Санкт–Петербургский государственный университет

Менкеев Александр Саналович

Выпускная квалификационная работа

Применение методов нейроэволюции при разработке мобильной игры с процедурной генерацией контента

Уровень образования: бакалавриат
Направление: 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2020 «Программирование и информационные технологии»

Научный руководитель:
доцент, кафедра теории систем управления электрофизической аппаратурой, к.ф. - м.н.
Головкина Анна Геннадьевна

Рецензент:
доцент, кафедра технологии программирования, к.т.н.
Блеканов Иван Станиславович

Санкт-Петербург
2024 г.

Содержание

Введение

С тех пор как были изобретены компьютерные игры, количество человеко-месяцев, которые уходят на разработку успешной коммерческой игры, постоянно увеличивалось. Сейчас является нормой, что игра разрабатывается сотнями людей на протяжении года или даже дольше. Вследствие этого всё меньше игр являются прибыльными и всё меньше разработчиков способны выпустить собственную игру, что в свою очередь, приводит к меньшей готовности рисковать и меньшему разнообразию на рынке игр. Процедурная генерация контента способна частично устранить эту проблему, ведь многие из дорогостоящих сотрудников, необходимых в этом процессе являются дизайнерами и художниками, а не программистами. Компания по разработке игр, которая смогла бы заменить некоторых художников и дизайнеров алгоритмами, получила бы конкурентное преимущество, поскольку была бы способна производить игры быстрее и дешевле, сохраняя при этом качество.

Более того, методы процедурной генерации способны значительно повысить среднее игровое время. Ведь если игра реализует алгоритм, способный генерировать игровой контент со скоростью, с которой он потребляется, то в принципе нет причин, почему она должна завершаться. У игроков не будут заканчиваться уровни для прохождения, области для исследования, оружие для использования, персонажи для взаимодействия.

Ещё более захватывающей является идея игрового контента, который бы соответствовал вкусам игрока. Используя нейронные сети и методы их эволюции, можно создать систему процедурной генерации контента, которая бы подстраивалась под предпочтения игрока.

Очевидно, чтобы генерировать что-то, нужно знать что именно требуется получить. Какие свойства артефакта являются желаемыми? Какие будут нарушать игровой процесс и их следует исключить? Какой алгоритм использовать?

Данная работа посвящена методам процедурной генерации контента на примере генерации игрового оружия.

Постановка задачи

Целью данной работы является разработка программного инструмента, интегрированного в игровой движок Unity и предназначенного для процедурной генерации игрового оружия с уникальными траекториями снарядов.

Разрабатываемая система призвана облегчить разработку 2D игр в жанре шутеры и должна удовлетворять следующим требованиям:

1. Система должна иметь достаточное количество настраиваемых параметров оружия, чтобы пользователь мог генерировать именно то, что ему нужно.
2. Система не должна требовать больших усилий со стороны пользователя для генерации оружия.
3. Система должна иметь качественный пользовательский интерфейс, работающий внутри Unity.
4. Система должна иметь демонстрационные игровые сцены, показывающие возможные сценарии использования.

Целью также является создание прототипа мобильной 2D игры, которая бы использовала эту систему. Требования для игры:

1. Интерфейс, позволяющий игроку выбирать или отказываться от оружия, а также редактировать некоторые его параметры.
2. Два джойстика: один для движения, другой для стрельбы.
3. Наличие стреляющего врага, при этом вражеские снаряды и снаряды игрока должны иметь возможность сталкиваться.

Обзор литературы

Глава 1. Обзор существующих решений

Есть множество игр, в которых реализована процедурная генерация игрового оружия. Из них можно выделить две с наиболее интересными системами: Borderlands[14] и GAR[15].

Borderlands – это 3D шутер от первого лица с огромным разнообразием оружия. Система процедурной генерации оружия в этой игре основана на том, что каждое оружие представляется 35 деталями, среди них: ствол, глушитель, рукоять, цевье, конденсатор и другие. Каждая деталь меняет некоторые характеристики оружия: темп стрельбы, урон, наличие силового щита и т.п. При этом уникальных деталей в инвентаре игрока может быть до 1500. Также есть классификация оружия по его типу, редкости и производителю, что ещё больше увеличивает количество возможных вариантов.
Плюсы системы из этой игры:

- Крайне большое количество генерируемого оружия.
- Уникальный внешний вид оружия за счет комбинации различных деталей.

Минусы системы из этой игры:

- В системе не предусмотрено получение обратной связи от игрока, поэтому он не способен никак повлиять на генерацию оружия.
- Генерируемое оружие не сильно различается траекториями полета пуль.

Galactic Arms Race – это 2.5D космический шутер с видом сверху. Система процедурной генерации оружия в этой игре основана на том, что каждое оружие представляется нейронной сетью с помощью системы частиц. Затем эти нейронные сети подвергаются эволюции и тем самым создается новое оружие.
Плюсы системы из этой игры:

- Система подстраивается под предпочтения игроков.
- Уникальные траектории снарядов.

Минусы системы из этой игры:

– Для нормальной работы необходима группа игроков, так как в одиночном режиме оружие эволюционирует медленно.
– Игрок не имеет возможности редактировать параметры оружия.
– Генератор иногда выдает неприемлемое оружие, снаряды которого либо не движутся, либо колеблются так быстро, что коллизии с врагами не фиксируются игрой.

Вывод. В результате исследования рынка игр было выявлено, что большинство игр с процедурной генерацией оружия, используют системы, похожие на систему из Borderlands, то есть оружие собирается из деталей и тем самым происходит модификация его характеристик. В некоторых играх встречалась простая рандомизация параметров оружия, где диапазон случайных значений зависел от его типа или редкости. При этом GAR оказалась единственной игрой, где у оружия менялась сама траектория полёта снарядов.

Стоит отметить, что приведенные выше примеры являются коммерческими играми и поэтому получить исходный код их алгоритмов генерации оружия не получится. В Unity Asset Store не было найдено аналогов разрабатываемой в ходе этой работы системы. Данная система в конечном итоге будет выложена в открытый доступ, тем самым каждый разработчик Unity сможет интегрировать её в свой проект и использовать для разработки своей игры.

Глава 2. Оружие, основанное на системе частиц

Оружие, генерируемое системой, основано на идее из программы NEAT Particles[2, 3]. Каждое оружие содержит в себе нейронную сеть (Рисунок 1). Каждый кадр анимации каждая частица, выпущенная из оружия, подает на вход в нейронную сеть:

1. **RelativePos.x** – координата x текущего положения частицы относительно точки выстрела.
2. **RelativePos.y** – координата y текущего положения частицы относительно точки выстрела.
3. **DistanceFromOrigin** – расстояние от точки выстрела.

После этого нейронная сеть активируется и выводит для этого анимационного кадра:

1. **x** – координата x вектора скорости или силы, приложенной к частице.
2. **y** – координата y вектора скорости или силы, приложенной к частице.
3. **hue** – компонента цвета частицы в модели HSV.
4. **maxSpeed** – максимальная скорость частицы.
5. **force** – длина вектора скорости или силы, приложенной к частице.

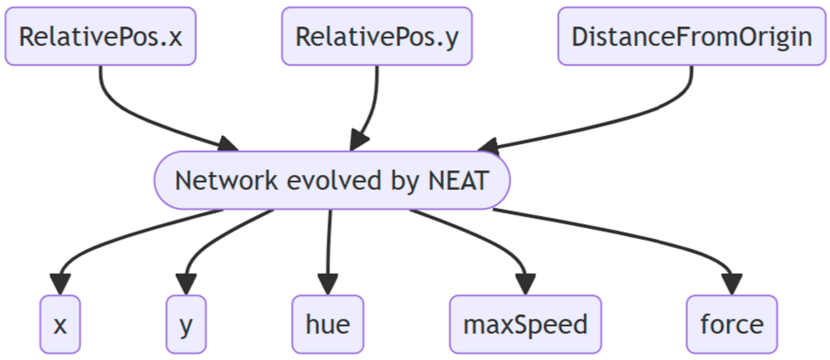

Рис. 1: Как нейронная сеть управляет снарядами.

Поскольку оружие представляется нейронной сетью, можно генерировать новое оружие, изменяя веса или топологию нейронной сети. Выбор алгоритма для этого обосновывается в следующей главе.

Глава 3. Методы эволюции нейронных сетей

Существует множество алгоритмов для эволюции нейронных сетей. Они классифицируются по схемам кодирования и тому, модифицируют ли они веса или топологию или и то, и другое. Однако не все они пользуются популярностью и имеют хорошие библиотеки с реализацией. Можно выделить два алгоритма NEAT[1] и HyperNEAT[1], которые находят своё применение до сих пор и имеют регулярно обновляемые библиотеки.

3.1 NEAT

Название этого алгоритма расшифровывается как нейроэволюция расширяющихся топологий (NeuroEvolution of Augmenting Topologies). Это метод, предназначенный для эволюции искусственных нейронных сетей с помощью генетического алгоритма. Главная идея NEAT заключается в том, что эволюцию наиболее эффективно начинать с маленьких, простых сетей, которые постепенно становятся всё более сложными с каждым поколением.

Алгоритм основан на трех ключевых принципах. Во-первых, для того, чтобы позволить структурам нейронных сетей усложняться с течением поколений, необходим метод отслеживания того, какой ген является каким. В противном случае в последующих поколениях будет неясно, какая особь с какой совместима и как их гены должны быть объединены для получения потомства. NEAT решает эту проблему, присваивая уникальную историческую метку каждому новому элементу структуры сети, который появляется в результате структурной мутации. Историческая метка – это число, присвоенное каждому гену в соответствии с порядком его появления в ходе эволюции. Эти числа наследуются без изменений во время скрещиваний и позволяют NEAT выполнять скрещивание без необходимости трудоемкого топологического анализа. Таким образом геномы различной организации и размеров остаются совместимыми на протяжении всей эволюции, что решает ранее открытую проблему сопоставления различных топологий в эволюционирующей популяции.

Во-вторых, NEAT разделяет популяцию на виды таким образом, что отдельные особи конкурируют в основном внутри своих собственных ниш, а не с

популяцией в целом. Благодаря этому топологические инновации защищены и имеют время оптимизировать свою структуру, прежде чем конкурировать с другими нишами в популяции. NEAT использует исторические метки на генах, чтобы определить, к какому виду принадлежат различные особи.

В-третьих, в отличие от других существующих систем эволюции нейронный сетей алгоритм NEAT начинается с однородной популяции простых сетей без скрытых узлов. Возникает новая топологическая структура по мере того, как происходят структурные мутации, и выживают только те структуры, которые признаны полезными в результате оценки функцией приспособленности. Таким образом, NEAT выполняет поиск топологических структур, начиная с самых простых, и находит оптимальную структуру для решения задачи.

Плюсы алгоритма:

– Наличие библиотеки на C#, которая регулярно обновляется.
– Подходит для трудно формализируемых задач, где функция потерь и функция приспособленности могут быть не определены.

Минусы алгоритма:

– Медленно сводится к оптимальному решению, если условия задачи очень сложны.
–

3.2 HyperNEAT

Плюсы алгоритма:

– Наличие библиотеки на C++, которая регулярно обновляется.
– Подходит для трудно формализируемых задач, где функция потерь и функция приспособленности могут быть не определены.

Минусы алгоритма:

– Намного медленнее, чем NEAT, поскольку требуется больше шагов для создания нейронных сетей на каждом из поколений.
– Изменяет только веса, топология нейронной сети определяется пользо-

вателем с помощью «субстрата».

Глава 4. Разработка системы генерации оружия

Исходный код доступен в репозитории[7].

4.1 Цвет и траектория снарядов

В этом параграфе описан упрощенный алгоритм управления снарядами нейронной сетью, который может отличаться от того, что представлено в исходном коде. Этот алгоритм выполняется каждый кадр игры.

Листинг 1: Projectile. Part 1

```
1  Box.ResetState();
2
3  Vector2 RelativePos = transform.localPosition;
4  float DistanceFromOrigin = RelativePos.magnitude;
5  float maxDistance = NNControlDistance * math.SQRT2;
6
7  _inputArr[0] = Mathf.Lerp(-1f, 1f, Mathf.Abs(RelativePos.x) /
       NNControlDistance);
8  _inputArr[1] = Mathf.Lerp(-1f, 1f, Mathf.Abs(RelativePos.y) /
       NNControlDistance);
9  _inputArr[2] = Mathf.Lerp(-1f, 1f, DistanceFromOrigin / maxDistance);
10
11 Box.Activate();
```

Рассмотрим подробнее этот фрагмент кода. `_inputArr[]` – это массив, через который нейронная сеть получает данные. Все значения, принимаемые нейронной сетью должны лежать в отрезке $[-1, 1]$. Для нормализации входных данных используется встроенный в Unity метод для линейной интерполяции `Lerp()`, а также параметр `NNControlDistance`, подбираемый пользователем.

Чтобы у пользователя была возможность менять направление полета снарядов относительно оси y на противоположное, их паттерн должен быть симметричен относительно этой оси. Также для более приятного визуального эффекта было решено сделать паттерн симметричным и относительно оси x. Для этого в сеть подаётся модуль первых двух параметров, отвечающих за координаты.

Если обобщить, то в коде выше берутся актуальные для данного кадра

позиция снаряда `RelativePos`, его расстояние от источника `DistanceFromOrigin` и подаются на вход в нейронную сеть после нормализации. Затем происходит активация сети `Box.Activate()`.

Листинг 2: Projectile. Part 2

```
13 float x = Mathf.Lerp(-1f, 1f, (float)_outputArr[0]) * SignX;
14 float y = Mathf.Lerp(-1f, 1f, (float)_outputArr[1]) * SignY;
15 Vector2 forceDir = OriginTransform.TransformDirection(x, y, 0f).normalized;
16
17 float hue, maxSpeed, force;
18 hue = Mathf.Lerp(HueRange.x, HueRange.y, (float)_outputArr[2]);
19 maxSpeed = Mathf.Lerp(SpeedRange.x, SpeedRange.y, (float)_outputArr[3]);
20 force = Mathf.Lerp(ForceRange.x, ForceRange.y, (float)_outputArr[4]);
21
22 SpriteRenderer.color = Color.HSVToRGB(hue, Saturation, Brightness);
23 Rigidbody.AddForce(forceDir * force);
24
25 float speed = Rigidbody.velocity.magnitude;
26 if (speed > maxSpeed)
27     Rigidbody.velocity = Rigidbody.velocity.normalized * maxSpeed;
28
29 transform.up = Rigidbody.velocity;
```

После активации сети можно считывать выходные данные через массив `_outputArr[]`. Сеть выдает все значения в отрезке $[0, 1]$. Данные с первых двух выходов `_outputArr[0]` и `_outputArr[1]`, отвечающих за x и y компоненты скорости, сначала преобразуются в отрезки $[-1, 1]$. Затем на их основе строится нормированный вектор силы `forceDir`, который потом прилагается к снаряду посредством метода `AddForce`.

Так было сделано для плавности и реалистичности полета снарядов, если бы нейронная сеть меняла их скорость напрямую, они бы двигались дёрганно и слишком резко. Чтобы под действием силы скорость снаряда не росла до бесконечности, она ограничена параметром `maxSpeed`.

Изначально планировалось использовать цветовую модель RGB, но возникли проблемы с контролем цветов, которые может выдавать сеть. Например, если у игры темный фон, то снаряды должны быть ярких оттенков, чтобы не сливаться с ним. У пользователя должна быть возможность задать желаемое пространство цветов, внутри которого будут заключены значения нейронной сети. RGB представляет цвета в виде точек в трехмерном пространстве, поэтому выделить такое пространство было бы затруднительно.

Эту проблему решает цветовая модель HSV (Hue, Saturation, Value — тон, насыщенность, значение), в которой цвет представляется более интуитивно понятным образом. Компонента `hue` задается в виде отрезка, минимальное и максимальное значение которого выбирается пользователем. `Saturation` и `Value` задаются как константы и тоже могут быть изменены пользователем.

4.2 Параметры оружия

Для точной настройки оружия было добавлено 29 параметров, которые контролируются пользователем.

Параметры стрельбы. Эти параметры регулируют скорострельность и начальную фазу полета снарядов:

1. **FireRate** – Время в секундах между каждой очередью или выстрелом.
2. **ProjectilesInOneShot** – Количество снарядов в очереди или выстреле.
3. **WeaponMode** – Определяет режим стрельбы. Их два:
    - **MultiShot** – Одиночный выстрел, состоящий из множества снарядов. Снаряды вылетают полукругом, который определяется параметром **Angle**.
    - **Burst** – Очередь, при которой снаряды вылетают по одному.
4. **BurstMode** – Определяет подвид стрельбы очередью. Их шесть:
    - **Clockwise** – Снаряды вылетают полукругом. Первый с максимальным углом, последний с минимальным. **SignX** у первой половины положительный, у второй – отрицательный.
    - **CounterClockwise** – Снаряды вылетают полукругом. Первый с минимальным углом, последний с максимальным. **SignX** у первой половины отрицательный, у второй – положительный.
    - **Alternate** – Начальные траектории снарядов чередуются от большего угла к меньшему. **SignX** чередует знак.
    - **Straight** – Снаряды летят прямо. **SignX** чередуется знак.
    - **MaxMinAngle** – Одна половина снарядов летит с максимальным уг-

лом, другая – с минимальным.

- **Random** – Угол и знак **SignX** выбираются случайно.

5. **BurstRate** – Время в секундах между снарядами при стрельбе очередью.

Параметры систем координат. Системы координат являются родителями снарядов в иерархии сцены, поэтому при их движении движутся и снаряды. Регулируя эти параметры, можно создавать более сложное оружие:

6. **RotationSpeed** – Вращение систем координат в градусах в секунду при нажатии кнопки запуска. Если значение отрицательное, вращение будет в противоположном направлении.
7. **MoveSpeed** – Скорость систем координат при нажатии кнопки запуска.

Параметры снарядов. Эти параметры отвечают за поведение и внешний вид снарядов:

8. **NetworkControlMode** – Определяет, как сеть управляет снарядом. Есть два режима:
   - **ForceSum** – Сеть выдает вектор силы, который затем суммируется с вектором силы, уже действующим на снаряд.
   - **VelocitySum** – Сеть выдает вектор скорости, который затем суммируется с вектором скорости, которым уже обладает снаряд.
9. **ReadMode** – Определяет, каким образом из выходных параметров (x, y) формируется вектор $\overrightarrow{dir}$, определяющий направление скорости или силы. Есть два режима:
   - **Default**:
   $$\overrightarrow{dir} = x\hat{x} + y\hat{y},$$
   где $\hat{x}, \hat{y}$ – орты локальной системы координат снаряда с началом в точке выстрела.
   - **Rotator**:
   $$\overrightarrow{dir} = x\hat{q} + y\hat{r},$$
   где $\hat{r}$ – орт радиус-вектора снаряда, и $\hat{q} \perp \hat{r}$.

10. **Size** – Размер снаряда. Для рендеринга снарядов используется текстура круга. Изменяя этот параметр, можно получать снаряды в форме эллипсов.
11. **Lifespan** – Срок жизни снаряда в секундах. По истечению этого срока снаряд уничтожается.
12. **HueRange** – Диапазон возможных значений для цветовой компоненты hue. Точное значение зависит от вывода нейронной сети.
13. **Saturation** – Цветовая компонента Saturation.
14. **Brightness** – Цветовая компонента Value.

Параметры паттерна. Эти параметры определяют настройки, связанные входными и выходными значениями нейронной сети:

15. **SpeedRange** – Диапазон возможных значений для максимальной скорости снаряда. Точное значение зависит от вывода нейронной сети.
16. **ForceRange** – Диапазон возможных значений силы, приложенной к снаряду. Точное значение зависит от вывода нейронной сети.
17. **NNControlDistance** – Определяет область, в которой снаряды управляются нейронной сетью.
18. **SignX** – Множитель вывода x. Если он равен нулю, вывод будет проигнорирован.
19. **SignY** – Множитель вывода y. Если он равен нулю, вывод будет проигнорирован.
20. **ForwardForce** – Прилагать ли к снаряду силу, направленную вперед. Эта сила равна выводу force нейронной сети.

Параметры начальной фазы:

21. **InitialFlightRadius** – Определяет область, в которой снаряды находятся в состоянии начальной фазы. В этом состоянии снаряды не управляются нейронной сетью, вместо этого они движутся с постоянной начальной скоростью.
22. **InitialSpeed** – Длина вектора начальной скорости.

23. **Angle** – Максимальный угол между начальной скоростью снаряда и вертикальной осью игрового объекта оружия.

Параметры отражения:

24. **FlipXOnReflect** – Изменять ли значение **SignX** на противоположное после отражения.
25. **FlipYOnReflect** – Изменять ли значение **SignY** на противоположное после отражения.
26. **Mode** – Определяет границы, от которых снаряд отражается. Есть три режима:
    - **CircleReflection** – Границей является окружность.
    - **RectangleReflection** – Границей является прямоугольник.
    - **Polar** – Границей является угол.
27. **ReflectiveCircleRadius** – Радиус границы-окружности.
28. **RectDimensions** – Размеры границы-прямоугольника.
29. **MaxPolarAngleDeg** – Максимально возможный полярный угол.

4.3 Пользовательский интерфейс

Для создания кастомных окон инспектора использовалась система IMGUI[10], которая предназначена для расширения функциональности Unity и разработки инструментов для разработчиков.

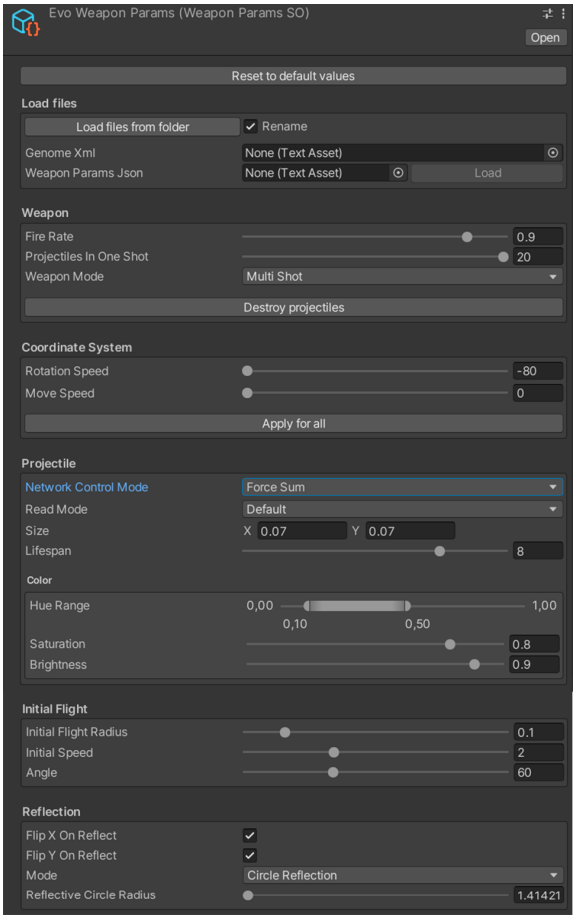

Рис. 2: Кастомный инспектор параметров оружия.

Особенности инспектора параметров оружия:

- Кнопка **Reset to default values**, возвращающая параметры к начальным значениям, заданным разработчиком.
- Раздел **Load files** для загрузки генома и параметров с файлов.
- Кнопка **Destroy Projectiles**, уничтожающая снаряды.
- Двойные слайдеры, позволяющие задавать два значения: минимальное и максимальное.
- Интерфейс подстраивается под выбранные значения. К примеру, если в

параметре **Weapon Mode** будет выбран режим **Burst**, появятся поля **Burst Mode** и **Burst Rate**.

- Есть поддержка редактирования нескольких объектов.
- Есть поддержка истории редактирования. То есть Undo/Redo.

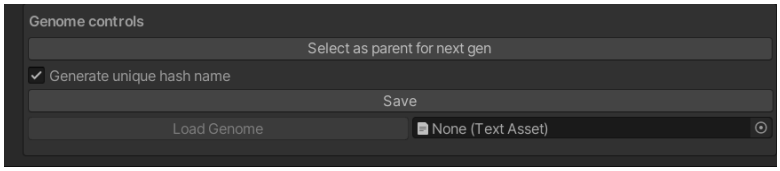

Рис. 3: Кастомный инспектор оружия.

Особенности инспектора оружия:

- Кнопка **Select as parent for next gen**, позволяющая пользователю выбирать понравившиеся геномы.
- Кнопка **Save** для сохранения генома и параметров оружия.
- Флажок **Generate unique hash name**. Если отмечен, то при сохранении названия файлов генерируется автоматически.
- Есть поддержка редактирования нескольких объектов.

4.4 Система эволюции оружия

Выбор библиотеки. Для эволюции нейронных сетей была выбрана библиотека SharpNEAT[8] версии 2.4.4, реализующая алгоритм NEAT на языке C#. Выбор этой библиотеки и её версии обусловлен в первую очередь совместимостью с игровым движком Unity.

Существует SharpNEAT версии 4.0.0, в которой значительно улучшена производительность и удобство программного интерфейса, но целевая платформа этой версии – .NET Core, несовместимая с Unity. По этой причине выбор остановился на более ранней версии 2.4.4.

Для использования библиотеки в Unity необходимо создать её DLL в какой-либо среде разработки и поместить эту DLL в специальную папку «Plugins». Плагины в Unity могут быть на трех языках: C, C++, C#. Плагины на C# доступны в бесплатной версии Unity, а вот для использования плагинов

на других двух языках придется покупать профессиональную версию.

Параметры алгоритма NEAT. Обычно NEAT используют для эволюции огромного количества нейронных сетей на протяжении множества поколений с помощью функции приспособленности, заданной математически. В нашем случае функцией приспособленности является человек – игрок или разработчик игры, поскольку задача оценки эстетичности и полезности паттернов не является строго формализируемой. По этой причине оцениваемая популяция и количество поколений не должны быть большими.

Приведенные ниже значения параметров были получены опытным путем, остальные параметры сохранили значение по умолчанию. Параметры подбирались таким образом, чтобы интересные паттерны в среднем выводились за 10-15 поколений.

Листинг 3: Params

```
1 PopulationSize = 6;
2 CloneOffspringCount = 2;
3 SexualOffspringCount = 4;
4 neatGenomeParams.InitialInterconnectionsProportion = 0.2;
5 neatGenomeParams.AddConnectionMutationProbability = 0.7;
6 neatGenomeParams.AddNodeMutationProbability = 0.7;
7 neatGenomeParams.ConnectionWeightMutationProbability = 0.8;
8 neatGenomeParams.DisjointExcessGenesRecombinedProbability = 0.5;
9 neatGenomeParams.NodeAuxStateMutationProbability = 0.2;
```

Краткое описание этих параметров:

- PopulationSize – Количество геномов, подвергаемых эволюции.
- CloneOffspringCount – Сколько геномов создается путем клонирования.
- SexualOffspringCount – Сколько геномов создается путем полового размножения.
- InitialInterconnectionsProportion – Плотность нейронных сетей первого поколения.
- AddConnectionMutationProbability – Вероятность появления нового ребра.
- AddNodeMutationProbability – Вероятность появления нового узла.

- ConnectionWeightMutationProbability – Вероятность изменения веса ребра.
- DisjointExcessGenesRecombinedProbability – Вероятность копирования всех не совпавших генов в один из потомков при половом размножении.
- NodeAuxStateMutationProbability – Вероятность узла приобрести дополнительное состояние (стать неактивным).

Создание нового поколения. Для создания нового поколения оружия необходимо выбрать родителей, на основе которых оно построится. Пользователь может выбрать любое их количество:

- Если пользователь не выбрал ни одного генома, программа случайно выберет два.
- Если пользователь выбрал один геном, то этот геном будет скрещен с не выбранными.
- Если пользователь выбрал два или более геномов, то выбранные геномы будут скрещены между собой.

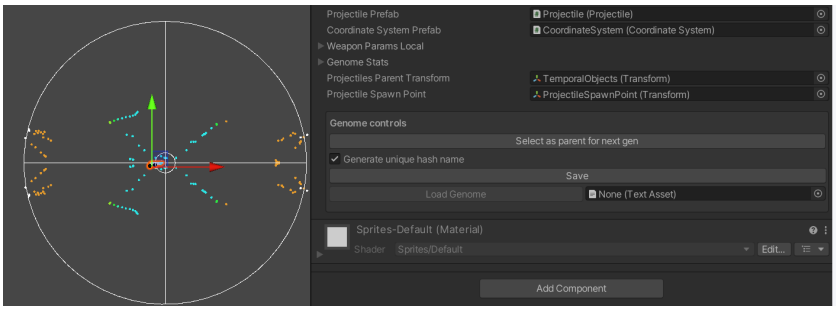

Рис. 4: Выбор родителей для нового поколения.

Чтобы выбрать геном в качестве родителя, нужно выбрать игровой объект оружия и в его инспекторе нажать кнопку **Select as parent for next gen**. После того как нужное количество родителей было выбрано, нужно нажать на кнопку **New Generation**.

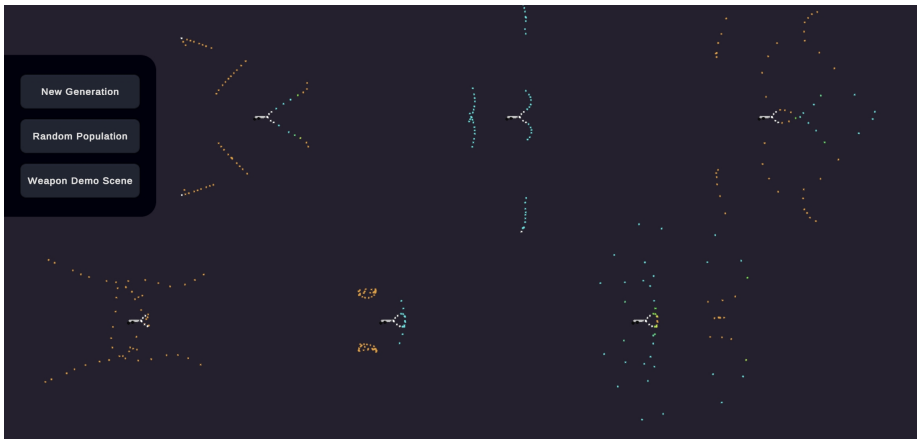

Рис. 5: Сцена эволюции оружия.

Если эволюция зашла в тупик, нужно нажать на кнопку **Random Population**, которая создаст случайное поколение.

4.5 Сохранение оружия

Чтобы сохранить понравившееся оружие, нужно выбрать игровой объект оружия и в его инспекторе (Рисунок 4) нажать кнопку **Save.** Будет создано и сохранено два файла:

- **Genome_<UniqueHash>.xml** содержит в себе информацию об узлах нейронной сети и функциях активации.
- **Params_<UniqueHash>.json** является представлением параметров оружия в формате json.

Глава 5. Разработка игры

WebGL сборку проекта можно посмотреть здесь[9].

5.1 Документация

Для публикации Unity требует документацию ассета на английском. Для написания документации использовалась система генерации статических сайтов Jekyll[11] и тема Just the Docs[12]. Документация доступна здесь[13].

5.2 Демонстрационные сцены

Для демонстрационных целей было создано три сцены:

- **Weapon Demo** – сцена, в которой демонстрируется 140 оружий, заранее созданных разработчиком. Кроме того, в этой сцене есть визуализация силовых полей, действующих на снаряды.

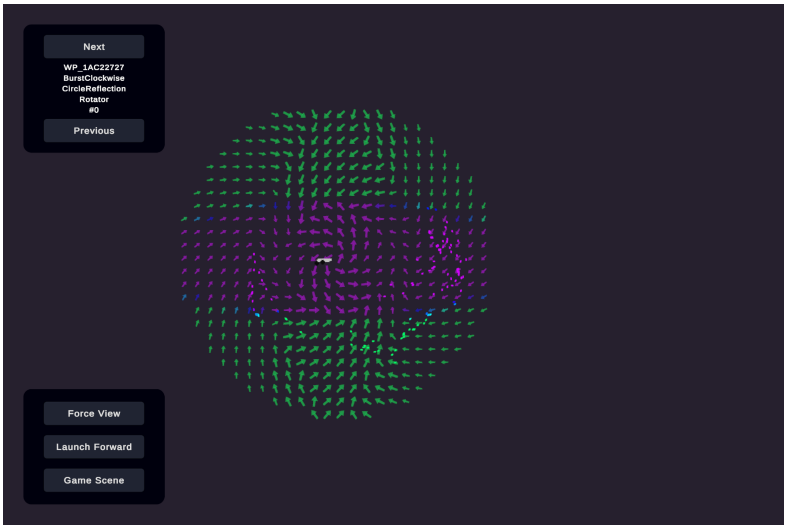

Рис. 6: Weapon Demo.

Замечание. Разные группы снарядов могут иметь разные значения параметров **SignY** и **SignX,** поэтому визуализация силового поля может быть некорректна для каких-то из них.

- **Shadow Survival** – простая игра в жанре «Shoot 'em up». Во время игры генерируется оружие, которое можно подобрать и настроить некоторые параметры.

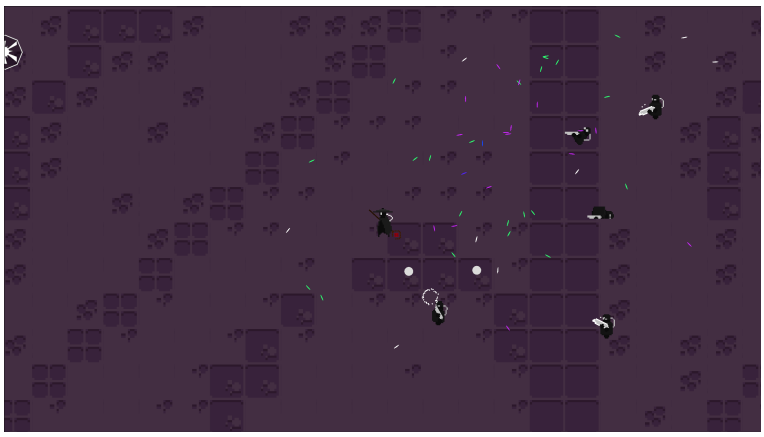

Рис. 7: Shadow Survival.

- **Space Shooter** – простая игра тоже в жанре «Shoot 'em up». Здесь оружие генерируется у врагов случайным образом.

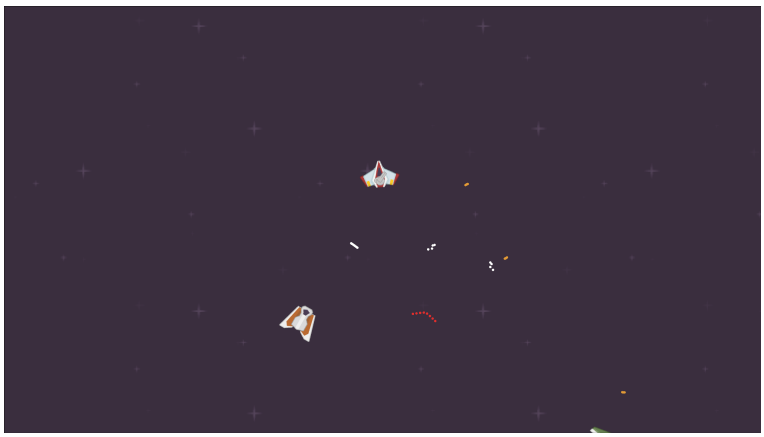

Рис. 8: Space Shooter.

Ассеты для создания двух игр были получены с таких сайтов, как:

1. Unity Asset Store[4]
2. opengameart.org[5]
3. itch.io[6]

5.3 Unity Asset Store

Для публикации было написано описание ассета, добавлены видео и скриншоты. На момент написания отчета ассет проходит модерацию.

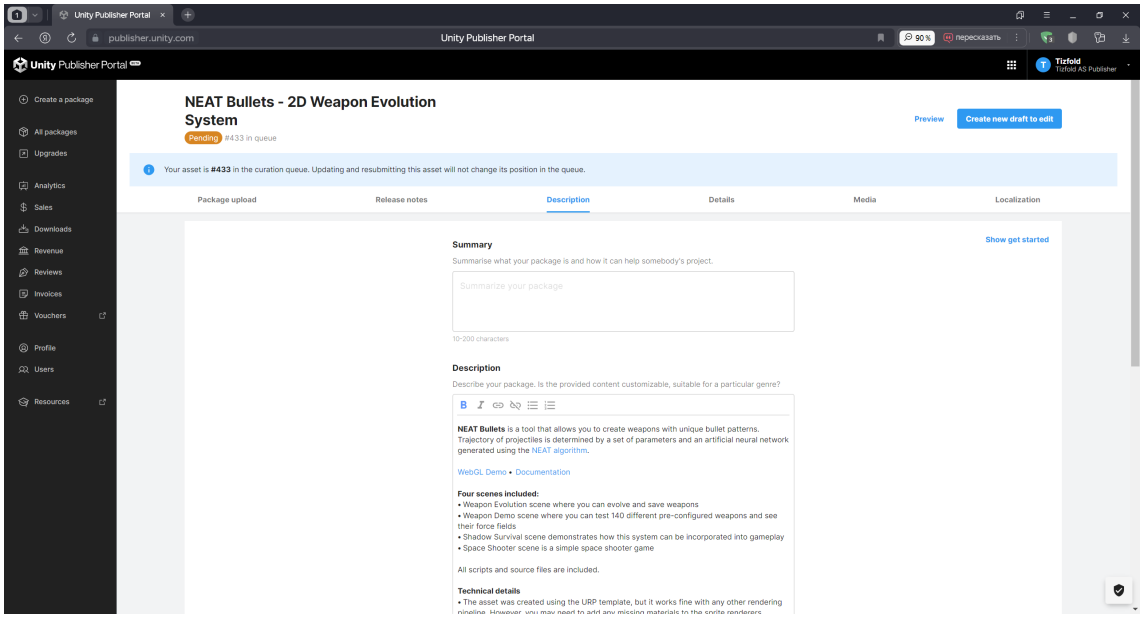

Рис. 9: Publisher Portal.

Заключение

В результате научно-исследовательской практики был разработан инструмент, полностью удовлетворяющий поставленным требованиям. Были созданы три демонстрационные сцены, написана документация и сделана WebGL сборка проекта. Кроме того, ассет был отправлен на модерацию в Unity Asset Store и вскоре будет опубликован.

В дальнейшем планируется доработать игру «Shadow Survival».

Список использованных источников

[1] Kenneth O. Stanley, Risto Miikkulainen. «Evolving Neural Networks through Augmenting Topologies». 2002. Available: https://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[2] Erin Hastings, Ratan Guha, and Kenneth O. Stanley. «NEAT Particles: Design, Representation, and Animation of Particle System Effects». 2007. Available: http://eplex.cs.ucf.edu/papers/hastings_cig07.pdf

[3] Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. «Interactive Evolution of Particle Systems for Computer Graphics and Animation». 2009. Available: http://eplex.cs.ucf.edu/papers/hastings_ieeetec09.pdf

[4] Unity Asset Store. Available: https://assetstore.unity.com/

[5] OpenGameArt. Available: https://opengameart.org/

[6] itch.io. Available: https://itch.io/game-assets

[7] Репозиторий. Available: https://gitfront.io/r/AlexanderMenkeev/pC4QrbyLrhVJ/WeaponAsset/

[8] SharpNEAT. Available: https://github.com/colgreen/sharpneat

[9] WebGL Demo. Available: https://tizfold.itch.io/neat-bullets

[10] IMGUI. Available: https://docs.unity3d.com/Manual/GUIScriptingGuide.html

[11] Jekyll. Available: https://jekyllrb.com/

[12] Jekyll theme Just the Docs. Available: https://github.com/just-the-docs/just-the-docs

[13] NEAT Bullets documentation. Available: https://alexandermenkeev.github.io/NEAT-Projectiles-docs/

[14] Borderlands wiki. Available: https://borderlands.fandom.com/wiki/Borderlands_2_Weapons

[15] Paper about GAR. Available: https://www.researchgate.net/publication/221157637_Evolving_content_in_the_Galactic_Arms_Race_video_game