Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему
САМООБУЧАЮЩИЙСЯ АЛГОРИТМ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Студент гр.053501 Н.В. Ермолаев

Руководитель: И.А. Удовин

2021

Белорусский государственный университет информатики и радиоэлектроники
Факультет компьютерных систем и сетей

«УТВЕРЖДАЮ»

Заведующий кафедрой информатики

_____Н.А. Волорова

«___»_____2021 года

ЗАДАНИЕ
по курсовой работе
студенту Ермолаеву Н.В.

1. Тема курсовой работы «Самообучающийся алгоритм на основе нейронных сетей».
2. Дата защиты курсовой работы «__» _____2021 г.
3. Исходные данные для курсовой работы
   3.1. Операционная система Windows.
   3.2. Язык программирования C++.
4. Содержание пояснительной записки
   4.1. Введение.
   4.2. Анализ предметной области.
   4.3. Разработка программного средства.
   4.4. Проверка работоспособности.
   4.5. Заключение.
   4.6. Список использованных источников.
5. Консультант по курсовой работе Удовин И.А.
6. Дата выдачи задания «___»_____2021 г.
7. Календарный график выполнения курсовой работы.
   7.1. Раздел 1, Введение к – 28.02.2021 г. – 10%
   7.2. Раздел 2 к – 15.03.2021 г. – 30%
   7.3. Раздел 3 к – 15.04.2021 г. – 60%
   7.4. Раздел 4 к – 10.05.2021 г. – 80%
   7.5. Заключение к – 20.05.2021 г. – 90%
   7.6. Оформление пояснительной записки и графического материала к – 24.05.2021 г. – 100%

Руководитель курсовой работе_____ И.А. Удовин

Задание принял для исполнения _____ Н.В. Ермолаев
(дата и подпись студентов)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В связи с бурным развитием сферы информационных технологий в 2000-х годах произошел молниеносный рост вычислительных технологий, что привело к революции в сфере машинного обучения. Выделилась совершенно новая ветвь развития машинного обучения, которая до недавнего времени без огромных вычислительных мощностей считалась бесполезной и бесперспективной — нейронные сети. С течением времени люди научились разрабатывать и обучать самые различные архитектуры нейронных сетей, которые решают абсолютно разные типы задач: от распознавания лиц до игры в го с результатами, которые на голову превосходят человеческие. Современные нейронные сети способны решать целый класс задач, в которых поиск наиболее оптимального решения нельзя четко алгоритмизировать. К примеру, нахождение оптимальной стратегии в компьютерных играх. Алгоритм, написанный человеком, может не учитывать всех факторов и совокупностей, влияющих на результат в игре, в отличие от гибких моделей нейронных сетей, которые методом проб и ошибок способны обучиться и находить пути решения задач, о которых люди могли даже и не подозревать.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Нейронные сети и матричные операции

Нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямиком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти. Другими словами, нейросеть — это машинная интерпретация мозга человека, в котором находятся миллионы нейронов передающих информацию в виде электрических импульсов.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной. Также есть нейрон смещения и контекстный нейрон. В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: input = output. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации и попадает в поле output.

Функция активации — это способ нормализации входных данных и внесения нелинейности в модель. Без функции активации теряется смысл моделей нейронных сетей с большим количеством скрытых слоев, ведь все

операции можно было бы заменить одним слоем с некоторым количеством нейронов.

Функция потерь (целевая функция) — функция, значение которой отражает успешность выполнения моделью поставленной задачи на некотором наборе данных. В общем, задачей всего машинного обучения является поиск подходящей функции и ее оптимизации (поиска глобального или близкого к глобальному минимума/максимума).

Нейронную сеть можно представить в виде математической модели, состоящей из линейных операций над матрицами. Слой входных нейронов будет представлен в виде вектора-строки параметров, которые передаются на вход в нейронную сеть для получения предсказания. Каждый скрытый слой будет представлять собой матрицу NxM, где каждый из M столбцов будет хранить в себе веса очередного нейрона в слое, состоящего из N нейронов. Выходной слой будет строиться по аналогии со скрытыми слоями, а нейрон смещений будет являться вектором-столбцов, устанавливающим, на какое значение необходимо сместить итоговый результат для достижения наиболее оптимального результата.

Последовательно перемножая матрицы, из которых состоит модель, и применяя функцию активации на каждом слое, мы получим итоговый результат предсказания модели, который будет в точности таким же, как если бы мы хранили нейросеть как наборы вершин и проводили бы операции на ней как обход графа. Поэтому данный метод является наиболее оптимальным для хранения нейронной сети в памяти компьютера.

1.2. Обучение с подкреплением и генетический алгоритм

Обучение с подкреплением, наряду с обучением с/без учителя, является одной из трех ключевых ветвей машинного обучения. В ходе обучения с подкреплением некоторая обучаемая система (также называемая

агентом) взаимодействует с некоторой средой, совершая определенные действия и реагируя на изменения системы, которые затем оцениваются на фоне успехов или неудач модели.

Генетический алгоритм — алгоритм поиска оптимального решения для задач оптимизации с использованием случайного подбора, комбинирования и вариации искомых параметров с использованием процессов, аналогичных естественному отбору в природе.

1.3. Описание генетического алгоритма

Задача ставится таким образом, чтобы ее решение было закодировано в виде «генотипа» идеального агента, который сможет идеально решать эту задачу. В нашем случае «генотипом» является модель нейронной сети, улучшаемой из поколения в поколение, пока не будет достигнут результат. Оценкой успеха очередного поколения должна служить некоторая функция приспособленности, благодаря которой мы сможем отличить провальные модели, которые должны вымереть в процессе эволюции, и перспективные модели, которые дадут потомство.

1.3.1. Создание начальной популяции

Первым шагом создается начальная популяция, состоящая из агентов с полностью случайными параметрами, которые совершенно не приспособлены для исполнения поставленной цели. Однако даже эти агенты будут отличаться между собой способностью к выполнению задачи, и мы выберем в процессе селекции лучших из них для создания перспективного потомства.

1.3.2. Селекция

Каждый агент оценивается при помощи функции приспособленности, и принимается решение, какие из них перейдут на следующее поколение. Это можно делать различными способами, но мы просто возьмем самых перспективных особей и создадим на их основе новое поколение.

1.3.3. Размножение и мутация

После выбора агентов, которые дадут потомство, остальные вымирают. Оставшиеся перспективные модели создают несколько своих копий, которые затем мутируют. При этом оригинальная модель не получает никаких изменений. Это нужно для того, чтобы в случае неудачной мутации всех потомков оставить нетронутой модель, которая показывала хорошие результаты, и попробовать изменить ее в другой раз, не теряя при этом прогресс. Мутация зависит от двух ключевых параметров: скорость и вероятность мутации. Вероятность мутации определяет, с каким шансом каждый вес в модели будет изменен, а от скорости зависит величина изменения. Эти две величины задаются в начале тренировки и определяются в основном эмпирически. Слишком низкие значения могут привести к слишком долгому обучению, в то время как выбор слишком высоких значений приведет к невозможности моделью найти оптимальные значения.

1.3.4. Переход на следующее поколение

Все старые успешные модели и их потомство переходят на следующее поколение, где вновь происходит процесс селекции, мутации и перехода. Алгоритм является циклическим и будет работать либо до заданного

максимального числа поколений, либо пока агенты не начнут показывать идеальный результат.

1.4. Игра «Snake»

«Snake» - компьютерная игра, возникшая в середине или в конце 1970-х. Игрок управляет длинным, тонким существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. В некоторых вариантах на поле присутствуют дополнительные препятствия. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Игрок управляет направлением движения головы змеи (обычно 4 направления: вверх, вниз, влево, вправо), а хвост змеи движется следом.

1.5. Выбор инструментария

Для реализации курсовой работы была выбрана интегрированная сред разработки Microsoft Visual Studio 2019 для разработки на языке C++. Также для создания графической оболочки была использована библиотека “Glut” для приложений под “OpenGL”, которая отвечает за создание окна и управление им.

Microsoft Visual Studio 2019 упрощает работу с C++, позволяя сосредоточиться на решении проблем. Также в данной среде разработки есть удобный Just-in-Time отладчик кода, позволяющий легко обнаружить ошибки в коде.

Библиотека GLUT позволяет создать GUI (graphical user interface) для наблюдения за обучением нейронной сети.

C++ — компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы

программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. Был выбран из-за его высокой производительности, требующейся для нейронной сети.

Git — распределённая система контроля версий.

GitHub — веб-сервис для хостинга IT-проектов и их совместной разработки.

1.6. Постановка задачи

В задачу курсовой работы входит разработка программного средства на языке C++, которое:

- Имеет возможность тренировки модели с гибкими начальными настройками
- Позволяет визуализировать весь процесс тренировки, в частности
    - Вывод текстовой информации об очередном поколении на экран пользователю
- Дает возможность протестировать работу обученной модели нейронной сети на самой игре
- Имеет возможность сохранять и загружать на диск и обратно натренированную модель

2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1. Интерфейс

Для создания легкого и приятного в использовании интерфейса приложения были выделены следующие критерии:

- Интерфейс должен быть максимально функциональным, чтобы тренировка модели могла начаться с различных первоначальных настроек программы
- Интерфейс должен быть удобен в использовании, чтобы пользователь мог легко начать обучение и тестирование модели, при этом не вызвав исключительную ситуацию в работе программы своим действием
- Интерфейс должен быть информативным, чтобы пользователь мог проследить за всем процессом тренировки и тестирования от начала и до конца

Пользовательский интерфейс данного программного средства представляет собой одно окно, которое было создано при помощи OpenGL.

Окно, которое видит пользователь не содержит в себе элементов управления, а лишь даёт возможность наблюдать за обучением нейросети и делать определённые выводы.

2.2. Реализация игры

Игра реализуется при помощи функции библиотеки GLUT для OpenGL, которые взаимодействуют непосредственно с классами, создающими игровой и тренировочный процесс.

1. Функции

- draw – указатель на эту функцию мы передаём в функцию библиотеки GLUT glutDisplayFunc(void (*func)(void)), которая будет запускаться с определённой частотой кадров, настроенной с помощью функции glutTimerFunc(unsigned int msecs, void (*func)(int value), value), с целью отрисовки игрового поля и установки некоторой логики игры.
- reshape – указатель на эфу функцию мы передаём в функцию библиотеки GLUT glutReshapeFunc(void (*func)(int width, int height)), которая отвечает за изменение размеров окна.
- timer – указатель на эту функцию передаётся в функцию библотеки GLUT glutTimerFunc(unsigned int millis, void (*func)(int value), int value), которая устанавливает таймер, частоту, с которой будет происходить смена картинки.

2. Классы
    - Snake – этот класс содержит в себе все данные, необходимые для описания одной змейки:
        - Report (метод) – метод, который необходим непосредственно для отслеживания процесса обучения: выводит в консоль информацию о лучшей змейке
        - Step (метод) – основной метод, который на основе математических операций выбирает оптимальное направление для движения змейки
        - m_gatherObservations_ (метод) – данный метод возвращает вектор, который содержит в себе 24 float: расстояние до стены, хвоста, фрукта в восьми направлениях(С, СВ, В, ЮВ, Ю, ЮЗ, З, СЗ)
        - m_inverseDistanceToWall_ (метод) – в данный метод мы передаём вектор направления, в котором хотим

узнать расстояние до стены от головы змейки, возвращаем 1.0f / i, где i – расстояние

- m_inverseDistanceToApple_ (метод) – в данный метод мы передаём вектор направления, в котором хотим узнать расстояние до фрукта. Если фрукт найдён, возвращаем расстояние до него, нет – возвращаем 0
- m_inverseDistanceToTail_ (метод) – в данный метод мы передаём вектор направления, в котором хотим узнать расстояние до хвоста. Если хвост найден, возвращаем расстояние до него, нет – возвращаем 0
- int* m_posX_ (поле) – массив, который хранит в себе каждую координату тела змейки по оси X
- int* m_posY_ (поле)– массив, который хранит в себе каждую координату тела змейки по оси Y
- m_snakeLength_ (поле) – длина тела змейки, максимальное значение этого поля – константа MAX, определённая в файле Constants.h
- m_gameOver_ (поле) – состояние змейки, жива она или мертва, в результате удара о стену, хвост или окончания время жизни
- m_fruits_ (поле) – количество съеденных фруктов
- m_direction_ (поле) – направление движения змейки, которое является членом класса перечисления sDirection, который определён в файле Directions.h

- Fruit
  - m_snake_ (поле) – указатель на змейку, непосредственно относящуюся к данному фрукту, в дальнейшем во время обучении змейки оказывается, что необходимо хранить

Snake* у каждого объекта класса Fruit, чтобы различать: какое яблоко к какой змейке относится

- m_reset_ (поле) – логическая переменная, отвечающая за состояние фрукта: необходимо его перерисовывать или нет
- m_x_ (поле) – координата фрукта по оси X
- m_y_ (поле) – координата фрукта по оси Y

- Board
  - Reset (метод) – данный метод отвечает за удаление старой змейки и, соответственно, фрукта, относящегося к ней, и создание новой на данном поле
  - m_snake_ (поле) - указатель на змейку, относящуюся к данному полю
  - m_fruit_ (поле) - указатель на фрукт, относящийся к данному полю и змейке m_snake_

2.3. Реализация нейронной сети и матриц

Класс Brain, CollectionsExtensions и Matrix содержат все необходимые методы для работы с нейронными сетями и матрицами в контексте приложения, а именно:

- Brain
  - Random (метод) – Возвращает новый мозг, у которого матрицы inputToHidden и hiddenToOutput заполнены абсолютно случайными весами, представляющими собой десятичное 32-битное число с плавающей точкой, лежащими в интервале [-1; 1]
  - Think (метод) – основной метод для принятия решения, в каком направлении необходимо двигаться змейке.

Исходный код метода Think:

```
std::vector<float> inputsWithOne =
CollectionsExtensions::AttachOne(inputs, m_inputSize_);
Matrix* inputMatrix =
CollectionsExtensions::VectorToColumnMatrix(inputsWithOne,
m_inputSize_ + 1);
Matrix* hiddenMatrix = m_inputToHidden_ * *(inputMatrix);
std::vector<float> hiddens =
CollectionsExtensions::ColumnMatrixToVector(*hiddenMatrix);
for (int i = 0; i < m_hiddenSize_; ++i) {
                ReLU(hiddens[i]);
}
if (inputMatrix) {
        delete inputMatrix;
        inputMatrix = nullptr;
}
if (hiddenMatrix) {
        delete hiddenMatrix;
        hiddenMatrix = nullptr;
}
std::vector<float> hiddensWithOne =
CollectionsExtensions::AttachOne(hiddens, m_hiddenSize_);
hiddenMatrix =
CollectionsExtensions::VectorToColumnMatrix(hiddensWithOne,
m_hiddenSize_ + 1);
Matrix* outputMatrix = m_hiddenToOutput * *(hiddenMatrix);
std::vector<float> outputs =
CollectionsExtensions::ColumnMatrixToVector(*outputMatrix);
if (outputMatrix) {
        delete outputMatrix;
        outputMatrix = nullptr;
}
if (hiddenMatrix) {
        delete hiddenMatrix;
        hiddenMatrix = nullptr;
}
return outputs;
```

- ReLU (метод) – метод, применяющий функцию активации ReLU к значению, переданному в этот метод

- Cross (метод) – данный метод принимает два объекта типа const Brain*, у которых имеется по две матрице у каждого: inputToHidden и hiddenToOutput — возвращается объект типа Brain*, у которого матрицы inputToHidden и hiddenToOutput – результат скрещивания исходных аналогичных матриц
- m_inputSize_ — количество строк в матрице, которая приходит на вход, без добавления единицы
- m_hiddenSize_ — количество строк в матрице inputToHidden
- m_outputSize_ — количество строк в матрице hiddenToOutput
- m_mutationChance_ — вероятность мутации веса некоторого нейрона
- m_inputToHidden_ — матрица, которая представляет собой веса input слоя
- m_hiddenToOutput_ — матрица, которая представляет собой веса output слоя

- CollectionExtensions – данный класс представляет собой лишь три статических метода, необходимых для преобразования векторов в матрицы и наоборот
  - AttachOne (метод) — данный метод принимает std::vector<float> vector и возвращает новый std::vector<float> vector, который отличается от принимаемого наличием единице в последней ячейке
  - VectorToColumnMatrix (метод) — данный метод принимает std::vector<float> vector и преобразует его в матрицу-столбец

Исходный код метода VectorToColumnMatrix:

```
float** columnArr = new float* [size];
if (!columnArr) {
        throw std::out_of_range("not enough memory");
        exit(1);
}
for (int i = 0; i < size; ++i) {
        columnArr[i] = new float[1];
        if (!columnArr[i]) {
                throw std::out_of_range("not enough memory");
                exit(1);
        }
}
for (int i = 0; i < size; ++i) {
        columnArr[i][0] = vector[i];
}
return new Matrix(columnArr, size, 1);
```

- ColumnMatrixToVector (метод) — данный метода принимает матрицу-столбец и преобразует её в std::vector<float>

- Matrix
  - Random (метод) – данный метод принимает количество строк и столбцов матрицы, которую необходимо заполнить абсолютно случайными весами и вернуть
  - operator* (метод) — выполняет умножение матриц между собой
  - Cross (метод) — метод, который поддерживает одну из основных концепций генетического алгоритма: скрещивание. С 50-ти процентной вероятностью вес в новую матрицу берётся из первой матрицы, с такой же вероятностью – из второй, а с 2-ух процентной вероятностью происходит мутация (вес берётся случайный) — тоже один из основных концептов генетического алгоритма

Вырезка из исходного кода метода Cross:

```
for (int i = 0; i < mom.m_rows_; ++i) {
        for (int j = 0; j < mom.m_columns_; ++j) {
                if (Rng::GetFloat(0.0f, 1.0f) < 0.5f)
                        cells[i][j] = mom.m_cells_[i][j];
                else
                        cells[i][j] = dad.m_cells_[i][j];
                if (Rng::GetFloat(0.0f, 1.0f) < mutationChance)
                        cells[i][j] = Rng::GetFloat(-1.0f, 1.0f);
        }
    }
```

2.4. Реализация генетического алгоритма и тренировки

Генетический алгоритм реализуется в классах Universe и Breed и содержит в себе необходимый для тренировки набор полей и методов:

- Universe
  - GetBestSnake (метод) — данный метод возвращает самую успешную из всех змеек данного поколения, успешность измеряется в заработанных очках(Score)
  - Step (метод) — при вызове данного метода все змейки данного поколения делают шаг(Step)
  - SpawnNextGeneration (метод) — основной метод, который отражает суть генетического алгоритма, здесь и происходит селекция, скрещивание и мутация

Исходный код метода SpawnNextGeneration:

```
std::vector<const Snake*> snakes;
for (size_t i = 0; i < m_boards_.size(); ++i) {
        snakes.push_back(m_boards_[i]->GetSnake());
}
Breed* breed = new Breed(snakes);
m_boards_.clear();
std::vector<Brain*> bestBrains = Breed::KeepTopN(0);
for (size_t i = 0; i < bestBrains.size(); ++i)
        m_boards_.push_back(new Board(new Brain(*bestBrains[i])));
```

```
for (size_t i = 0; i < m_worls_ - bestBrains.size(); ++i) {
        m_boards_.push_back(new Board(Breed::Spawn()));
}
++m_generation_;
```

- m_worlds_ (поле) — данное поле хранит количество полей(Boards)
- m_keepTopN_ (поле) — данное поле задаёт, какой количество самых успешных змеек из предыдущего поколения мы хотим сохранить
- m_boards_ (поле) — вектор, который хранит в себе все поля(Boards)
- m_generation_ (поле) — счётчик поколений

- Breed
  - Breed (конструктор) — в конструкторе мы собираем информацию о всех змейках, то есть std::pair<Brain*, double score>, в вектор m_scores_

Вырезка из исходного кода конструктора Breed():

```
for (int i = 0; i < snakes.size(); ++i) {
   score = snakes[i]->GetScore();
   m_scores_.push_back(std::make_pair(new Brain(snakes[i]->GetBrain()),
   score));
   m_totalScore_ += score;
}
std::sort(m_scores_.begin(), m_scores_.end(), [](auto& left, auto& right) {
   return left.second > right.second;
   });
```

- Spawn (метод) — выбираем два случайных мозга (Brain), скрещиваем их

- KeepTopN (метод) — в зависимости от переданного параметра count, оставляем ровно столько самых успешных мозгов
- m_chooseParent (метод) — с вероятностью, прямо пропорциональной количеству набранных очков (Score) выбираем мозг (Brain) для скрещивания
- m_scores_ (поле) — данное поле хранит в себе вектор пар, где пара – это std::pair<Brain*, double score>
- m_totalScore_ (поле) – общий счёт(Score), набранный всеми змейками

3. ПРОВЕРКА РАБОТОСПОСОБНОСТИ

Посмотрим на прогресс процесса обучения нейронной сети, в зависимости от поколения

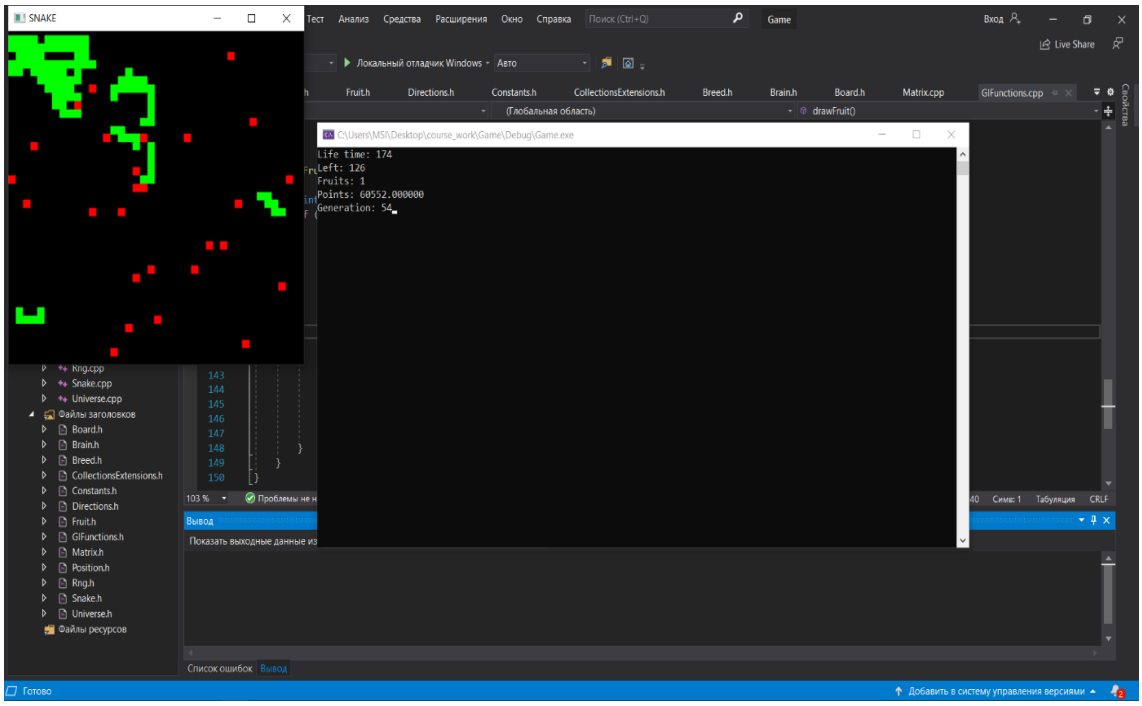

Рисунок 1 – 50-е поколение

На Рисунке 1 мы наблюдаем, что в результате эволюции змейки научились примитивно избегать столкновения со стенкой, а также изучили верхнюю часть поля, где могут время от времени случайно собрать один-два фрукта.

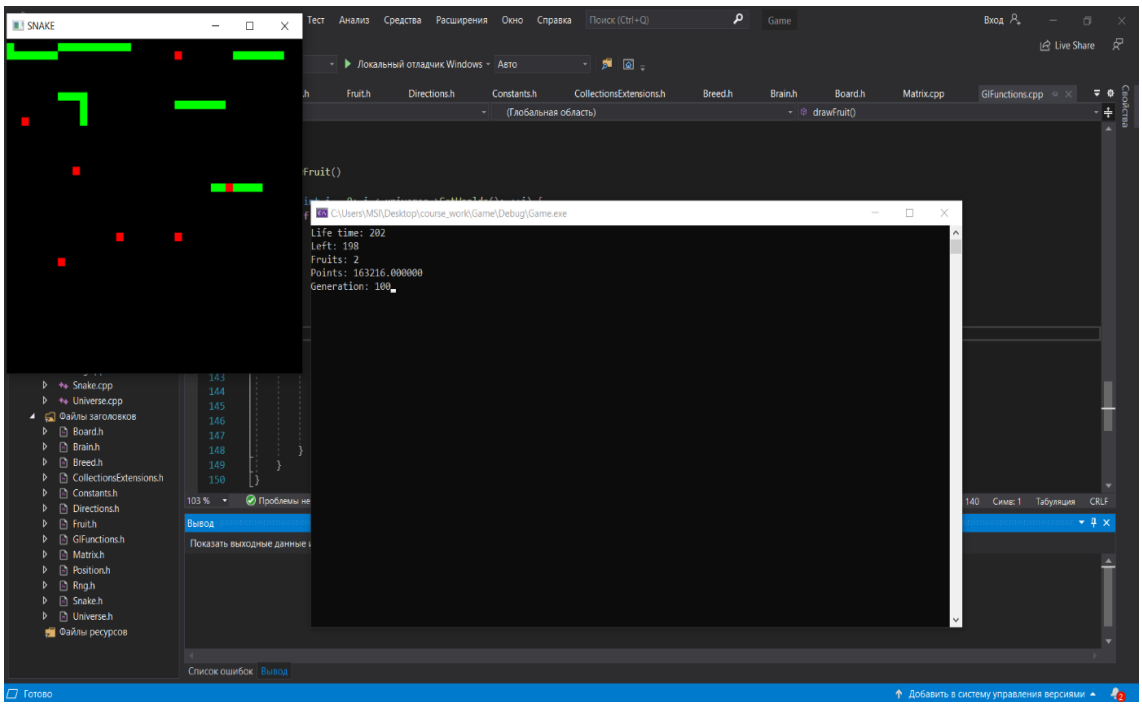

Рисунок 2 – 100-е поколение

На Рисунке 2 мы наблюдаем, чему научились змейки к 100-му поколению. Они по-прежнему стараются держаться ближе к верхней части поля, но уже активно по диагонали изучают общее состояние поля. На яблоки иногда правильно начинают реагировать рецепторы.

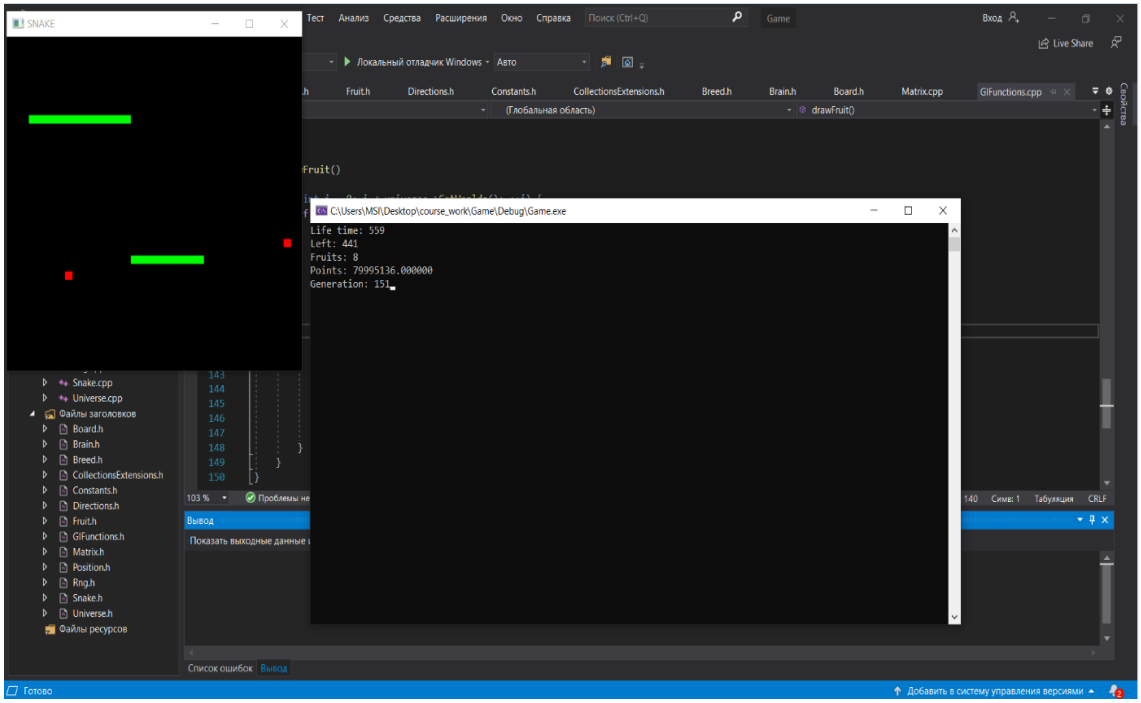

Рисунок 3 – 150-е поколение

На Рисунке 3 мы наблюдаем уже 150-е поколение змеек. Змейки окончательно освоились на всем поле, а не только в верхней его части. В результате эволюции была выбрана стратегия движения по кругу, с реагированием на попадающиеся фрукты.

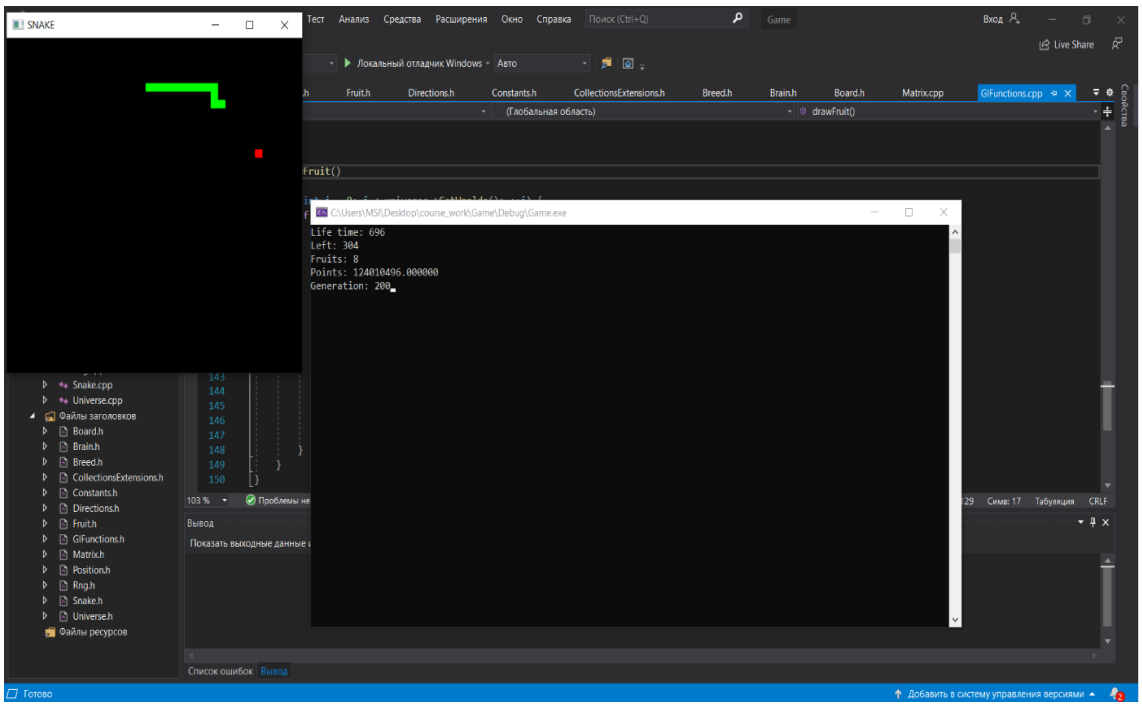

Рисунок 4 – 200-е поколение

На Рисунке 4 мы наблюдаем 200-е поколение змеек. Тактика по-прежнему остаётся старая и наиболее удачная – движение по кругу, но за пятьдесят поколений змейки научились безошибочно бросаться на яблоко, когда оно попадает в зону видимости их рецепторов.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано приложение, корректно справляющееся с процессом обучения нейронной сети методом обучения с подкреплением с использованием генетического алгоритма, а также аналог игры Snake, на котором было произведено тестирование программы. Генетический алгоритм показал достойные результаты обучения, а благодаря продуманной архитектуре приложения существует возможность дальнейшего расширения и добавления новых игр к уже существующим без существенных модификаций исходного кода тренировки.

СПИСОК ИСТОЧНИКОВ

1. Wikipedia: https://ru.wikipedia.org/wiki/
2. MSDN: https://docs.microsoft.com/ru-ru/
3. Habr: https://habr.com/ru/
4. Николенко, Кадурин, Архангельская: Глубокое обучение. Погружение в мир нейронных сетей: https://library.bsuir.by/m/12_101945_1_128064.pdf