища ботов), имеет значения освещения и температуры. Органика остается после смерти бота по естественным причинам или же убийства его другим ботом. Так же есть возможность питаться через фотосинтез, используя энергию света (только при условии достаточной освещенности). Освещенность варьируется от 0 до 100 процентов. Каждый бот имеет свой предел использования фотосинтеза, при определенной освещенности, для первых гибридов он равняется 50%. Каждый бот имеет комфортную температуру, в которой он может существовать (у начальных гибридов [-50;50]). Комфортная температура в клетке означает что бот может жить в клетке без ущерба уровню здоровья. Диапазон температур на поле от -100 до 100.

Каждый бот имеет параметры, характеризующие его как отдельную особь:

1) HP (Heat Point) – очки здоровья от 0 до 100;
2) Energy – внутренний запас энергии бота, который он накапливает и тратит на действия и поддержания жизни;
3) TempRange (Temperature Range) – комфортная температура для выживания;
4) Damage – урон, который, в случаи схватки наносит бот противнику (другому боту);
5) Brain – мозг бота, представленной нейронной сетью, которая управляет ботом на основе полученных данных из среды.

Имя бота состоит из набора цифр, которые формируются хеш функцией из предыдущих параметров бота, и буквы означающий поколение. Совокупность характеристик назовем геном бота. В случаи смерти всех ботов на поле, создаются новые на основе генов: лучшего из поколения (последний

умерший) и среднего гибрида (умер в середине просчета) по следующей схеме (рисунок 4). Где clone – полный клон исходного объекта, а hybrid – это гибрид, полученный в результате мутации с заданной частотой.

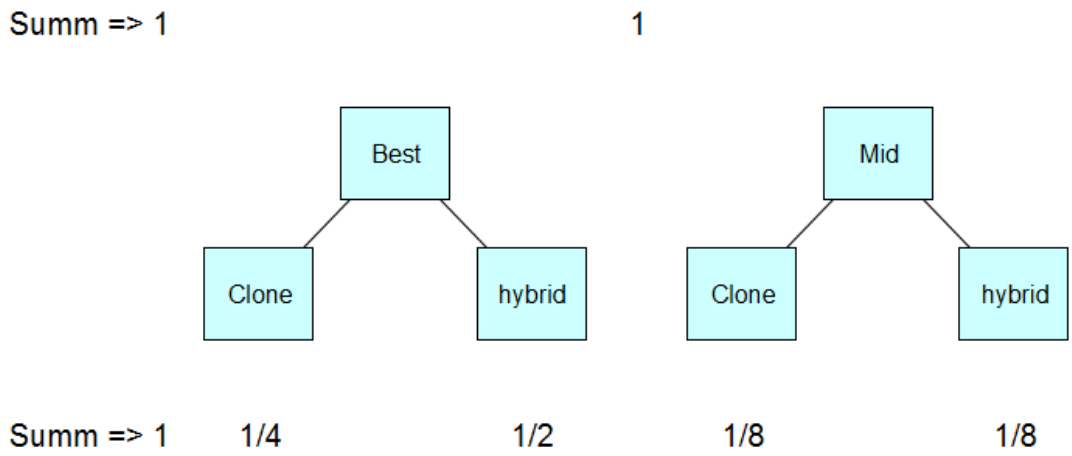

Рисунок 4 - Схема воссоздания поколения

ПРАКТИЧЕСКАЯ РАБОТА

Реализация класса Bot. Класс содержит в себе атрибуты, характеризующие бота. Метод LookAround который позволяет осмотреться в окрестности Мура 1 порядка и центральной клетке рисунок 5, и получить необходимую информацию для обработки нейронной сетью.

Окрестности Мура — совокупность четырёх клеток на квадратном паркете, имеющих общую сторону с данной клеткой.

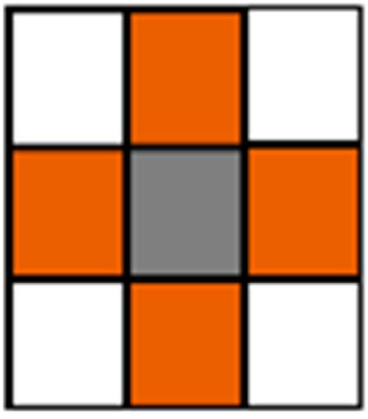

Рисунок 5 – Окрестность мура 1 порядка

Также в классе реализованы действия, которые может совершать бот:

1) Движения. Движения вправо, влево, вниз и вверх, при условии, что клетка, выбранная для движения свободна и не является границей области.

```
///<summary> Движение вверх </summary>
    void MoveUp() {
     try {
      Link.SetBot( this , (int)IG - 1 , (int)JG );
      Link.ClearBot( (int)IG , (int)JG , false );
      IG--;
      Info.Move++;
     }
     catch {}
    }
```

Другие движение производятся аналогично.

2) Питание. Питание возможно органикой, если она находится в одной клетке с ботом, а также фотосинтез при условии нахождения в освещенной зоне.

```
///<summary> Поедание органики, если она находится в одной с
ботом клетке </summary>
void EatOrganic() {
    try {
     Link[ (int)IG, (int)JG ].PLACE_ORGANIC_MATTER = false;
     ENERGY += 50;
     Info.Eat++;
      Link[ (int)IG , (int)JG ].CHANGES = true;
    }
    catch {}
}
```

3) Убийство. Убийство бота, который находится в соседней клетке. При этом действии происходит расчет показателя, который зависит от уровня здоровья и базового урона. Тот бот, у которого этот показатель больше и выходит победителем из схватки.

4) Размножение. Бот может произвести потомство, при условии наличия достаточного уровня энергии и свободного пространства вокруг. Потомок будет иметь мутировавшие гены предка.

5) Лечение. Бот может произвести восполнение очков здоровья за счет внутренней энергии. Не выше, чем базовое значение.

```
///<summary> Лечение за счет энергии бота </summary>
void HealBot() {
    try {
      int MisingHp = ( 100 - HP_GET ) * 2;
     if (MisingHp >= ENERGY)
     MisingHp = ( MisingHp - ENERGY ) - 10;
     ENERGY -= MisingHp;
     HP_GET += ( MisingHp / 2 );
    }
    catch {}
}
```

6) Реализация класса Square: класс представляет одну клетку всего поля, на котором обитают боты. Содержит 4 атрибута: наличие бота, наличие органики, освещенность и температуру. А также атрибут Change который показывает была ли клетка изменена на текущем просчете.

```
class Square {
    ///<summary> Поле бота </summary>
    Bot PlaceBot;
    ///<summary> Поле органической материи </summary>
    bool PlaceOrganicMatter;
    ///<summary> Поле освещенности </summary>
    byte PlaceLight;
    ///<summary> Поле температуры </summary>
    sbyte PlaceTemp;
    ///<summary> Была ли клетка изменена </summary>
    bool Changes;
    ...
}
```

Класс Field позволяет работать с полем N x N размера из клеток Square, реализуя необходимые методы: начальная инициализация температуры, освещенности и органики. Позволяет хранить всех ботов в Hashtable, проверять условия в отдельных клетках и поддерживает доступ к отдельным участкам поля.

```
class Field {
    ///<summary> Матрица клеток с ботом </summary>
    Square[,] MatrixBot;
///<summary> Хеш таблица для хранения ботов </summary>
    Hashtable ListBot;
///<summary> Размер квадратного поля  </summary>
    int N;
    ///<summary> Коструктор класса Field </summary>
```

```
    ///<param name="n">размер поля</param>
    ...
}
```

Классы, обеспечивающие работу нейронной сети рисунок 6:

- NeuralComponets – абстрактный класс, содержащий основные поля, обеспечивающие работу нейронной сети и ее компонентов;
- OutputLayer – класс выходного слоя нейронной сети, обеспечивающий вывод обработанных данных;
- InputLayer – класс входного слоя, обеспечивающий прием данных и передачу на первый уровень скрытых нейронов.
- HiddenLayer – класс скрытых нейронов, обеспечивающий прием и передачу данных, также формализацию и хранения их.
- SynapseLayer – класс синапсов, служит для обработки информации.

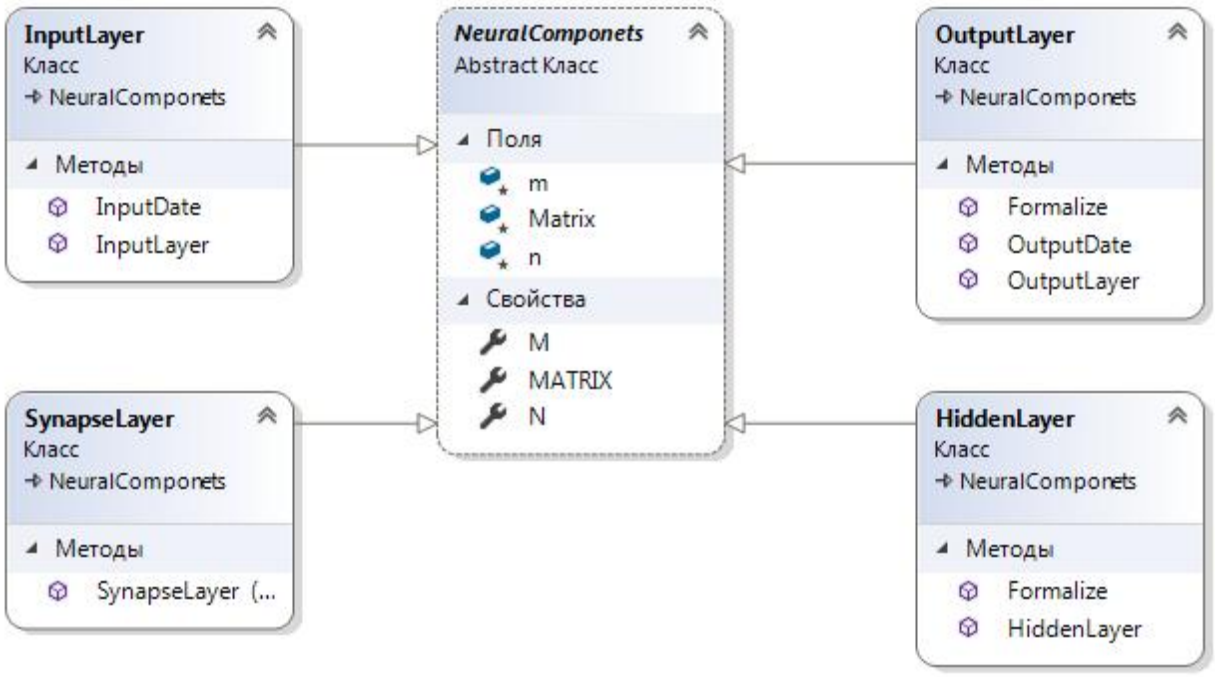

Рисунок 6 - Диаграмма классов нейронной сети

Дополнительные классы для работы сети (рисунок 7):

- NeuralNetwork – Класс, обеспечивающий работу нейронной сети архитектуры (рисункок 8).

- Function и Matrix – статические классы, содержащие: функцию активации, распределения Гаусса и метод умножения матриц.

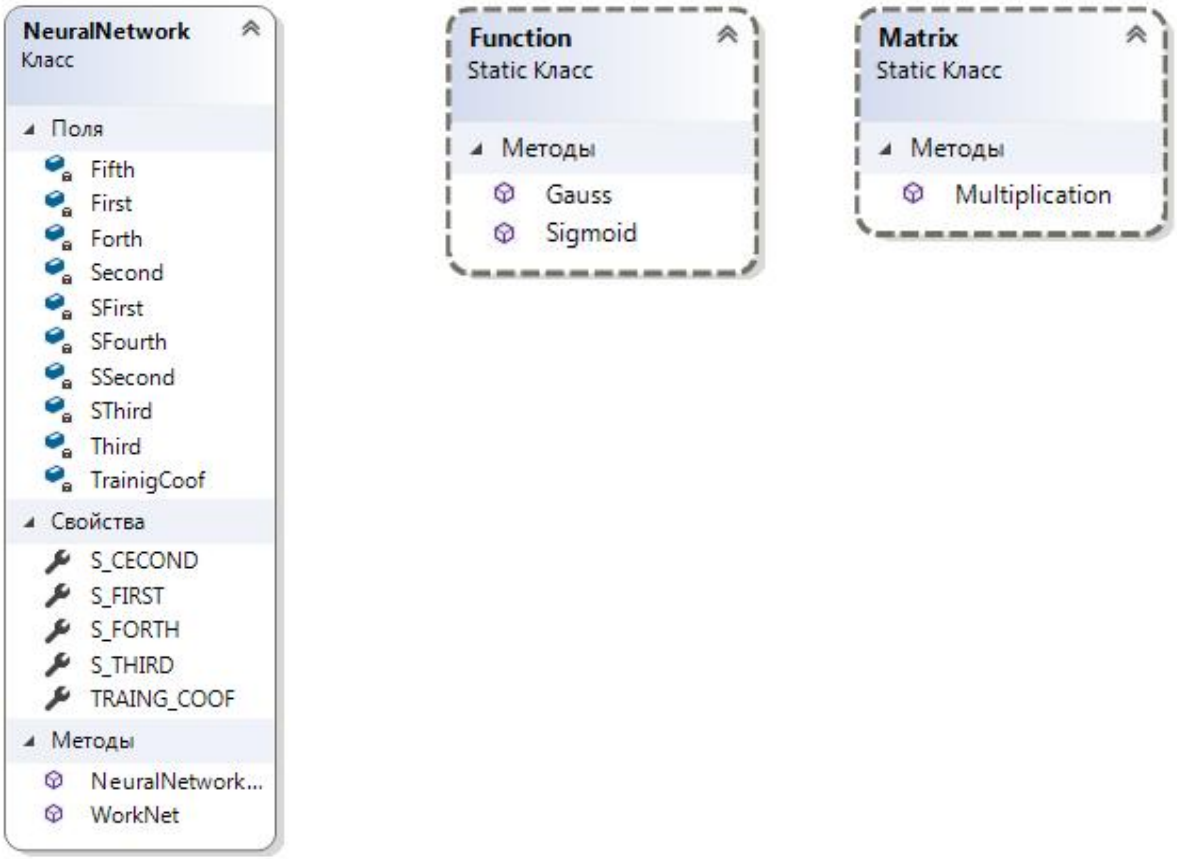

Рисунок 7 - Дополнительные классы

Класс ConsoleDebugging (рисунок 8) предназначен для работы с консолью, в нее идет отображение отладочной информации, отсканированной ботом информации, информации, полученной после обработки нейросетью и выполнение действия.

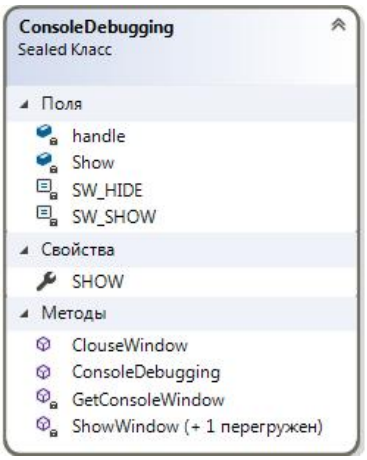

Рисунок 8 - Диаграмма класса

Последний класс WebCenter обеспечивает работу программы, связь между отдельными компонентами и графическим интерфейсом, а также сохранение и загрузку файлов нейросети (рисунок 9).

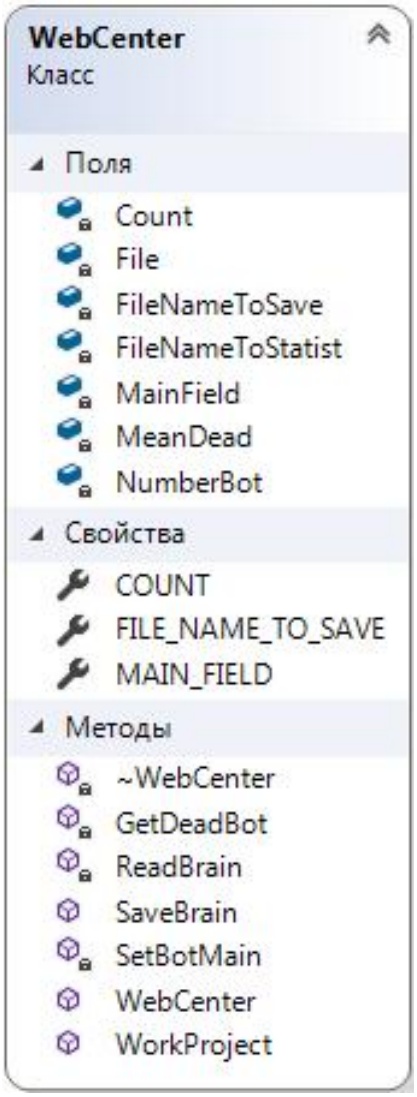

Рисунок 9 – Диаграмма класса

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПРОГРАММЫ

Графический интерфейс выполнен в window form и состоит из 3 форм:

1) Страница загрузки;
2) Страница меню;
3) Главное окно.

1) Страница загрузки. Представляет собой форму с картинкой и таймером на 3 секунды (рисунок 10).

Рисунок 10 - Стартовая страница

2) Страница меню. Форма, где возможно выбрать: файл с сохранением нейросети, количество ботов, коэффициент обучаемости ботов и динамические условия среды(температура). Все окна снабжены всплывающими подсказками (рисунок 11).

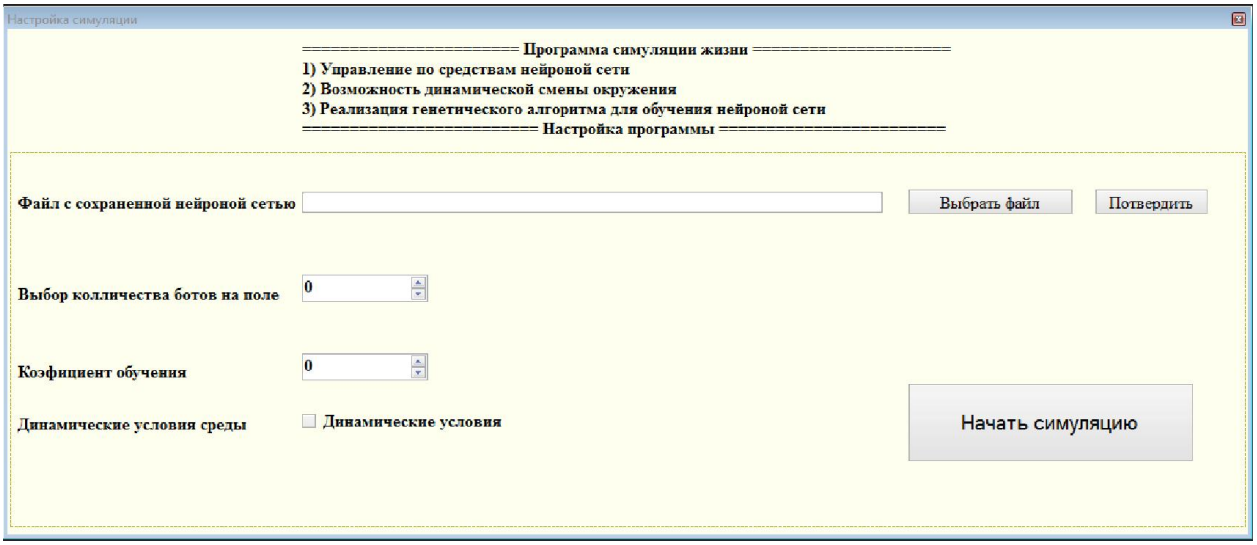

Рисунок 11 - Страница меню

3) Главная страница. На этой странице осуществляется основная работа программы. Состоит из трех главных полей: поле ботов, поле освещения, поле температуры. Также есть пункты управления просчетами,

отладка открывающая консоль и возможность упростить графическое отображение для более быстрого просчета. Также есть возможность сохранять файлы статистики и преобразовать его в файл с расширением xlsx, сохранить файл нейронной сети (рисунок 12).

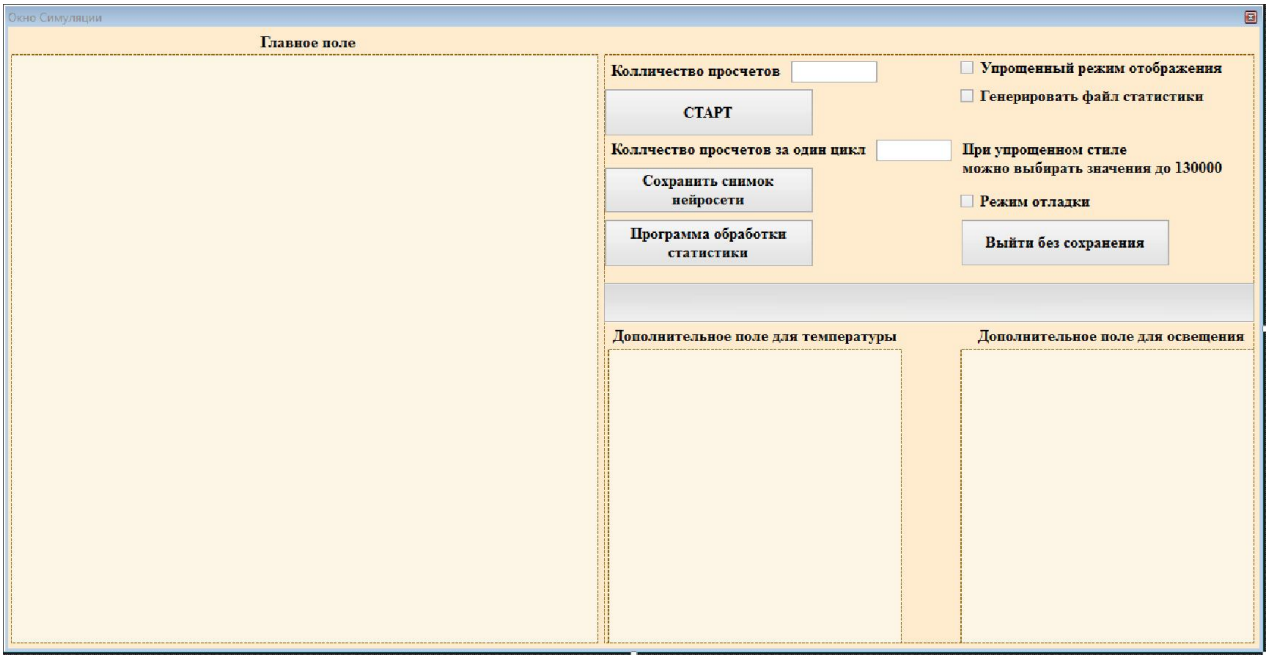

Рисунок 12 - Главное окно

ПРИМЕР РАБОТЫ ПРОГРАММЫ

В окне меню выбран файл с сохранением нейросети и выбраны стандартные настройки (рисунок 13).

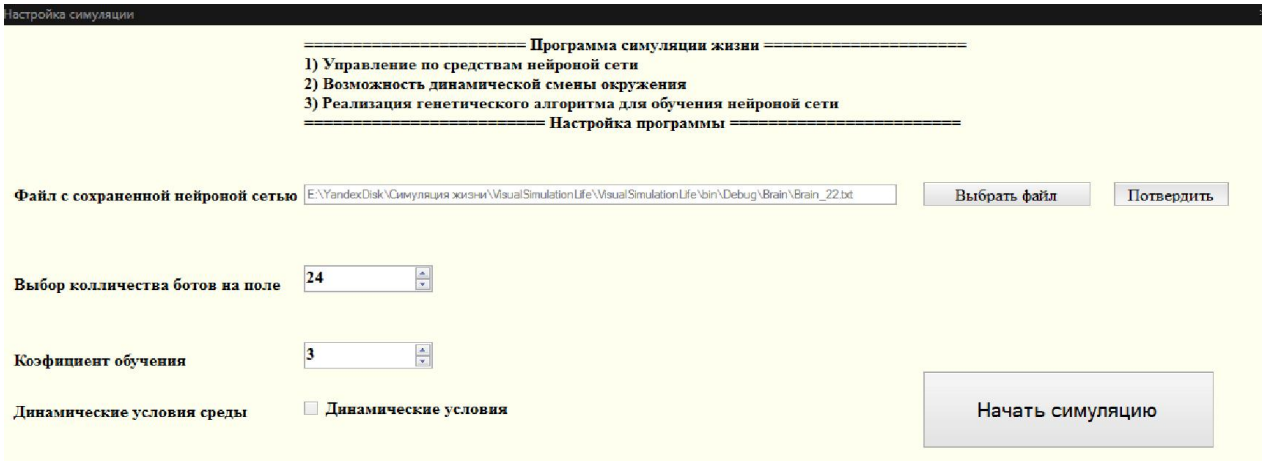

Рисунок 13 – Окно меню

На рисунке 14 происходит просчет 140 и графическое отображение работы программы.

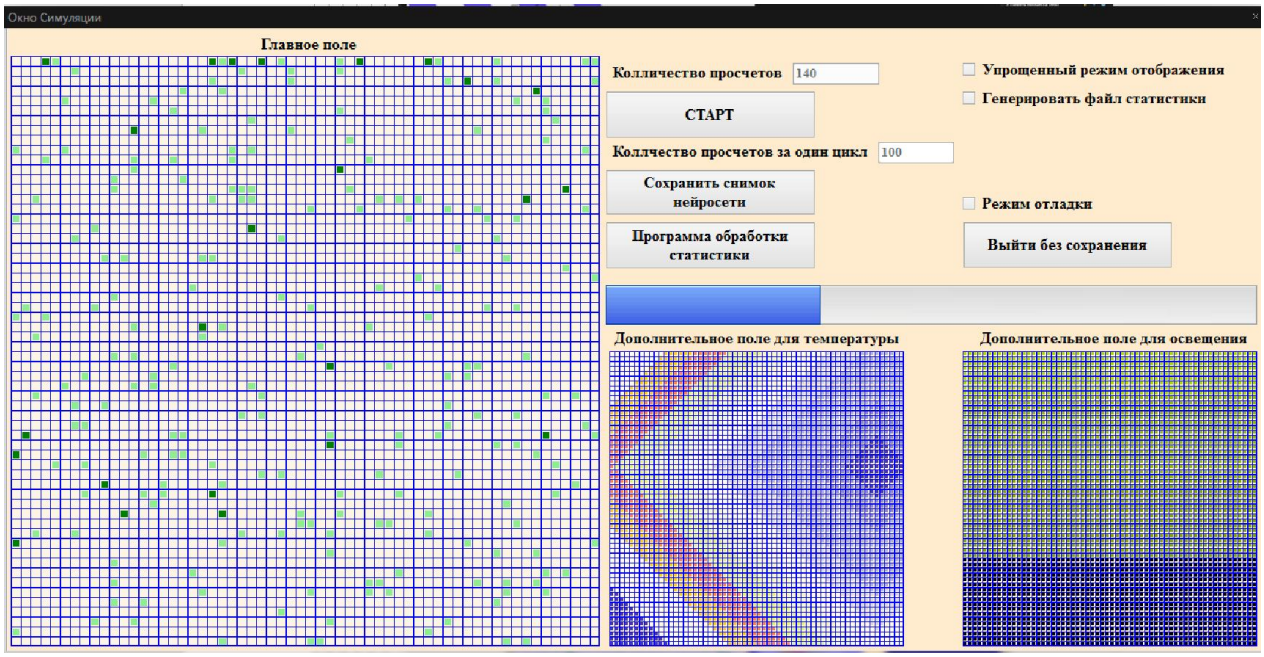

Рисунок 14 - Работа программы в главном окне

АНАЛИЗ ЭКСПЕРЕМЕНТАЛЬНЫХ ДАННЫХ

В результате почти миллионного просчета получены экспериментальные данные в количестве 22161 единиц информации о ботах. Они включают: возраст, имя, количество действий общее и по категориям. Воспользовавшись разработанным методом по экспорту данных в Excel, получаем таблицу (рисунок 15).

Имя	Еда	Движения	Убийство	Размножение	Возраст	Энергия	Номер
46799533v	0	15	0	0	15	-56.	1
10994532m	0	15	0	0	15	-50.	2
34751559e	0	15	0	0	15	-50.	3
44271515v	0	15	0	0	15	-53.	4
17643021w	0	15	0	0	15	-50.	5
544835021	0	15	0	0	15	-50.	6
49629946j	0	15	0	0	15	-50.	7
36049206b	0	15	0	0	15	-50.	8
34123687t	0	15	0	0	15	-50.	9
16606917n	0	15	0	0	15	-50.	10
39910505x	0	15	0	0	15	-50.	11

Рисунок 15 – Данные расчетов

Построим графики по полученным данным (рисунок 16, 17, 18):

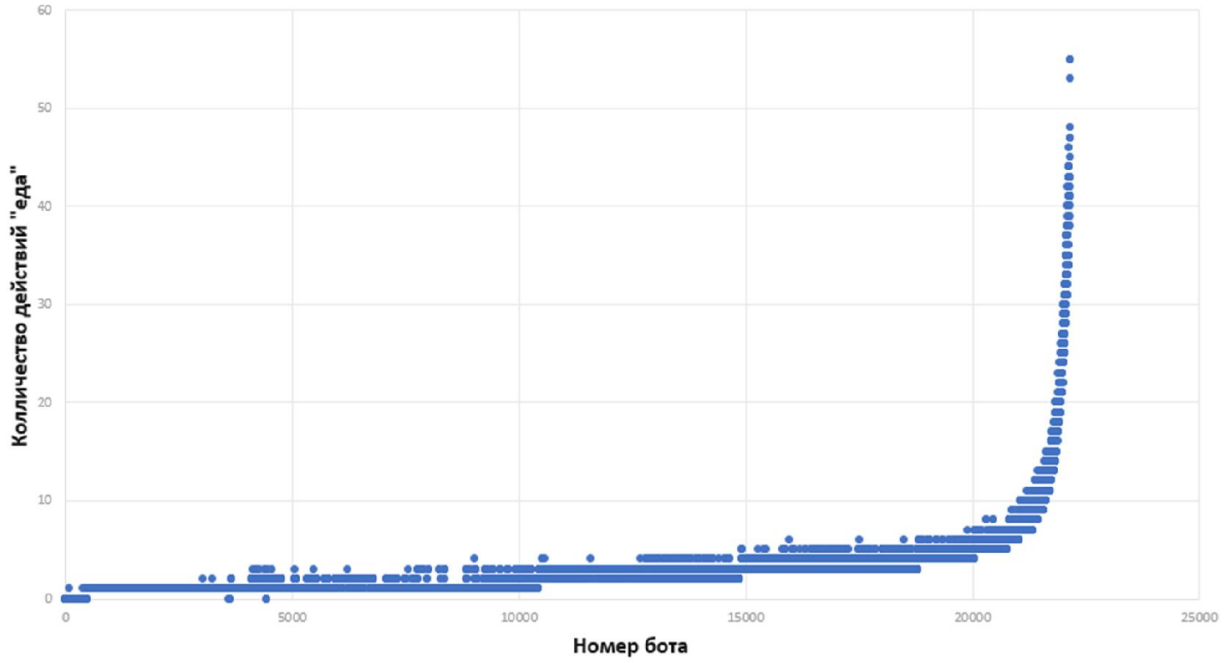

Рисунок 16 - График зависимости номера расчета от количества действий «еда»

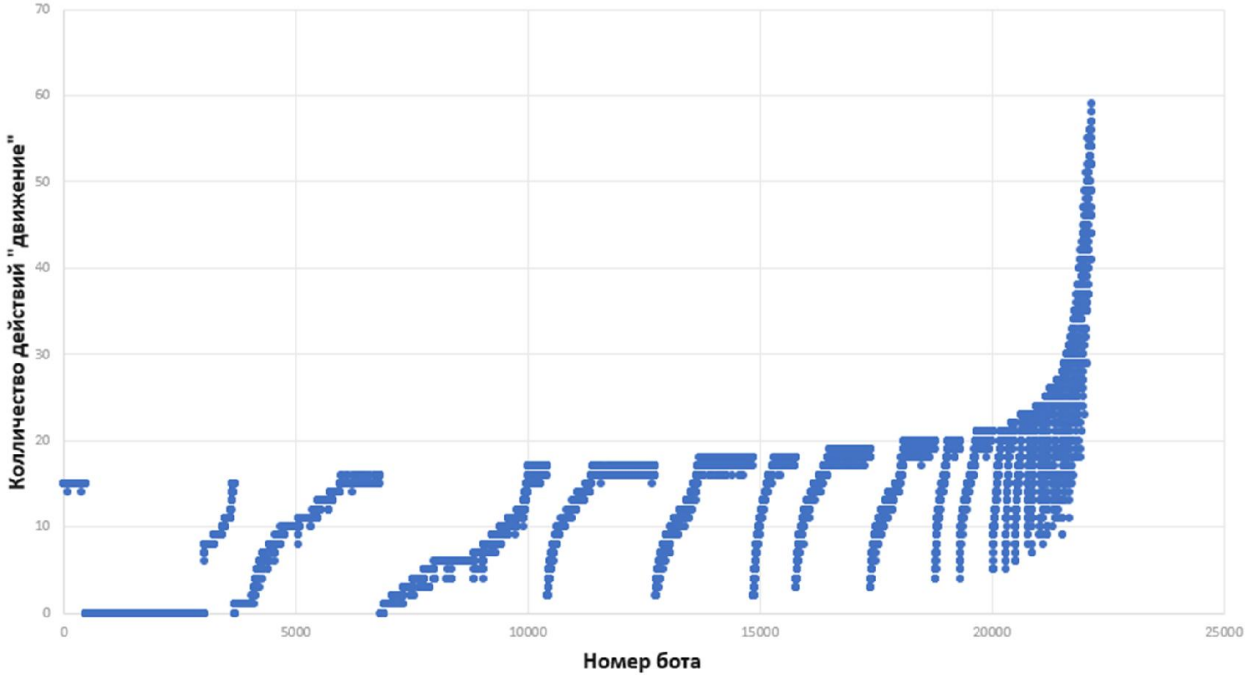

Рисунок 17 - График зависимости номера расчета от количества действий «движение»

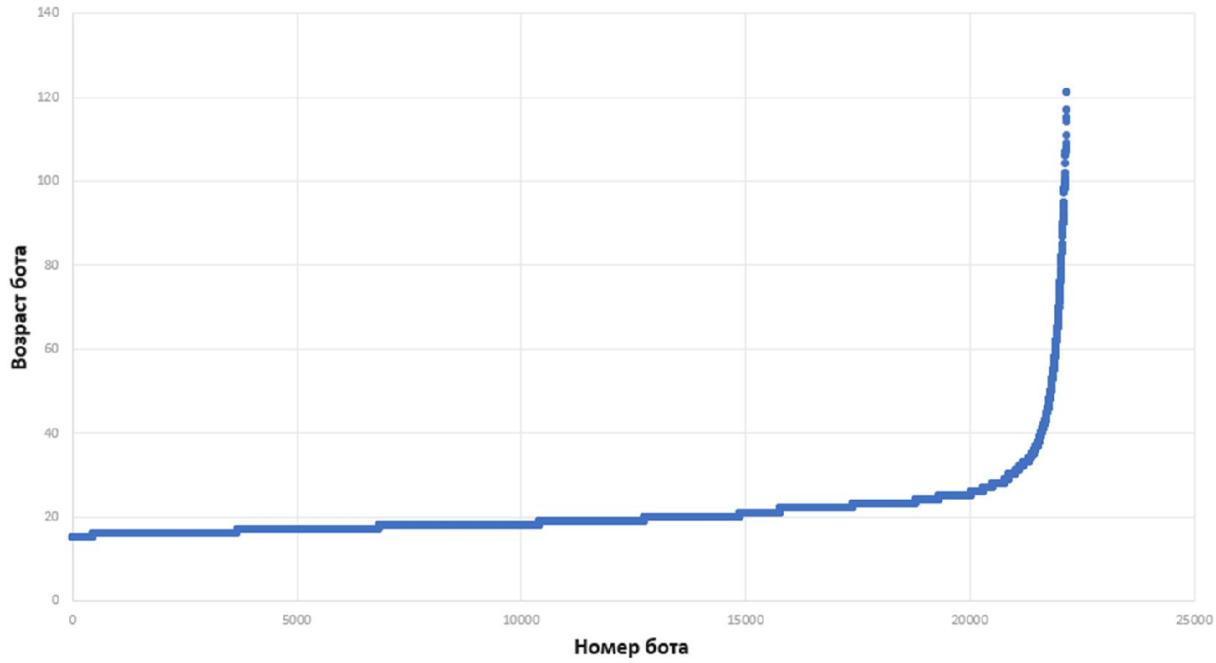

Рисунок 18 - График зависимости возраста от номера расчета

По графикам видно, что зависимость присутствует, чем дольше идет расчет, тем лучшие результаты получаются, увеличивается возраст бота и количество действий. На рисунке 17 видно «рывки» что обусловлено подбором ботами тактики выживания. В ходе расчетов были замечены следующие тактики:

- «Корова». Бот движется в комфортную зону поглощая всю встреченную органику;
- «Стадо». Боты движутся группой, где движение обеспечивается смертью последних и рождению первых;
- «Бездействие». Замечено, что боты в любой случаи, временно используют эту тактику. Заключается тактика в полном бездействии, для сохранения энергии.

ВЫВОД ПО ПЕРВОЙ ВЕРСИИ

Из проведенных экспериментов и полученных данных можно сделать следующие выводы: программа корректно, максимальный полученный возраст бота 120 получен при 1 млн. расчетов, проанализирован используемая архитектура нейронной сети, разработанные правила. Для улучшения

программы предлагается сделать: перенести программу на другой движок для лучшей скорости и гибкости, на пример, Unity. Также лучшим решением будет совместить эволюционный подход обучения нейронной сети и другие методы (с учителем и смешанная), для лучшего результата.

ВТОРАЯ ВЕРСИЯ ПРОГРАММЫ

Для разработки приложения выбран движок Unity, так как в нем используется автоматическое распределения действий на потоки при использовании «coroutines», гибкий движок предоставляет большое количество возможностей для реализации. В качестве языка написания скриптов выбран C#, так как в отличии от второго доступного JavaScript, используется статическая типизация и поддерживается Visual Studio, что способствует написанию кода без неоднозначностей.

ДОРАБОТКИ И ОТЛИЧИЕ ОТ ПРЕДЕДУЩИЙ ВЕРСИИ

Мир представлен ограниченной картой 1000x1000 относительных единиц, на которой живут боты. Органика спавнится случайным образом до достижения определенного значения и является пищей для ботов. Так же есть возможность питаться через фотосинтез, используя энергию света от «солнца» которая располагается в случайной точке над картой. Освещенность рассчитывается как отношения расстояний от солнца к боту и перпендикуляр от солнца до поверхности.

Боты имеют зону видимости и направления взгляда (рисунок 19). В приложении отсутствует зависимость от температуры, бот имеет возможность поворачивается вокруг себя и двигаться на дискретное расстояние. Применена другая схема размножения, мутации и сохранения нейронных сетей. Мир выполнен в 3D.

Доступные действия для ботов:

1. Движение вперед;
2. Вращение вокруг оси (45°);
3. Фотосинтез;
4. Питание органикой;

5. Атака другого бота;
6. Размножение.

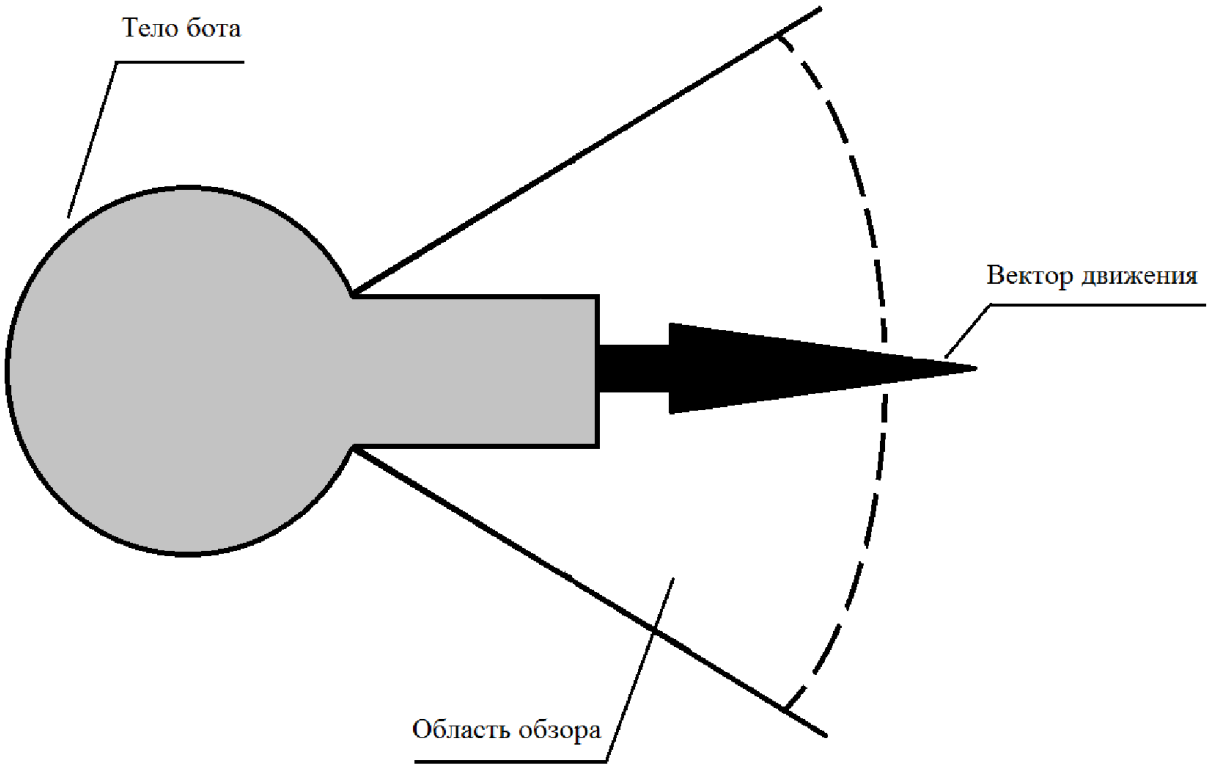

Рисунок 19 – Схема «взгляда» бота

1) Движение вперед:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveForward : MonoBehaviour {
    /// <summary>
    /// object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;
    private float _distance;

    /// <summary>
    ///  Move bot forward
    /// </summary>
    public void Move() {
     _distance = 0;
     ///Check border field
     if (490 <= Stat.Speed + transform.position.x || -490 >=
Stat.Speed + transform.position.x || 490 <= Stat.Speed +
transform.position.z || -490 >= Stat.Speed +
transform.position.z) {
```

```
        ///Fine network
        Stat.Web.ValueCrit--;
        Stat.HP -= 10;
        return;
       }

       StartCoroutine( MoveAnimation() );
      }

      /// <summary>
      /// Animation parallel flow
      /// </summary>
      IEnumerator MoveAnimation() {
       if (_distance < Stat.Speed) {
        if (490 <= transform.position.x || -490 >=
transform.position.x || 490 <= transform.position.z || -490 >=
transform.position.z) {
         StopCoroutine( MoveAnimation() );
         yield return null;
        }
        else {
         _distance += Stat.Speed / 20f;
         transform.Translate( Stat.Speed / 20f, 0, 0 );

         yield return new WaitForSeconds( 0.01f );
         StartCoroutine( MoveAnimation() );
        }
       }
       else {
        StopCoroutine( MoveAnimation() );
       }
      }
}
```

2) Вращение вокруг оси:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotationBody : MonoBehaviour {
    /// <summary>
    /// object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;

    /// <summary>
```

```
    /// Rotation bot
    /// </summary>
    public void Rotation(float value) {
     transform.Rotate( 0, value * Stat.RotationAngle, 0 );
    }
}
```

3) Фотосинтез:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Fotosintez : MonoBehaviour {
    /// <summary>
    /// Object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;

    /// <summary>
    /// Food photosynthesis
    /// </summary>
    public void MoveFotosintez() {
     Stat.Energy += Stat.LightLevel * Stat.Photosensitivity;
    }
}
```

4) Питание органикой:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Eat : MonoBehaviour {
    /// <summary>
    /// Object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;

    /// <summary>
    /// Eat organic
    /// </summary>
    public void EatMove() {
     /// Find object around
     Collider [] EatColl = Physics.OverlapSphere(
transform.position, Stat.RadiusAction );
     foreach (Collider thisObject in EatColl) {
      Vector3 direction = thisObject.transform.position -
transform.position;
```

```
        ///  True distance
        if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
         if (this.tag == "Organic") {
          Stat.Energy +=
thisObject.GetComponent<SpecificationsOrganic>().val;
          thisObject.GetComponent<EatOrganic>().MoveEat();
          return;
         }
        }
       }
       /// Find web
       Stat.Web.ValueCrit--;
       Stat.HP -= 10;
      }
}
```

5) Атака:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Attack : MonoBehaviour {
    /// <summary>
    /// Object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;

    /// <summary>
    /// Attac bots
    /// </summary>
    public void AttacMove() {
     Collider [] BotColl = Physics.OverlapSphere(
transform.position, Stat.RadiusAction );
     foreach (Collider thisObject in BotColl) {
      Vector3 direction = thisObject.transform.position -
transform.position;
      if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
       if (this.tag == "Bot") {
        try {
         Stat.Energy +=
thisObject.GetComponent<Specifications>().HP;
         thisObject.GetComponent<Specifications>().HP = 0;
        }
        catch { }
        return;
       }
```

```
      }
     }
     /// Find web
     Stat.Web.ValueCrit--;
     Stat.HP -= 10;
    }}
```

6) Размножение:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using NeiralNet;

public class Reproduction : MonoBehaviour {
    /// <summary>
    /// Object reference with characteristics
    /// </summary>
    [SerializeField] Specifications Stat;
    /// <summary>
    /// Template bot
    /// </summary>
    [SerializeField] GameObject BotPrafab;
    /// Specification reproduct
    private float _priceRep = 70f;
    private float _randZoneMin = -3;
    private float _randZoneMax = 3;

    public void MoveReproduction() {

     if (Stat.Energy > _priceRep) {
      Stat.Energy -= _priceRep;
      Debug.Log( "Reproduction => 1" );
      GameObject newBot = Instantiate( BotPrafab ) as
GameObject;
      newBot.transform.position =
this.gameObject.transform.position;
      newBot.transform.position = new Vector3(
newBot.transform.position.x, newBot.transform.position.y,
newBot.transform.position.z );
      newBot.transform.parent = GameObject.Find( "Floor"
).transform;
      newBot.transform.localScale =
this.gameObject.transform.localScale;
      newBot.GetComponent<Specifications>().Name =
char.ConvertFromUtf32( Random.Range( 65, 91 ) ) + Random.Range(
```

```
1000, 10000 ).ToString() + char.ConvertFromUtf32( Random.Range(
65, 91 ) );
      Mutation( newBot.GetComponent<Specifications>() );
      GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots.Add( newBot );
     }
     else
     {
      Stat.Web.ValueCrit--;
      Stat.HP -= 10;
     }
    }
    /// <summary>
    /// Parameter mutation
    /// </summary>
    private void Mutation(Specifications baseStat) {
     baseStat.Photosensitivity = Stat.Photosensitivity +
Random.Range( _randZoneMin, _randZoneMax );
     baseStat.Speed = Stat.Speed + Random.Range( _randZoneMin,
_randZoneMax );
     baseStat.RotationAngle = Stat.RotationAngle + Random.Range(
_randZoneMin, _randZoneMax );
     baseStat.Energy = 50;
     baseStat.HP = 100;
     baseStat.Web = new NeiralNet.NeuralNetwork( Stat.Web, true
);
    }
}
```

Объект Specifications содержит параметры бота (жизни, энергия и т.д.), а также «осмотр», окружающий среды бота, подготовка данных для нейронной сети, расчет и выбор действий.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;
using NeiralNet;

public class Specifications : MonoBehaviour {

    public int LifeTime = 0;

    public string Name = string.Empty;

    public float Speed = 10f;
```

```
public float RotationAngle = 45;

public NeiralNet.NeuralNetwork Web;

private GameObject _sun;

public float Energy {
 set {
  if (value > 100) {
   _energy = 100;
   return;
  }
  _energy = value;
 }
 get { return _energy; }
}

public float Photosensitivity = 50f;

public float RadiusAction = 2f;

public delegate void baseMod(GameObject obj);

public event baseMod EventDie;

public int TrainCoof = 2;

public float LightLevel = 0;
public float PossibilityMove = 0;
public float AvailabilityFood = 0;
public float BreedingOpportunity = 0;
private float _energy =50;
private float _hp = 100;

public float HP {
 get { return _hp; }
 set {
  if (value < 0) {
   if (EventDie!=null)
    EventDie( this.gameObject );
   return;
  }
  if (value > 100)
   _hp = 100;
  else
```

```
        _hp = value;
      }
    }

    private void Start() {
      _sun = GameObject.FindWithTag("Sun");
      Web = new NeuralNetwork( TrainCoof );
      StartCoroutine( HPUpdateEnrgy() );
      StartCoroutine(LookAround());
    }

    private IEnumerator HPUpdateEnrgy() {
      if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
        HP += (float)( -0.0045 * _energy * _energy + 1.152 *
_energy );

        yield return new WaitForSeconds( 5f );
      }
      else {
        yield return new WaitForSeconds( 0.1f );
      }


      StartCoroutine( HPUpdateEnrgy() );
    }

    private IEnumerator LookAround() {
      if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
        LightLevel = ( _sun.transform.position -
gameObject.transform.position ).magnitude /
_sun.transform.position.y;
        try {
          LightLevel = Mathf.Sqrt( (float)( -0.42 * ( LightLevel -
2.2 ) ) );
          if (LightLevel != LightLevel)
            throw new System.Exception();
        }
        catch {
          LightLevel = 0;
        }

        PossibilityMove = 1;
        Collider [] BotColl = Physics.OverlapSphere(
transform.position, RadiusAction );
```

```
        foreach (Collider thisObject in BotColl) {
         Vector3 direction = thisObject.transform.position -
transform.position;
         if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
          if (thisObject.tag != "Space" || thisObject.tag ==
"Wall") {
           PossibilityMove = 0;
           break;
          }
         }
        }

        AvailabilityFood = 0;
        Collider [] EatColl = Physics.OverlapSphere(
transform.position, RadiusAction );
        foreach (Collider thisObject in EatColl) {
         Vector3 direction = thisObject.transform.position -
transform.position;
         if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
          if (thisObject.tag == "Organic") {
           AvailabilityFood = 1;
           break;
          }
         }
        }

        if (_energy >= 70) {
         BreedingOpportunity = 1;
        }
        else {
         BreedingOpportunity = 0;
        }

        StartMove();

        _hp -= 5;

        if (GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.MaxLifeCount <
LifeTime)
         GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.MaxLifeCount =
LifeTime;

        yield return new WaitForSeconds( 0.1f );
       }
```

```
        else {
         yield return new WaitForSeconds( 0.1f );
        }
        StartCoroutine( LookAround() );
       }

       private void StartMove() {
        float[] hashArray = new float[6];

        hashArray[0] = LightLevel;
        hashArray [1]= PossibilityMove;
        hashArray [2]= AvailabilityFood;
        hashArray [3]= BreedingOpportunity;
        hashArray [4]= _energy;
        hashArray [5]= _hp;

        float[,] outValue = Web.WorkNet( hashArray );

        float maxValue = -2;
        int maxIndex = 0;

        for (int i = 0 ; i < 6 ;++i) {
         if (maxValue < outValue [ 0, i ]) {
          maxValue = outValue [ 0, i ];
          maxIndex = i;
         }
        }

        switch (maxIndex){
         case 0: {
          this.gameObject.GetComponent<MoveForward>().Move();
          break;
         }
         case 1: {
          this.gameObject.GetComponent<RotationBody>().Rotation(
maxValue );
          break;
         }
         case 2: {
          this.gameObject.GetComponent<Eat>().EatMove();
          break;
         }
         case 3: {

       this.gameObject.GetComponent<Fotosintez>().MoveFotosintez();
          break;
```

```
        }
        case 4: {

    this.gameObject.GetComponent<Reproduction>().MoveReproduction
();
         break;
        }
        case 5: {
         this.gameObject.GetComponent<Attack>().AttacMove();
         break;
        }
       }
       Web.ValueCrit++;
       LifeTime++;
      }
}
```

РАБОТА С НЕЙРОНОЙ СЕТЬЮ

Спроектированная схема нейронной сети представлена на рисунке 20. Она содержит 6 нейронов входного слоя, 2 скрытого слоя и 6 выходного слоя, а также в сумме 24 синапса с весами.

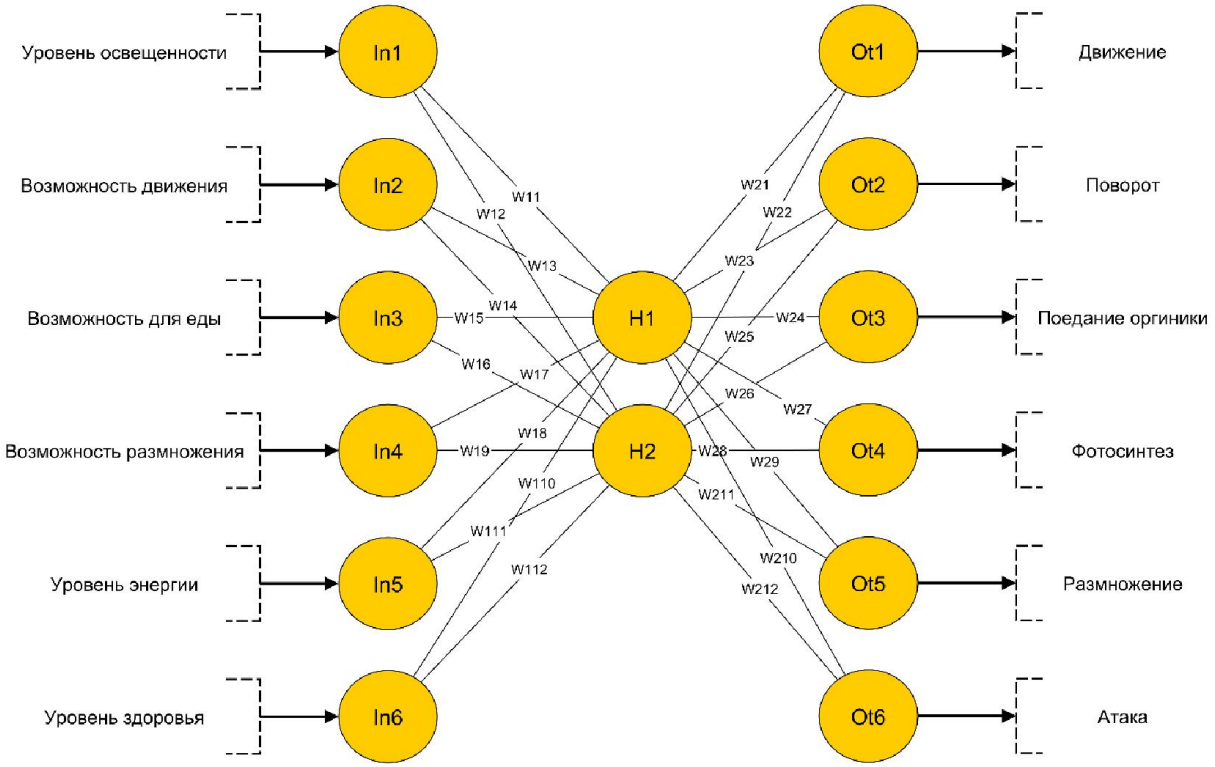

Рисунок 20 – Схема используемой нейронной сети

$sigmoid(x) = \frac{1}{1+e^{-x}} \in [-1; +1]$ – функция активации (нормализации) значений в нейронах. [3]

Расчетная формула для узла H1:

$$H1 = sigmoid(\sum_{i=0}^{6} In_i * w_{1i})$$

Расчетная формула для узла Ot1:

$$Ot1 = sigmoid(\sum_{i=0}^{2} H_i * w_{2i})$$

Нейронная сеть организована несколькими классами: `InputLayer` (Входные данные), `OutputLayer` (Выходные данные), `SynapseLayer` (Слой синапсов), `HiddenLayer` (Скрытый слой) которые наследуют абстрактный класс `NeuralComponents`.

```
namespace NeiralNet {
    ///<sumarry>
    ///Абстрактный класс, для компонетов нейронной сети
    ///</sumarry>
    abstract class NeuralComponets {
     ///<sumarry>
     ///Матрица весов компонента
     ///</sumarry>
     protected float [,] _matrix;
     ///<summary>
     ///Размеры матрицы
     ///</summary>
     protected int _n, _m;
     ///<summary>
     ///Чтение кол-ва строк
     ///</summary>
     public int N {
      get { return _n; }
     }
     ///<summary>
     ///Чтение кол-ва столбов
     ///</summary>
     public int M {
      get { return _m; }
     }
     /// <summary>
     ///  Чтение матрицы весов
     /// </summary>
     public float [,] Matrix {
      get { return _matrix; }
      set { _matrix = value; }
     }
    }
}
```

Диаграмма классов представлена на рисунке 21.

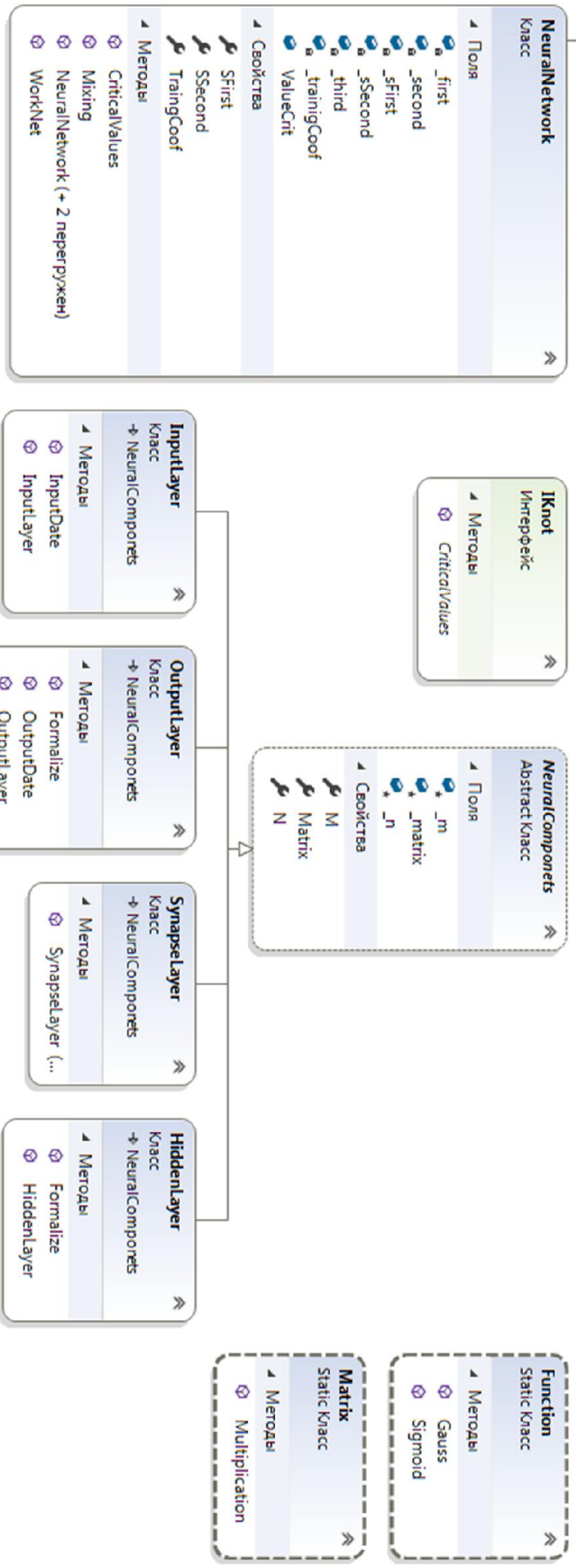

Рисунок 21 – Диаграмма классов

Класс `NeuralNetwork` организует сеть, содержит методы для расчетов и модификации сетей. Классы `Function` и `Matrix` содержат функцию активации и метод перемножения матриц соответственно.

Принцип работы приложения выглядит следующим образом: после начала работы создаются боты, содержащие собственные экземпляры нейронной сети, заполнены случайно, боты проживают свой цикл, после смерти всех ботов лучший экземпляр сети записывается в бинарное дерево, где критерием оценки является количество прожитых циклов.

Бинарное дерево организовано шаблонным классом `Tree<Template>` с методами получения крайних левых и правых значений.

Далее для новой популяции выбирается лучший и худший из экземпляров сетей, на их основе создается новое поколение в отношениях, представленных на рисунке 22.

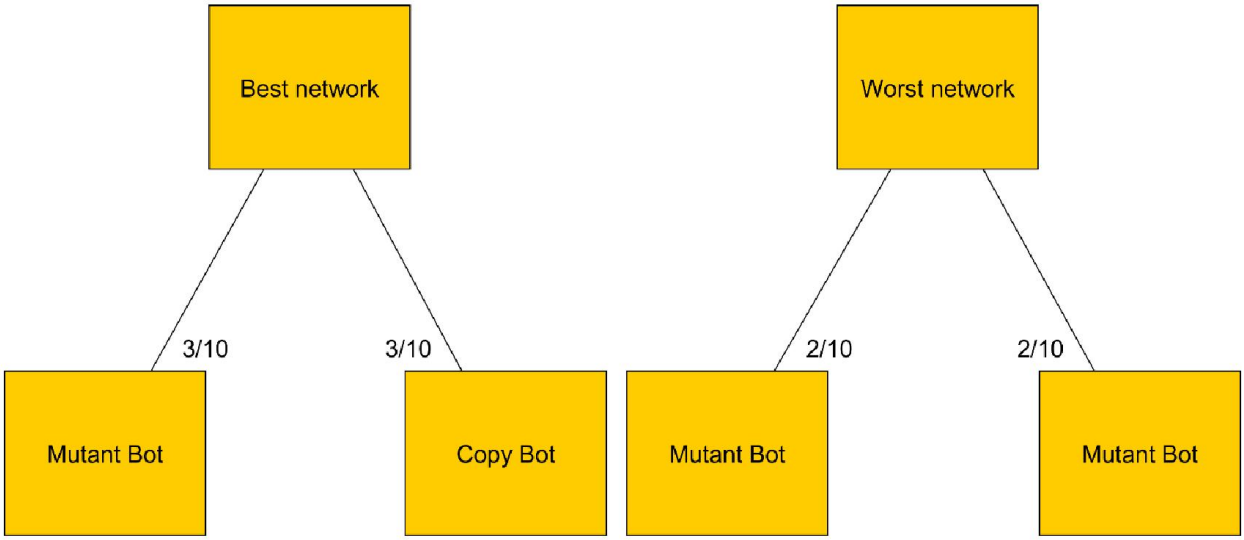

Рисунок 22 – Соотношения сетей в новом поколении

РАБОТА С ДВИЖКОМ UNITY

В разработке приложения используется unity версии 3.1.f1, в качестве языка написания скриптов выбран C#.

Поле, на котором выживают боты выглядит как «коробка» (стены и пол), (рисунок 23), вокруг этой области расположены камеры, позволяющие наблюдать за процессом развития ботов (рисунок 24).

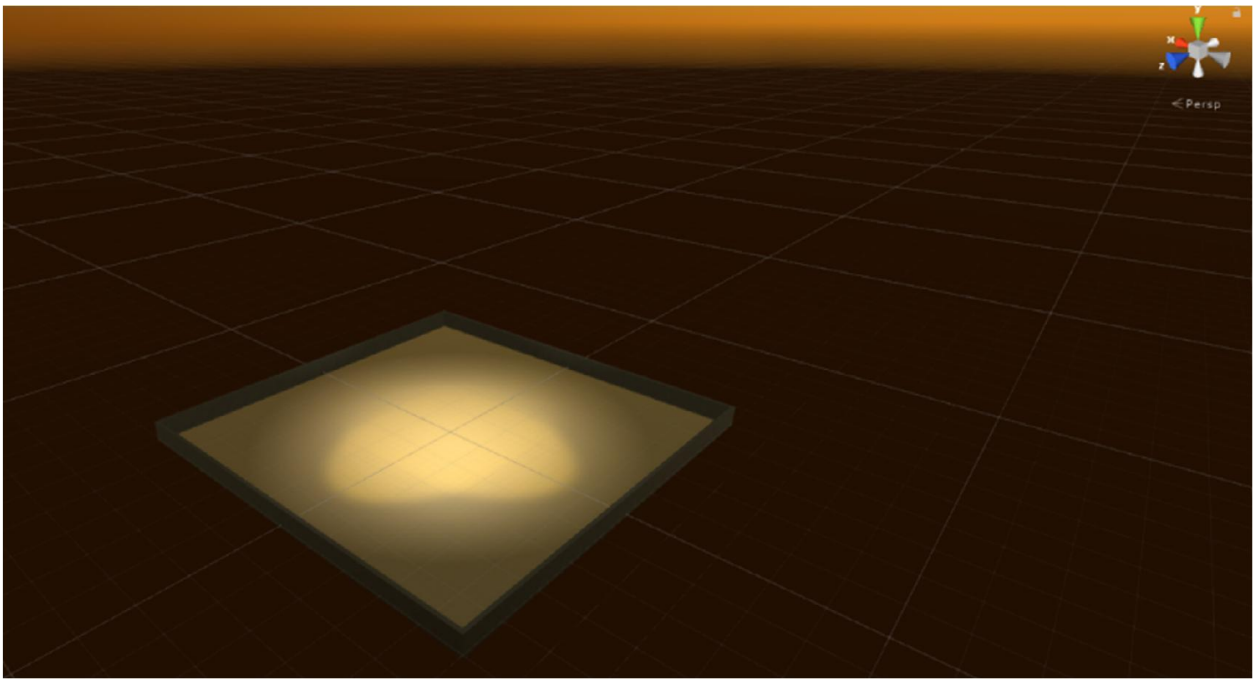

Рисунок 23 – Поле для ботов

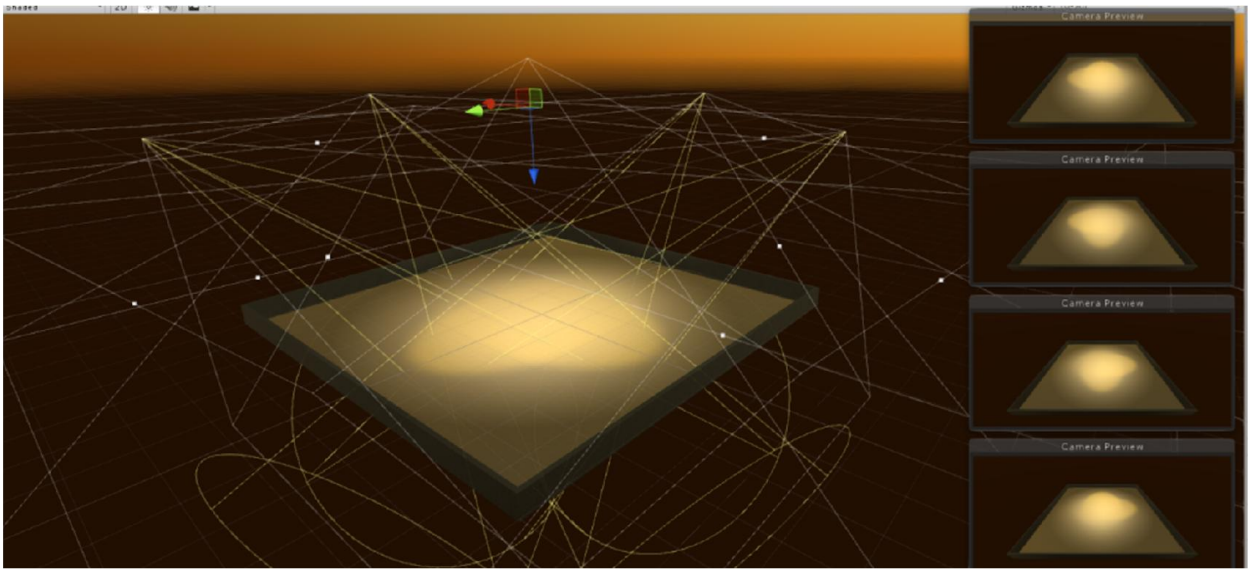

Рисунок 24 – Камеры и зоны их видимости

Каждая камера содержит скрипты Rotate (вращения), Zoom (приближение) и Light (подстветка) для управления. Для примера код `CameraRotate`.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraRotate : MonoBehaviour {

    public float _rotateX = 0;
    public float _rotateY = 0;

    public float _speedRotateX = 5f;
    public float _speedRotateY = 5f;

    public float _rotateMin = -90;
    public float _rotateMax = 90;

    [SerializeField] private GameObject StatusCursor;
    private LockCursor _statusCursor;
    // Use this for initialization
    void Start() {
     _statusCursor = StatusCursor.GetComponent<LockCursor>();
     Rigidbody body = GetComponent<Rigidbody>();
     if (body != null)
      body.freezeRotation = true;
    }

    // Update is called once per frame
    void Update() {
     if (!_statusCursor._ctrlTrigger) {
      _rotateX -= Input.GetAxis( "Mouse Y" ) * _speedRotateY;

      _rotateX = Mathf.Clamp( _rotateX, _rotateMin, _rotateMax
);

      float delta = Input.GetAxis( "Mouse X" ) * _speedRotateX;
      _rotateY = transform.localEulerAngles.y + delta;

      transform.localEulerAngles = new Vector3( _rotateX,
_rotateY, 0 );
     }
    }
}
```

Управление осуществляется через контрольный блок, который содержит набор скриптов для управления интерфейсом, спавном ботов и органики представленных на рисунке 25.

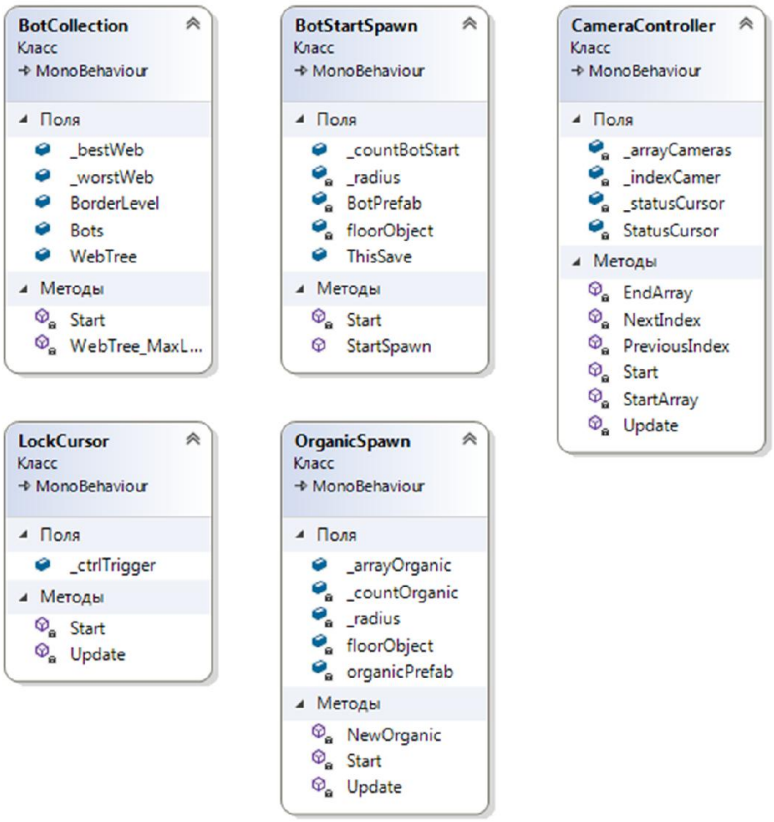

Рисунок 25 – Диаграмма классов контрольного блока

Освещения для фотосинтеза ботов создано точечным источником света, который меняет свое положения в начале расчета (рисунок 18).

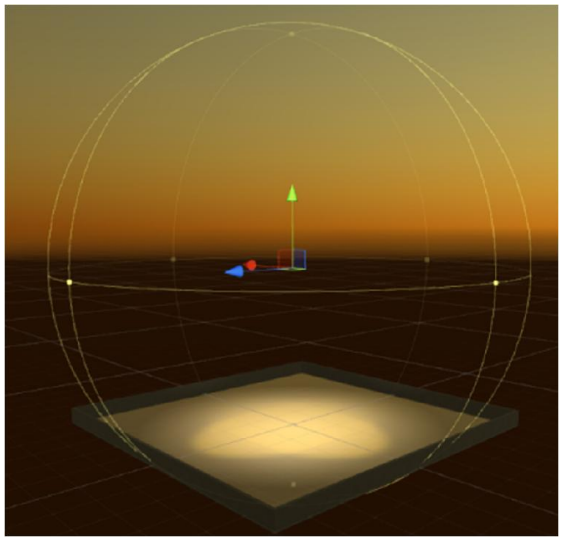

Рисунок 26 – «Солнце» мира ботов

Расчет освещенности бота происходит при каждом его шаге. Смысл расчёта — это модуль скалярного произведения между векторами положения бота и «солнца». Чем меньше это значение, тем сильнее освещен бот.

Интерфейс программы содержит список хеш имен ботов, где первая буква — это код поколения. Также содержит кнопки старта – паузы, загрузки и сохранения состояния приложения и информацию о номере поколения, максимального возраста бота и количества ботов, рисунок 27.

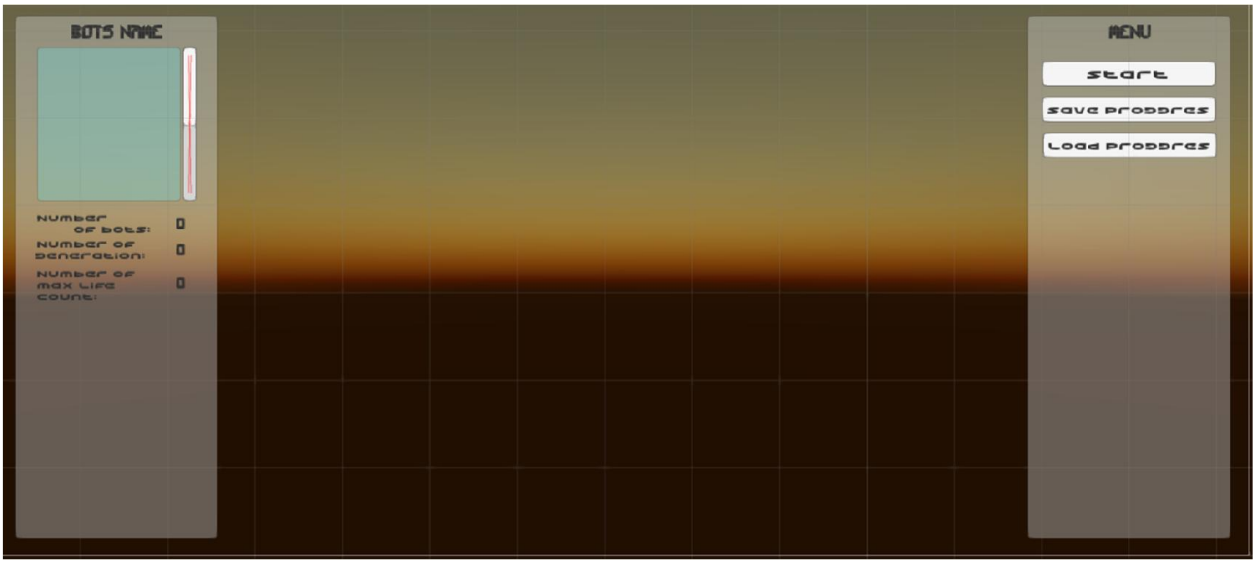

Рисунок 27 – GUI интерфейс приложения

На рисунках 28 – 30 представлена работа программы.

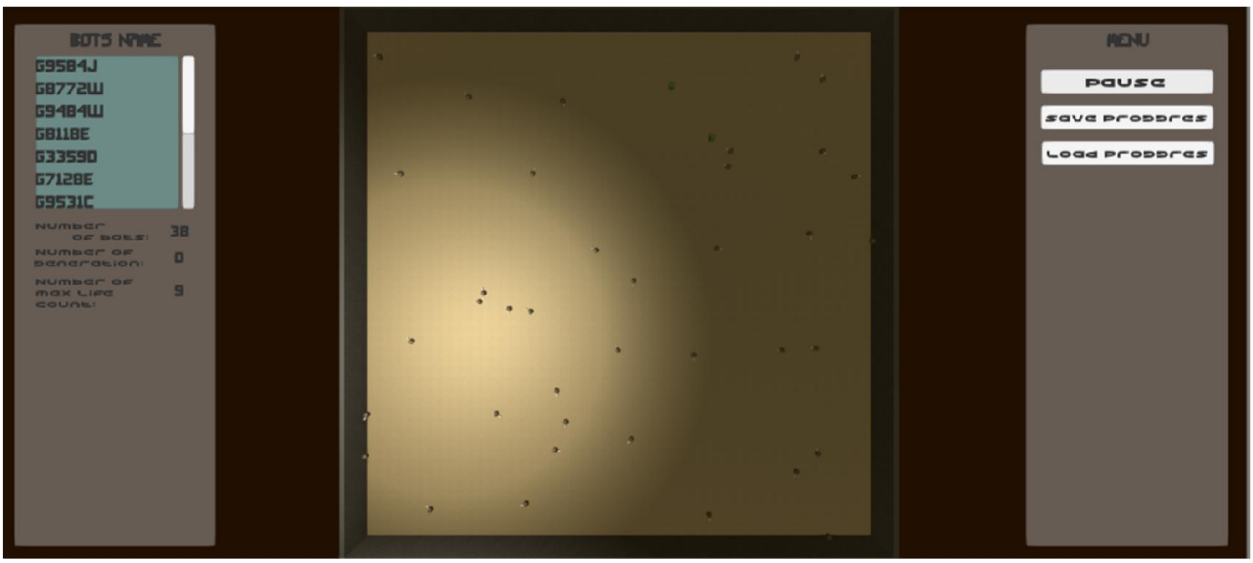

Рисунок 28 – Первый расчет в приложении

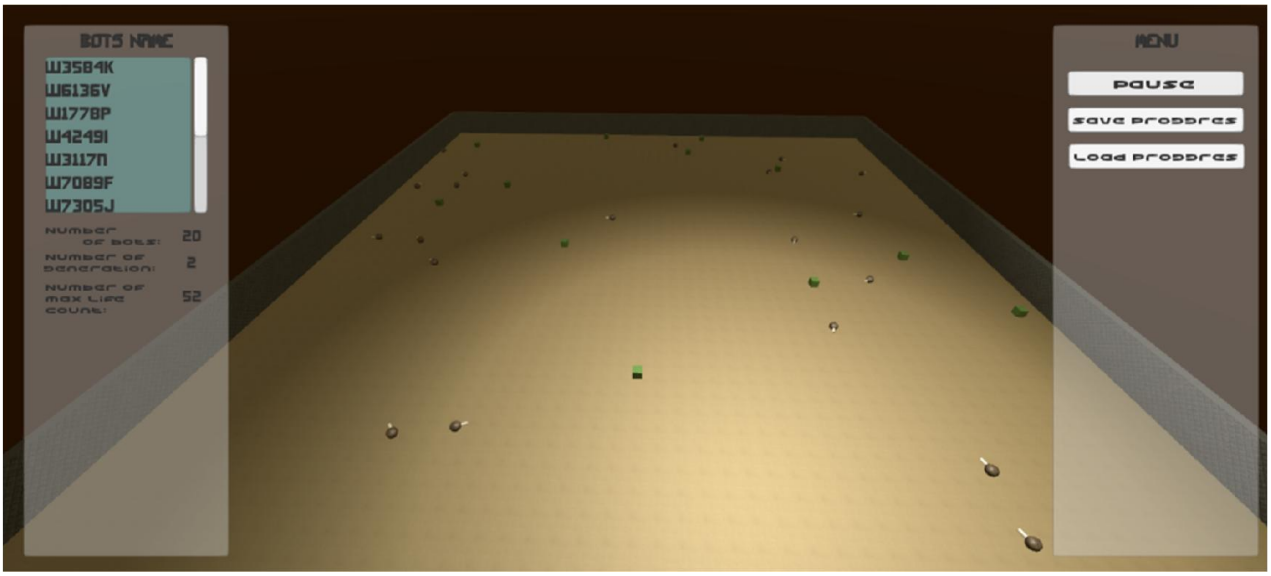

Рисунок 29 – Приближение с камеры

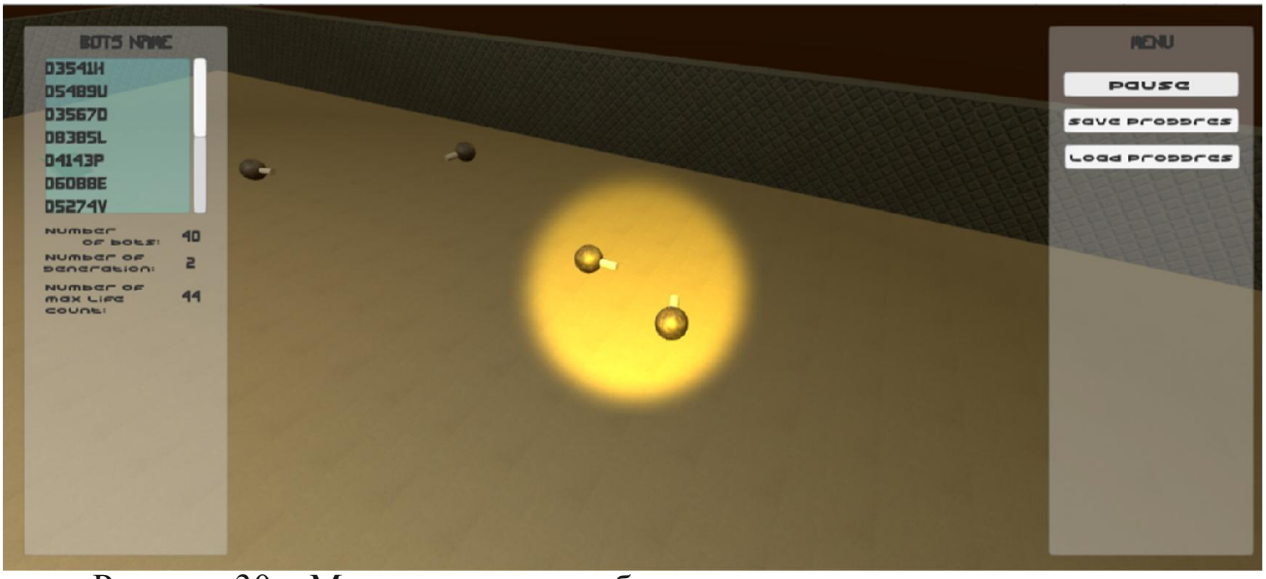

Рисунок 30 – Максимальное приближение и включение освещения

ВЫВОД

Подводя итоги можно сказать: выполнено изучение теории по нейронным сетям, генетическому алгоритму и движку unity. На текущий момент не удалось полностью реализовать задачи по сохранению, сбору и обработке статистики, часть визуальной составляющей. Разработана архитектура нейронной сети, применение генетического алгоритма. Написаны скрипты работы ботов, спавна органики управления и GUI интерфейс.

Далее планируется оптимизировать расчеты, разработать методы сохранения и загрузки данных. Разработать более наглядный GUI интерфейс, графики жизни ботов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- Андрейчиков А. В., Андрейчикова О. Н. Системный анализ и синтез стратегических решений в инноватике. Математические, эвристические и интеллектуальные методы системного анализа и синтеза инноваций. Учебное пособие; Ленанд - М., 2015. - 306 с.
- Редько В. Г. Моделирование когнитивной эволюции. На пути к теории эволюционного происхождения мышления; Ленанд - М., 2015. - 256 с.
- Тархов Д. А. Нейронные сети. Модели и алгоритмы. Книга 18; Радиотехника - М., 2012. - 256 с.
- Люгер Дж. Ф. Искусственный интеллект: стратегии и методы решения сложных проблем = Artificial Intelligence: Structures and Strategies for Complex Problem Solving / Люгер Дж. Ф., Под ред. Н. Н. Куссуль. - 4-е изд. - Москва: Вильямс, 2005. - 864 с.
- cjmaxik4 Симуляция жизни в системе Darwinbots. I. Первое знакомство. января 2013 URL: https://habr.com/post/164711/
- И. Рузмайкиной. Unity в действии. Мультиплатформенная разработка на C#/ Пер. с англ. – СПБ.: Питер, 2016. – 336 с.: ил. – (Серия «Для профессионалов»).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А ABSTRACTNERIALCOMPONENT

```
namespace NeiralNet {
     ///<sumarry>
     ///Абстрактный класс, для компонетов нейронной сети
     ///</sumarry>
     abstract class NeuralComponets {
      ///<sumarry>
      ///Матрица весов компонента
      ///</sumarry>
      protected float [,] _matrix;
      ///<summary>
      ///Размеры матрицы
      ///</summary>
      protected int _n, _m;
      ///<summary>
      ///Чтение кол-ва строк
      ///</summary>
      public int N {
        get { return _n; }
      }
      ///<summary>
      ///Чтение кол-ва столбов
      ///</summary>
      public int M {
        get { return _m; }
      }
      /// <summary>
      ///  Чтение матрицы весов
      /// </summary>
      public float [,] Matrix {
        get { return _matrix; }
        set { _matrix = value; }
      }
     }
}
```

ПРИЛОЖЕНИЕ В SYNAPSELAYER

```
using System;

namespace NeiralNet {
     /// <summary>
     /// Класс синапса нейросети
     /// </summary>
     class SynapseLayer : NeuralComponets {
      ///<summary>
      ///Конструктор с заданием кол-во строк и кол-ва столбов
      ///</summary>
      public SynapseLayer(int n, int m) {
        System.Random ForSpace = new System.Random( DateTime.Now.Millisecond +
DateTime.Now.Second + DateTime.Now.Minute + DateTime.Now.Hour);
        _m = m;
        _n = n;
        Matrix = new float [ _n, _m ];
        for (int i = 0 ; i < _n ; ++i)
```

```
          for (int j = 0 ; j < _m ; ++j)
            _matrix [ i, j ] = ForSpace.Next( -10, 10 ) ;
        }
        ///<summary>
        ///Метод для формирования синапсов через наследования ботов
        ///</summary>
        public SynapseLayer(float [,] baseMatrix, int trainigCoof, bool
triggerTrain) {
         System.Random ForSpace = new System.Random( DateTime.Now.Millisecond +
DateTime.Now.Second + DateTime.Now.Minute + DateTime.Now.Hour );
         _m = baseMatrix.GetLength( 1 );
         _n = baseMatrix.GetLength( 0 );
         Matrix = new float [ _n, _m ];

         for (int i = 0 ; i < _n ; ++i)
           for (int j = 0 ; j < _m ; ++j) {
             _matrix [ i, j ] = baseMatrix [ i, j ];
             if (triggerTrain) {
               _matrix[i,j] += ForSpace.Next( -trainigCoof, trainigCoof );
             }
           }
        }
      }
}
```

ПРИЛОЖЕНИЕ С OUTPUTLAYER

```
using System;

namespace NeiralNet {
      /// <summary>
      /// Класс, выходного слоя нейросети
      /// </summary>
      class OutputLayer : NeuralComponets {
        ///<summary>
        ///Конструктор с заданием кол-ва столбов
        ///</summary>
        public OutputLayer(int m) {
          _m = m;
          _n = 1;
          _matrix = new float [ _n, _m ];
        }
        ///<summary>
        ///Фомирование хеш-строки из полученных значений
        ///</summary>
        public float[,] OutputDate() {
          return _matrix;
        }
        ///<summary>
        ///Прогон всех значений через активационную функцию
        ///</summary>
        public void Formalize() {
          for (int j = 0 ; j < _m ; ++j)
            _matrix [ 0, j ] = (float)Function.Sigmoid( _matrix [ 0, j ] );
        }
      }
```

```
}
```

ПРИЛОЖЕНИЕ D NEURALNETWORK

```
using System;

namespace NeiralNet {
     /// <summary>
     /// Класс перцетрона
     /// </summary>
     public class NeuralNetwork : Tree.IKnot{
      #region Состав сети
      /// <summary>
      /// Входной слой
      /// </summary>
      private InputLayer _first;
      /// <summary>
      /// Скрытый слой
      /// </summary>
      private HiddenLayer _second;
      /// <summary>
      /// Выходной слой
      /// </summary>
      private OutputLayer _third;
      /// <summary>
      ///  Слой синапсов
      /// </summary>
      private SynapseLayer _sFirst;
      /// <summary>
      ///  Слой синапсов
      /// </summary>
      private SynapseLayer _sSecond;
      #endregion

      /// <summary>
      /// Коофициент
      /// обучения </summary>
      private int _trainigCoof;

      public float ValueCrit;
      ///<summary>
      /// Конструктор для наследования
      ///</summary>
      public NeuralNetwork(int trainigCoof, float [,] sFirst, float [,]
sSecond) {
       _first = new InputLayer( 6 );

       _second = new HiddenLayer( 2 );

       _third = new OutputLayer( 6 );

       _sFirst = new SynapseLayer( 6, 2 );

       _sSecond = new SynapseLayer( 2, 6 );

       this._sFirst = new SynapseLayer( sFirst, trainigCoof, true );
```

```
        this._sSecond = new SynapseLayer( sSecond, trainigCoof, true );

        this._trainigCoof = trainigCoof;
        ValueCrit = 0;
      }
      /// <summary>
      /// Конструктор с параметром обучения нейросети
      /// </summary>
      public NeuralNetwork(int trainigCoof) {
        _first = new InputLayer( 6 );

        _second = new HiddenLayer( 2 );

        _third = new OutputLayer( 6 );

        _sFirst = new SynapseLayer( 6, 2 );

        _sSecond = new SynapseLayer( 2, 6 );

        this._trainigCoof = trainigCoof;
        ValueCrit = 0;
      }
      /// <sumarry>
      /// Конструктор для наследования нейросети
      /// </sumarry>
      public NeuralNetwork(NeuralNetwork baseNet, bool triggerTrain) {
        _first = new InputLayer( 6 );

        _second = new HiddenLayer( 2 );

        _third = new OutputLayer( 6 );

        _sFirst = new SynapseLayer( 6, 2 );

        _sSecond = new SynapseLayer( 2, 6 );

        _trainigCoof = baseNet.TraingCoof;

        _sFirst = new SynapseLayer( baseNet.SFirst, baseNet.TraingCoof,
triggerTrain );

        _sSecond = new SynapseLayer( baseNet.SSecond, baseNet.TraingCoof,
triggerTrain );
        ValueCrit = 0;

      }
      ///<sumarry>
      ///Метод работы нейросети, с результатом хеш-строки
      ///</sumarry>
      public float[,] WorkNet(float [] hashArray) {
        _first.InputDate( hashArray );

        _second.Matrix = Matrix.Multiplication( _first.Matrix, _sFirst.Matrix
);
```

```
        _second.Formalize();

        _third.Matrix = Matrix.Multiplication( _second.Matrix, _sSecond.Matrix
);
        _third.Formalize();

        return _third.OutputDate();
      }

      ///<sumarry>
      ///Коофициент обучения
      ///</sumarry>
      public int TraingCoof {
        get { return _trainigCoof; }
      }
      ///<sumarry>
      ///Свойства для наследования
      ///</sumarry>
      public float [,] SFirst {
        get { return _sFirst.Matrix; }
      }
      ///<sumarry>
      ///Свойства для наследования
      ///</sumarry>
      public float [,] SSecond {
        get { return _sSecond.Matrix; }
      }

      static public NeuralNetwork Mixing(NeuralNetwork web1, NeuralNetwork
web2) {
        SynapseLayer first = new SynapseLayer( web1.SFirst.GetLength( 0 ),
web1.SFirst.GetLength( 1 ) );
        SynapseLayer second = new SynapseLayer( web1.SSecond.GetLength( 0 ),
web1.SSecond.GetLength( 1 ) );

        for (int i = 0 ; i < first.N ; ++i) {
          for (int j = 0 ; j < first.M ; ++j) {
            first.Matrix [ i, j ] = ( web1.SFirst [ i, j ] + web2.SFirst [ i, j
] ) / 2f;
          }
        }

        for (int i = 0 ; i < second.N ; ++i) {
          for (int j = 0 ; j < second.M ; ++j) {
            second.Matrix [ i, j ] = ( web1.SSecond [ i, j ] + web2.SSecond [
i, j ] ) / 2f;
          }
        }

        return new NeuralNetwork( web1.TraingCoof, first.Matrix, second.Matrix
);
      }

      public float CriticalValues() {
        return ValueCrit;
```

```
      }
     }
}
```

ПРИЛОЖЕНИЕ E MATRIX

```
using System;

namespace NeiralNet {
     /// <summary>
     /// Класс с методом умножения матриц
     /// </summary>
     static class Matrix {
      /// <summary> Умножение двух матриц </summary>
      /// <param name="A">ссылка на матрицу</param>
      /// <param name="B">ссылка на матрицу</param>
      /// <returns>матрица C = A * B</returns>
      static public float [,] Multiplication(float [,] A, float [,] B) {
       try {
        int Am = A.GetLength( 1 ), Bn = B.GetLength( 0 ), An = A.GetLength(
0 ), Bm = B.GetLength( 1 );
         if (Am != Bn)
          throw new Exception();
         float [,] C = new float [ An, Bm ];
         for (int i = 0 ; i < An ; ++i)
          for (int j = 0 ; j < Bm ; ++j)
           for (int k = 0 ; k < Am ; ++k) {
            C [ i, j ] += A [ i, k ] * B [ k, j ];
           }
         return C;
       }
       catch {
        return null;
       }
      }
     }
}
```

ПРИЛОЖЕНИЕ F INPUTLAYER

```
using System;

namespace NeiralNet {
     /// <summary>
     /// Класс, входного слоя нейросети
     /// </summary>
     class InputLayer : NeuralComponets {
      ///<summary>
      ///Конструктор, с заданием кол-ва столбов
      ///</summary>
      public InputLayer(int m) {
       _m = m;
       _n = 1;
       _matrix = new float[ _n, _m ];
      }
      ///<summary>
      ///Чтение из хеш-строки действий
      ///</summary>
```

```
      public void InputDate(float [] hashArray) {
        for (int j = 0; j < _m; ++j)
          Matrix[ 0, j ] = hashArray[ j];
      }
    }
}
```

ПРИЛОЖЕНИЕ G HIDDENLAYER

```
using System;

namespace NeiralNet {
    /// <summary>
    /// Класс, скрытого слоя нейросети
    /// </summary>
    class HiddenLayer : NeuralComponets {
      ///<summary>
      ///Конструктор, с заданием кол-ва столбов
      ///</summary>
      public HiddenLayer(int m) {
        _m = m;
        _n = 1;
        _matrix = new float [ _n, _m ];
      }
      ///<summary>
      ///Прогнать все значения через активационную функкцию
      ///</summary>
      public void Formalize() {
        for (int j = 0 ; j < _m ; ++j)
          _matrix [ 0, j ] = (float)Function.Sigmoid( _matrix [ 0, j ] );
      }
    }
}
```

ПРИЛОЖЕНИЕ H FUNCTION

```
using System;

namespace NeiralNet {
    ///<summary>
    ///Класс, содержащий активационную функцию и распределение гаусса
    ///</summary>
    static class Function {
      /// <summary>
      /// Распределение гаусса
      /// </summary>
      static public double Gauss(double x) {
        return new Random((int)DateTime.Now.Ticks).NextDouble();
      }
      /// <summary>
      ///  Активационая функция
      /// </summary>
      static public double Sigmoid(double x) {
        return 1.0 / ( 1.0 + Math.Exp( -0.1 * x ) );
      }
    }
}
```

ПРИЛОЖЕНИЕ I TREE

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections;
using UnityEngine;


namespace Tree {
     [Serializable]
     public class Tree<Template> {
      private Knot<Template> _root;
      public int MaxLevel { get; set; }
      private int _borderLevel;
      public Tree(Template value, int borderLevel) {
        _root = new Knot<Template>( value, 0 );
        MaxLevel = 0;
        _borderLevel = borderLevel;
       }

      public void NewValue(Template value) {
        if (_root._value == null) {
          _root._value = value;
          return;
        }

        Knot<Template> valueKnot = _root;
        while (true) {
          if (( (IKnot)value ).CriticalValues() >= valueKnot.CriticalValues())
{
            if (valueKnot.RightKnot == null) {
              valueKnot.RightKnot = new Knot<Template>( value, valueKnot.Level
+ 1 );
              if (MaxLevel < valueKnot.Level + 1)
                MaxLevel = valueKnot.Level + 1;
              break;
            }
            else {
              valueKnot = valueKnot.RightKnot;
            }
          }
          else {
            if (valueKnot.LeftKnot == null) {
              valueKnot.LeftKnot = new Knot<Template>( value, valueKnot.Level +
1 );
              if (MaxLevel < valueKnot.Level + 1)
                MaxLevel = valueKnot.Level + 1;
              break;
            }
            else {
              valueKnot = valueKnot.LeftKnot;
            }
          }
```

```
        }

        if (MaxLevel == _borderLevel) {
          if (MaxLevelReach != null)
            MaxLevelReach();
        }
      }

      public Template RootValue {
        get {
          return _root._value;
        }
        set {
          _root._value = value;
        }
      }

      public Template MaxRightValue() {
        Knot<Template> valueKnot = _root;
        while (true) {
          if (valueKnot.RightKnot != null) {
            valueKnot = valueKnot.RightKnot;
          }
          else {
            return valueKnot._value;
          }
        }
      }

      public Template MaxLeftValue() {
        Knot<Template> valueKnot = _root;
        while (true) {
          if (valueKnot.LeftKnot != null) {
            valueKnot = valueKnot.LeftKnot;
          }
          else {
            return valueKnot._value;
          }
        }
      }

      public delegate void baseMethod();

      public event baseMethod MaxLevelReach;
    }
}
```

ПРИЛОЖЕНИЕ J KNOT

```
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections;
using UnityEngine;
using System;
```

```
namespace Tree {
     [Serializable]
     class Knot<Template> :IKnot{
      public Template _value { get; set; }
      private Knot<Template> _rightKnot;
      private Knot<Template> _leftKnot;
      public int Level { get; set; }

      public Knot(Template baseValue,int newLevel) {
        _value = baseValue;
        _rightKnot = null;
        _leftKnot = null;
        Level = newLevel;
       }

      public Knot<Template> RightKnot {
        get { return _rightKnot; }
        set { _rightKnot = value; }
       }

      public Knot<Template> LeftKnot {
        get { return _leftKnot; }
        set { _leftKnot = value; }
       }

      public float CriticalValues() {
        return ((IKnot)_value).CriticalValues();
       }
     }
}
```

ПРИЛОЖЕНИЕ К IKNOT

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Tree {
     interface IKnot {
      float CriticalValues();
     }
}
```

ПРИЛОЖЕНИЕ L SPECIFICATION

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;
using NeiralNet;

public class Specifications : MonoBehaviour {

     public int LifeTime = 0;

     public string Name = string.Empty;

     public float Speed = 10f;
```

```
public float RotationAngle = 45;

public NeiralNet.NeuralNetwork Web;

private GameObject _sun;

public float Energy {
 set {
  if (value > 100) {
   _energy = 100;
   return;
  }
  _energy = value;
 }
 get { return _energy; }
}

public float Photosensitivity = 50f;

public float RadiusAction = 2f;

public delegate void baseMod(GameObject obj);

public event baseMod EventDie;

public int TrainCoof = 2;

public float LightLevel = 0;
public float PossibilityMove = 0;
public float AvailabilityFood = 0;
public float BreedingOpportunity = 0;
private float _energy =50;
private float _hp = 100;

public float HP {
 get { return _hp; }
 set {
  if (value < 0) {
   if (EventDie!=null)
    EventDie( this.gameObject );
   return;
  }
  if (value > 100)
   _hp = 100;
  else
   _hp = value;
 }
}

private void Start() {
 _sun = GameObject.FindWithTag("Sun");
 Web = new NeuralNetwork( TrainCoof );
 StartCoroutine( HPUpdateEnrgy() );
 StartCoroutine(LookAround());
```

```
    }

    private IEnumerator HPUpdateEnrgy() {
     if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
       HP += (float)( -0.0045 * _energy * _energy + 1.152 * _energy );

       yield return new WaitForSeconds( 5f );
      }
      else {
       yield return new WaitForSeconds( 0.1f );
      }

      StartCoroutine( HPUpdateEnrgy() );
    }

    private IEnumerator LookAround() {
     if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
       LightLevel = ( _sun.transform.position - gameObject.transform.position
).magnitude / _sun.transform.position.y;
       try {
        LightLevel = Mathf.Sqrt( (float)( -0.42 * ( LightLevel - 2.2 ) ) );
        if (LightLevel != LightLevel)
         throw new System.Exception();
       }
       catch {
        LightLevel = 0;
       }

       PossibilityMove = 1;
       Collider [] BotColl = Physics.OverlapSphere( transform.position,
RadiusAction );
       foreach (Collider thisObject in BotColl) {
        Vector3 direction = thisObject.transform.position -
transform.position;
        if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
         if (thisObject.tag != "Space" || thisObject.tag == "Wall") {
          PossibilityMove = 0;
          break;
         }
        }
       }

       AvailabilityFood = 0;
       Collider [] EatColl = Physics.OverlapSphere( transform.position,
RadiusAction );
       foreach (Collider thisObject in EatColl) {
        Vector3 direction = thisObject.transform.position -
transform.position;
        if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
         if (thisObject.tag == "Organic") {
          AvailabilityFood = 1;
          break;
         }
```

```
          }
        }

        if (_energy >= 70) {
          BreedingOpportunity = 1;
        }
        else {
          BreedingOpportunity = 0;
        }

        StartMove();

        _hp -= 5;

        if (GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.MaxLifeCount < LifeTime)
          GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.MaxLifeCount = LifeTime;

        yield return new WaitForSeconds( 0.1f );
      }
      else {
        yield return new WaitForSeconds( 0.1f );
      }
      StartCoroutine( LookAround() );
    }

    private void StartMove() {
      float[] hashArray = new float[6];

      hashArray[0] = LightLevel;
      hashArray [1]= PossibilityMove;
      hashArray [2]= AvailabilityFood;
      hashArray [3]= BreedingOpportunity;
      hashArray [4]= _energy;
      hashArray [5]= _hp;

      float[,] outValue = Web.WorkNet( hashArray );

      float maxValue = -2;
      int maxIndex = 0;

      for (int i = 0 ; i < 6 ;++i) {
        if (maxValue < outValue [ 0, i ]) {
          maxValue = outValue [ 0, i ];
          maxIndex = i;
        }
      }

      switch (maxIndex){
        case 0: {
          this.gameObject.GetComponent<MoveForward>().Move();
          break;
        }
        case 1: {
```

```
            this.gameObject.GetComponent<RotationBody>().Rotation( maxValue );
            break;
           }
           case 2: {
            this.gameObject.GetComponent<Eat>().EatMove();
            break;
           }
           case 3: {
            this.gameObject.GetComponent<Fotosintez>().MoveFotosintez();
            break;
           }
           case 4: {
            this.gameObject.GetComponent<Reproduction>().MoveReproduction();
            break;
           }
           case 5: {
            this.gameObject.GetComponent<Attack>().AttacMove();
            break;
           }
         }
         Web.ValueCrit++;
         LifeTime++;
       }
}
```

ПРИЛОЖЕНИЕ M ROTATIONBODY

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotationBody : MonoBehaviour {
     /// <summary>
     /// object reference with characteristics
     /// </summary>
     [SerializeField] Specifications Stat;

     /// <summary>
     /// Rotation bot
     /// </summary>
     public void Rotation(float value) {
       transform.Rotate( 0, value * Stat.RotationAngle, 0 );
     }
}
```

ПРИЛОЖЕНИЕ N REPRODUCTION

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using NeiralNet;

public class Reproduction : MonoBehaviour {
     /// <summary>
     /// Object reference with characteristics
     /// </summary>
     [SerializeField] Specifications Stat;
     /// <summary>
```

```
    /// Template bot
    /// </summary>
    [SerializeField] GameObject BotPrafab;
    /// Specification reproduct
    private float _priceRep = 70f;
    private float _randZoneMin = -3;
    private float _randZoneMax = 3;

    public void MoveReproduction() {

     if (Stat.Energy > _priceRep) {
       Stat.Energy -= _priceRep;
       Debug.Log( "Reproduction => 1" );
       GameObject newBot = Instantiate( BotPrafab ) as GameObject;
       newBot.transform.position = this.gameObject.transform.position;
       newBot.transform.position = new Vector3( newBot.transform.position.x,
newBot.transform.position.y, newBot.transform.position.z );
       newBot.transform.parent = GameObject.Find( "Floor" ).transform;
       newBot.transform.localScale = this.gameObject.transform.localScale;
       newBot.GetComponent<Specifications>().Name = char.ConvertFromUtf32(
Random.Range( 65, 91 ) ) + Random.Range( 1000, 10000 ).ToString() +
char.ConvertFromUtf32( Random.Range( 65, 91 ) );
       Mutation( newBot.GetComponent<Specifications>() );
       GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots.Add( newBot );
     }
     else
     {
       Stat.Web.ValueCrit--;
       Stat.HP -= 10;
     }
    }
    /// <summary>
    /// Parameter mutation
    /// </summary>
    private void Mutation(Specifications baseStat) {
     baseStat.Photosensitivity = Stat.Photosensitivity + Random.Range(
_randZoneMin, _randZoneMax );
     baseStat.Speed = Stat.Speed + Random.Range( _randZoneMin, _randZoneMax
);
     baseStat.RotationAngle = Stat.RotationAngle + Random.Range(
_randZoneMin, _randZoneMax );
     baseStat.Energy = 50;
     baseStat.HP = 100;
     baseStat.Web = new NeiralNet.NeuralNetwork( Stat.Web, true );
}
```

ПРИЛОЖЕНИЕ О MOVEFORWARD

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveForward : MonoBehaviour {
    /// <summary>
    /// object reference with characteristics
```

```
     /// </summary>
     [SerializeField] Specifications Stat;
     private float _distance;

     /// <summary>
     ///  Move bot forward
     /// </summary>
     public void Move() {
      _distance = 0;
      ///Check border field
      if (490 <= Stat.Speed + transform.position.x || -490 >= Stat.Speed +
transform.position.x || 490 <= Stat.Speed + transform.position.z || -490 >=
Stat.Speed + transform.position.z) {
        ///Fine network
        Stat.Web.ValueCrit--;
        Stat.HP -= 10;
        return;
       }

      StartCoroutine( MoveAnimation() );
     }

     /// <summary>
     /// Animation parallel flow
     /// </summary>
     IEnumerator MoveAnimation() {
      if (_distance < Stat.Speed) {
        if (490 <= transform.position.x || -490 >= transform.position.x || 490
<= transform.position.z || -490 >= transform.position.z) {
         StopCoroutine( MoveAnimation() );
         yield return null;
        }
        else {
         _distance += Stat.Speed / 20f;
         transform.Translate( Stat.Speed / 20f, 0, 0 );

         yield return new WaitForSeconds( 0.01f );
         StartCoroutine( MoveAnimation() );
        }
      }
      else {
        StopCoroutine( MoveAnimation() );
      }
     }
}
```

ПРИЛОЖЕНИЕ Р FOTOSINTEZ

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Fotosintez : MonoBehaviour {
     /// <summary>
     /// Object reference with characteristics
     /// </summary>
```

```
     [SerializeField] Specifications Stat;

     /// <summary>
     /// Food photosynthesis
     /// </summary>
     public void MoveFotosintez() {
      Stat.Energy += Stat.LightLevel * Stat.Photosensitivity;
     }
}
```

ПРИЛОЖЕНИЕ Q EAT

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Eat : MonoBehaviour {
     /// <summary>
     ///  Object reference with characteristics
     /// </summary>
     [SerializeField] Specifications Stat;

     /// <summary>
     /// Eat organic
     /// </summary>
     public void EatMove() {
      /// Find object around
      Collider [] EatColl = Physics.OverlapSphere( transform.position,
Stat.RadiusAction );
       foreach (Collider thisObject in EatColl) {
        Vector3 direction = thisObject.transform.position -
transform.position;
        ///  True distance
        if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
         if (this.tag == "Organic") {
           Stat.Energy +=
thisObject.GetComponent<SpecificationsOrganic>().val;
           thisObject.GetComponent<EatOrganic>().MoveEat();
           return;
         }
        }
       }
       /// Find web
       Stat.Web.ValueCrit--;
       Stat.HP -= 10;
     }
}
```

ПРИЛОЖЕНИЕ R DIE

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Die : MonoBehaviour {
     [SerializeField] Specifications Stat;

     private void Start() {
```

```
      Stat.EventDie += Stat_EventDie;
     }

     private void Stat_EventDie(GameObject obj) {
      Debug.Log( "Die" );
      if (GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots.Count == 1) {
       Debug.Log( "New Generation" );
       GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.GenerationCount++;
       StartCoroutine( GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().StartSpawn() );
      }
      GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots.Remove( this.gameObject );
      GameObject.Find( "Controller"
).GetComponent<BotCollection>().WebTree.NewValue(this.gameObject.GetComponent
<Specifications>().Web );
      Destroy(this.gameObject);
     }
}
```

ПРИЛОЖЕНИЕ S ATTACK

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Attack : MonoBehaviour {
     /// <summary>
     ///  Object reference with characteristics
     /// </summary>
     [SerializeField] Specifications Stat;

     /// <summary>
     /// Attac bots
     /// </summary>
     public void AttacMove() {
      Collider [] BotColl = Physics.OverlapSphere( transform.position,
Stat.RadiusAction );
      foreach (Collider thisObject in BotColl) {
       Vector3 direction = thisObject.transform.position -
transform.position;
       if (Vector3.Dot( transform.forward, direction ) > 0.5f) {
        if (this.tag == "Bot") {
         try {
          Stat.Energy += thisObject.GetComponent<Specifications>().HP;
          thisObject.GetComponent<Specifications>().HP = 0;
         }
         catch { }
         return;
        }
       }
      }
      /// Find web
      Stat.Web.ValueCrit--;
```

```
      Stat.HP -= 10;
     }
}
```

ПРИЛОЖЕНИЕ Т CAMERAZOOM

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraZoom : MonoBehaviour {

     private float _speed = 50f;
     private float _minValue = 100f;

     [SerializeField] private GameObject StatusCursor;
     private LockCursor _statusCursor;
     struct AxesProperites {
      public float xMax;
      public float xSign;

      public float yMax;
      public float ySign;

      public float zMax;
      public float zSign;
     }

     AxesProperites _datePosition;
     // Use this for initialization
     void Start () {
      _statusCursor = StatusCursor.GetComponent<LockCursor>();

      try {
       _datePosition.xSign = this.gameObject.transform.position.x /
Mathf.Abs( this.gameObject.transform.position.x );
      }
      catch {
       _datePosition.xSign = 0;
      }
      _datePosition.xMax = this.gameObject.transform.position.x + (100*
_datePosition.xSign );


      try {
       _datePosition.ySign = this.gameObject.transform.position.y /
Mathf.Abs( this.gameObject.transform.position.y );
      }
      catch {
       _datePosition.ySign = 0;
      }
      _datePosition.yMax = this.gameObject.transform.position.y + ( 100 *
_datePosition.ySign );

      try {
```

```
        _datePosition.zSign = this.gameObject.transform.position.z /
Mathf.Abs( this.gameObject.transform.position.z );
       }
       catch {
        _datePosition.zSign = 0;
       }
       _datePosition.zMax = this.gameObject.transform.position.z + ( 100 *
_datePosition.zSign );
     }

     // Update is called once per frame
     void Update() {
      if (!_statusCursor._ctrlTrigger) {
       float axesValue = Input.GetAxis( "Mouse ScrollWheel" );
       if (axesValue != 0) {
        Camera obj = GetComponent<Camera>();
        if (obj != null) {
         obj.transform.position = ShiftVector( obj.transform.position,
axesValue * _speed );
        }
       }
      }
     }

     private Vector3 ShiftVector(Vector3 baseVector, float shiftValue) {
      Vector3 newVector = new Vector3( 0, 0, 0 );

      if (baseVector.x != 0) {
       if (Mathf.Abs( baseVector.x + ( shiftValue * _datePosition.xSign ) ) >
Mathf.Abs( _datePosition.xMax ) || Mathf.Abs( baseVector.x + ( shiftValue *
_datePosition.xSign ) ) <_minValue)
        return baseVector;
       newVector.x = baseVector.x + ( shiftValue * _datePosition.xSign );
      }
      if (baseVector.y != 0) {
       if (Mathf.Abs( baseVector.y + ( shiftValue * _datePosition.ySign ) ) >
Mathf.Abs( _datePosition.yMax ) || Mathf.Abs( baseVector.y + ( shiftValue *
_datePosition.ySign ) ) < _minValue)
        return baseVector;
       newVector.y = baseVector.y + ( shiftValue * _datePosition.ySign );
      }
      if (baseVector.z != 0) {
       if (Mathf.Abs( baseVector.z + ( shiftValue * _datePosition.zSign ) ) >
Mathf.Abs( _datePosition.zMax ) || Mathf.Abs( baseVector.z + ( shiftValue *
_datePosition.zSign ) ) < _minValue)
        return baseVector;
       newVector.z = baseVector.z + ( shiftValue * _datePosition.zSign );
      }

      return newVector;
     }
}
```

ПРИЛОЖЕНИЕ U LIGHTON

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;

public class LightOn : MonoBehaviour {
     [SerializeField] Light Flash;
     // Use this for initialization
     void Start () {
      Flash.gameObject.SetActive(false);
     }

     // Update is called once per frame
     void Update () {
      if (Input.GetKeyDown( KeyCode.L )) {
        Flash.gameObject.SetActive( !Flash .gameObject.active);
      }
     }
}
```

ПРИЛОЖЕНИЕ V CAMERAROTATE

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraRotate : MonoBehaviour {

     public float _rotateX = 0;
     public float _rotateY = 0;

     public float _speedRotateX = 5f;
     public float _speedRotateY = 5f;

     public float _rotateMin = -90;
     public float _rotateMax = 90;

     [SerializeField] private GameObject StatusCursor;
     private LockCursor _statusCursor;
     // Use this for initialization
     void Start() {
      _statusCursor = StatusCursor.GetComponent<LockCursor>();
      Rigidbody body = GetComponent<Rigidbody>();
      if (body != null)
        body.freezeRotation = true;
     }

     // Update is called once per frame
     void Update() {
      if (!_statusCursor._ctrlTrigger) {
        _rotateX -= Input.GetAxis( "Mouse Y" ) * _speedRotateY;

        _rotateX = Mathf.Clamp( _rotateX, _rotateMin, _rotateMax );

        float delta = Input.GetAxis( "Mouse X" ) * _speedRotateX;
        _rotateY = transform.localEulerAngles.y + delta;

        transform.localEulerAngles = new Vector3( _rotateX, _rotateY, 0 );
```

```
      }
     }
}
```

ПРИЛОЖЕНИЕ W ORGANICASPAWN

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OrganicSpawn : MonoBehaviour {
     [SerializeField] private GameObject organicPrefab;
     [SerializeField] private GameObject floorObject;
     public  List<GameObject> _arrayOrganic;

     private int _countOrganic = 10;
     private float _radius = 0.1f;

     // Use this for initialization
     void Start () {
      _arrayOrganic = new List<GameObject>();
      StartCoroutine( NewOrganic() );
     }

     // Update is called once per frame
     void Update () {
     }

     private IEnumerator NewOrganic() {
      if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
        if (_arrayOrganic.Count < 20) {
          for (int i = 0 ; i < _countOrganic / 5 ; ++i) {
            GameObject @object = Instantiate( organicPrefab ) as GameObject;
            @object.transform.position = new Vector3( Random.Range( -490, 490
), 10, Random.Range( -490, 490 ) );
           @object.transform.parent = floorObject.transform;
           _arrayOrganic.Add( @object );
           //добавить проверку на объекты вокруг
          }
        }
        yield return new WaitForSeconds( 3f );
       }
       else {
        yield return new WaitForSeconds( .1f );
       }
       StartCoroutine( NewOrganic() );
     }
}
```

ПРИЛОЖЕНИЕ X LOCKCURSOR

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LockCursor : MonoBehaviour {
```

```
    public bool _ctrlTrigger = false;
    // Use this for initialization
    void Start () {
     Cursor.lockState = CursorLockMode.Locked;
     Cursor.visible = false;
    }

    // Update is called once per frame
    void Update () {
     if (Input.GetKeyDown( KeyCode.LeftControl )) {
      _ctrlTrigger = true;
      Cursor.lockState = CursorLockMode.None;
      Cursor.visible = true;
     }

     if (Input.GetKeyUp( KeyCode.LeftControl )) {
      _ctrlTrigger = false;
      Cursor.lockState = CursorLockMode.Locked;
      Cursor.visible = false;
     }
    }
}
```

ПРИЛОЖЕНИЕ У CAMERACONTROL

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour {
    [SerializeField] Camera [] _arrayCameras;
    private int _indexCamer;

    [SerializeField] private GameObject StatusCursor;
    private LockCursor _statusCursor;

    private int NextIndex() {
     if (_indexCamer == _arrayCameras.Length - 1) {
      _indexCamer = 0;
     }
     else {
      ++_indexCamer;
     }
     return _indexCamer;
    }

    private int PreviousIndex() {
     if (_indexCamer == 0) {
      _indexCamer = _arrayCameras.Length - 1;
     }
     else {
      --_indexCamer;
     }
     return _indexCamer;
    }
```

```
    private int StartArray() {
     _indexCamer = 0;
     return _indexCamer;
    }

    private int EndArray() {
     _indexCamer = _arrayCameras.Length - 1;
     return _indexCamer;
    }

    // Use this for initialization
    void Start() {
     _statusCursor = StatusCursor.GetComponent<LockCursor>();
     _indexCamer = 0;
     foreach (Camera s in _arrayCameras) {
      s.gameObject.SetActive( false );
     }
     _arrayCameras [ _indexCamer ].gameObject.SetActive( true );
    }

    // Update is called once per frame
    void Update() {
     if (!_statusCursor._ctrlTrigger) {
      if (Input.GetKeyDown( KeyCode.RightArrow ) || Input.GetKeyDown(
KeyCode.D )) {
       _arrayCameras [ _indexCamer ].gameObject.SetActive( false );
       _arrayCameras [ NextIndex() ].gameObject.SetActive( true );
       goto end;
      }

      if (Input.GetKeyDown( KeyCode.LeftArrow ) || Input.GetKeyDown(
KeyCode.A )) {
       _arrayCameras [ _indexCamer ].gameObject.SetActive( false );
       _arrayCameras [ PreviousIndex() ].gameObject.SetActive( true );
       goto end;
      }

      if (Input.GetKeyDown( KeyCode.UpArrow ) || Input.GetKeyDown( KeyCode.W
)) {
       _arrayCameras [ _indexCamer ].gameObject.SetActive( false );
       _arrayCameras [ StartArray() ].gameObject.SetActive( true );
       goto end;
      }

      if (Input.GetKeyDown( KeyCode.DownArrow ) || Input.GetKeyDown(
KeyCode.S )) {
       _arrayCameras [ _indexCamer ].gameObject.SetActive( false );
       _arrayCameras [ EndArray() ].gameObject.SetActive( true );
       goto end;
      }

end:
      return;
     }
    }
```

```
}
```

ПРИЛОЖЕНИЕ Z BOTSTARTSPAWN

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BotStartSpawn : MonoBehaviour {
     [SerializeField] private GameObject BotPrefab;
     [SerializeField] private GameObject floorObject;

     public float _countBotStart = 40;
     private float _radius = 0.1f;
     public DateGame ThisSave;

     // Use this for initialization
     void Start () {
      StartCoroutine(StartSpawn());
      ThisSave = new DateGame();
     }

     public IEnumerator StartSpawn() {
      if (GameObject.Find( "StartButton"
).GetComponent<WorkBot>().TriggerWork) {
        string GenerationCode = char.ConvertFromUtf32( Random.Range( 65, 91 )
);
        for (int i = 0 ; i < _countBotStart ; ++i) {
          GameObject @object = Instantiate( BotPrefab ) as GameObject;
          @object.transform.position = new Vector3( Random.Range( -490, 490
),10, Random.Range( -490, 490 ) );
          @object.transform.Rotate(0,Random.Range(0,360),0);
          @object.transform.parent = floorObject.transform;
          this.gameObject.GetComponent<BotCollection>().Bots.Add( @object );
          if (GameObject.Find( "Controller"
).GetComponent<BotCollection>().WebTree.RootValue != null) {
           if (this.gameObject.GetComponent<BotCollection>()._bestWeb != null
&& this.gameObject.GetComponent<BotCollection>()._worstWeb != null) {
             if (_countBotStart * 0.1 >= i) {
               @object.GetComponent<Specifications>().Web = new
NeiralNet.NeuralNetwork( this.gameObject.GetComponent<BotCollection>()
._worstWeb,false);
             }
             if (_countBotStart * 0.3 >= i) {
               @object.GetComponent<Specifications>().Web = new
NeiralNet.NeuralNetwork(
this.gameObject.GetComponent<BotCollection>()._worstWeb, true );
             }
             if (_countBotStart * 0.5 >= i) {
               @object.GetComponent<Specifications>().Web = new
NeiralNet.NeuralNetwork(
this.gameObject.GetComponent<BotCollection>()._bestWeb, true );
             }
             if (_countBotStart * 0.8 >= i) {
```

```
            @object.GetComponent<Specifications>().Web = new
NeiralNet.NeuralNetwork(
this.gameObject.GetComponent<BotCollection>()._bestWeb, false );
           }
          }
          @object.GetComponent<Specifications>().Web = new
NeiralNet.NeuralNetwork( NeiralNet.NeuralNetwork.Mixing(
gameObject.GetComponent<BotCollection>().WebTree.MaxLeftValue(),
gameObject.GetComponent<BotCollection>().WebTree.MaxRightValue() ),false );
         }
         //добавить проверку на объекты вокруг

         @object.GetComponent<Specifications>().Name = GenerationCode +
Random.Range( 1000, 10000 ).ToString() + char.ConvertFromUtf32( Random.Range(
65, 91 ) );
        }
        StopCoroutine(StartSpawn());

      }
      else {
        yield return new WaitForSeconds( .1f );
        StartCoroutine( StartSpawn() );
      }
     }
}
```

ПРИЛОЖЕНИЕ АА BOTCOLLECTION

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BotCollection : MonoBehaviour {
     public List<GameObject> Bots;
     public Tree.Tree<NeiralNet.NeuralNetwork> WebTree;
     public int BorderLevel = 3;

     public NeiralNet.NeuralNetwork _bestWeb;
     public NeiralNet.NeuralNetwork _worstWeb;

     // Use this for initialization
     void Start () {
      Bots = new List<GameObject>();
      WebTree = new Tree.Tree<NeiralNet.NeuralNetwork>( null,BorderLevel );
      WebTree.MaxLevelReach += WebTree_MaxLevelReach;
      _bestWeb = null;
      _worstWeb = null;
     }

     private void WebTree_MaxLevelReach() {
      _bestWeb = WebTree.MaxRightValue();
      _worstWeb = WebTree.MaxLeftValue();
      WebTree = new Tree.Tree<NeiralNet.NeuralNetwork>(
NeiralNet.NeuralNetwork.Mixing( WebTree.MaxLeftValue(),
WebTree.MaxRightValue() ), BorderLevel );
     }
```

```
}
```

ПРИЛОЖЕНИЕ АВ WORKBOT

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class WorkBot : MonoBehaviour {

     public bool TriggerWork;
     private void Start() {
      TriggerWork = false;
      this.gameObject.transform.GetChild(0).GetComponent<Text>().text =
"Start";
     }

     public void OnClick() {
      TriggerWork = !TriggerWork;
      if (TriggerWork) {
       this.gameObject.transform.GetChild( 0 ).GetComponent<Text>().text =
"Stop";
      }
      else {
       this.gameObject.transform.GetChild( 0 ).GetComponent<Text>().text =
"Pause";
      }
     }
}
```

ПРИЛОЖЕНИЕ АС DATEGAME

```
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections;
using UnityEngine;

public class DateGame {
     public string Name;
     public List<GameObject> BotList { get; set; }
     public List<GameObject> OrganicList { get; set; }
     public GameObject Sun { get; set; }
     public int GenerationCount { get; set; }
     public int MaxLifeCount { get; set; }

     public DateGame() {
      BotList = null;
      OrganicList = null;
      Sun = null;
      Name = null;
      GenerationCount = 0;
     }

     public static DateGame Current;
}
```

ПРИЛОЖЕНИЕ AD BOTLISTNAMEUPDATE

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BotNameListUpdate : MonoBehaviour {
     string text = string.Empty;
     // Use this for initialization
     void Start() {

     }

     // Update is called once per frame
     void FixedUpdate() {
      text = string.Empty;
      //GameObject.Find( "ScrollbarLeft" ).GetComponent<Scrollbar>().value =
1;

      foreach (GameObject s in GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots) {
        text += s.GetComponent<Specifications>().Name + "\n";
      }
      GameObject.Find( "TextName" ).GetComponent<Text>().text = text;

      GameObject.Find( "CountBoxBot" ).GetComponent<Text>().text =
GameObject.Find( "Controller"
).GetComponent<BotCollection>().Bots.Count.ToString();

      GameObject.Find( "CountBoxGeneration" ).GetComponent<Text>().text =
GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.GenerationCount.ToString();

      GameObject.Find( "CountBoxLife" ).GetComponent<Text>().text =
GameObject.Find( "Controller"
).GetComponent<BotStartSpawn>().ThisSave.MaxLifeCount.ToString();
     }
}
```

ПРИЛОЖЕНИЕ AE EATORGANIC

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EatOrganic : MonoBehaviour {
     public void MoveEat() {
      Destroy( this.gameObject );
     }
}
```

ПРИЛОЖЕНИЕ AF SPECIFICATIONORGANIC

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpecificationsOrganic : MonoBehaviour {
```

```
    public float val = 50f;
}
```

ПРИЛОЖЕНИЕ AG RANDOMLOCATION

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RandomLocation : MonoBehaviour {

    // Use this for initialization
    void Start () {
     this.gameObject.transform.position = new Vector3( Random.Range( -451,
450 ), Random.Range( 449, 550 ), Random.Range( -451, 450 ) );
    }

    // Update is called once per frame
    void Update () {

    }
}
```