Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет**

**имени первого Президента России Б.Н. Ельцина»**

**Институт математики и компьютерных наук.**

**Кафедра алгебры и дискретной математики**.

**Эволюционирование стратегии движения танка на поле боя методом генетического программирования**

Курсовая работа

студента 4 курса

Антропова Д.А.

Научный руководитель:

к.ф.-м.н.

Горбенко А.А.

Екатеринбург

2016 год

Оглавление

[Введение 3](#_Toc453444844)

[Постановка задачи 4](#_Toc453444845)

[Формат стратегий 5](#_Toc453444846)

[Генетический алгоритм 7](#_Toc453444847)

[Создание начальной популяции 7](#_Toc453444848)

[Скрещивание 7](#_Toc453444849)

[Мутации 9](#_Toc453444850)

[Селекция 9](#_Toc453444851)

[Функция приспособленности 10](#_Toc453444852)

[Полученные результаты 10](#_Toc453444853)

[Визуализация данных 11](#_Toc453444854)

[Заключение 14](#_Toc453444855)

[Список литературы 15](#_Toc453444856)

[Приложение 1 17](#_Toc453444857)

[Приложение 2 18](#_Toc453444858)

[Приложение 3 19](#_Toc453444859)

# Введение

Генетическое программирование – процесс создания или улучшения программ с помощью генетических алгоритмов [8]. Генетическое программирование, наряду с генетическими алгоритмами и нейронными сетями, относятся к эволюционным или биологическим методам создания искусственного интеллекта. Задача создания искусственного интеллекта вызывает огромный интерес научного сообщества, эта отрасль активно исследуется и изучается. Системы, использующие искусственный интеллект, применяются в робототехнике [7], в медицине для диагностирования заболеваний (см., например, MYCIN [13]), в финансовой отрасли, в транспортной отрасли – создание беспилотных автомобилей (см., например, Google [14]). Производятся попытки обучения машин написанию книг и созданию музыки (см., например, Melomiks [15]) – области, которые раньше считались только прерогативой человека.

Создание программ является трудоемким процессом, требующем огромных знаний и опыта в различных областях у исполнителей – программистов, а также значительных затрат по времени. И чтобы избавить людей от рутинной работы применяются различные способы автоматизировать данный процесс. Научить компьютер самому решать задачи. Особенно это относится к задачам, не имеющим известного, детерминированного решения, задачам, имеющим большую размерность, решение которых известными методами занимает большое количество времени, а также задачам, единственным известным способом решения которых является полный перебор. Для решения такого класса задач и применяются различные эвристические подходы.

Для обучения компьютера автоматическому решению задач с помощью генетического программирования используются методы, позаимствованные у природы [9]. В соответствие с теорией эволюции Дарвина, природа, используя генетические методы, способна решать совершенно немыслимые задачи и создавать виды, отлично приспособленные к самым невероятным окружающим условиям.

Особенно интенсивно в последнее время различные эволюционные методы применяются в робототехнике. Так, можно отметить создание интеллектуальных автономных управляющих программ для систем робота [6]. Исследовалась проблема движения роботов в конкурентной среде [7].

Рассматриваемая в рамках данной работы задача нахождения стратегии движения танка по полю боя является актуальной и представляет существенный интерес в научном сообществе. Например, можно отметить международные соревнования по виртуальным танковым боям для программистов Robocode [12].

Следует отметить, что задача нахождения оптимальной стратегии движения танка также находит свое применение и в условиях реальных танковых боев. В частности, в России ежегодно проводятся соревнования по танковому биатлону [3]. Изучение и нахождение оптимальной стратегии автономного движения танка представляет существенный интерес.

# Постановка задачи

В данной работе рассматривается задача нахождения стратегии движения танка на игровом поле боя от начальной до конечной точки. Кроме исходного танка на виртуальной карте поля боя также перемещаются другие танки. Все танки имеют возможность производить выстрелы по противникам. Целью работы является нахождение методом генетического программирования такой стратегии движения танка, которая обеспечила бы ему проезд от начальной до конечной точки, при этом танк не должен быть подстрелен. Если же достичь конечной точки не получается, то прожить наибольшее число ходов и подбить наибольшее число противников.

Рассмотрим отдельные элементы задачи.

Поле боя представляет собой двумерную плоскость, которая генерируется случайным образом на основе задаваемых параметров: длины, ширины, количества препятствий, количества врагов. Параметры поля считываются из файла конфигурации[[1]](#footnote-1). Также на виртуальном поле указана начальная точка – место, где начинает свое движение танк и конечная точка – место, которое танк должен достичь.

Танк[[2]](#footnote-2) в единицу времени занимает одну клетку на поле. Каждый танк имеет направление (вверх, вниз, влево, вправо). За один ход танк может:

* Сдвинуться вперед или назад.
* Повернуться на 90 градусов по или против часовой стрелки.
* Произвести выстрел.

Танк обладает областью видимости (танк знает, что находится в квадрате с центром в текущей позиции танка и длиной стороны 2K+1, где K – область видимости, противник) и областью поражения (танк может вести огонь по целям, удаленным от него по прямой и не закрытым каким-либо препятствием на расстояние не больше R, где R – радиус поражения). Параметры K и R задаются в файле конфигурации[[3]](#footnote-3).

Движение танков происходит в пошаговом режиме.

# Формат стратегий

Определим формат программируемых стратегий следующим образом: множество терминалов состоит из команд:

|  |  |
| --- | --- |
| *TurnRight,* | Повернуться вправо |
| *TurnLeft* | Повернуться влево |
| *Forward* | Сдвинуться вперед |
| *Backward* | Сдвинуться назад |
| *Shoot* | Произвести выстрел |

Для множества функций было решено использовать команды с интуитивно понятным синтаксисом вида «if\_условие {\_\_\_}». В фигурных скобках содержится подпрограмма, которая выполняется при выполнении условия. Данная подпрограмма может содержать как терминалы, так и функции, для создания вложенных условий. Множество функций определим следующим образом:

|  |  |
| --- | --- |
| *If\_Enemy\_In\_Visible\_Area* | Проверяет есть ли враг в видимой области |
| *If\_Enemy\_In\_Fire\_Area* | Проверяет есть ли враг в простреливаемой области |
| *If\_Enemy\_Front* | Проверяет есть ли враг спереди |
| *If\_Enemy\_Right* | Проверяет есть ли враг справа |
| *If\_Enemy\_Left* | Проверяет есть ли враг слева |
| *If\_Enemy\_Back* | Проверяет есть ли враг сзади |
| *If\_Obstacle\_Front* | Проверяет есть ли препятствие спереди |
| *If\_Obstacle\_Right* | Проверяет есть ли препятствие справа |
| *If\_Obstalce\_Left* | Проверяет есть ли препятствие слева |
| *If\_Obstacle\_Back* | Проверяет есть ли препятствие сзади |

Пример программы, генерируемой с помощью генетического программирования, заданной по нашим правилам:

*If\_Enemy\_In\_Visible\_Area{ TurnLeft Forward Shoot } Backward Shoot Shoot Backward Shoot If\_Enemy\_In\_Visible\_Area{ TurnLeft } If\_Enemy\_In\_Visible\_Area{ } TurnLeft Shoot Forward Shoot Backward Backward Forward If\_Enemy\_In\_Visible\_Area{ TurnLeft } If\_Enemy\_In\_Visible\_Area{ } Shoot If\_Enemy\_Front{ TurnRight } If\_Enemy\_In\_Visible\_Area{ } Forward Shoot If\_Enemy\_In\_Visible\_Area{ TurnRight Shoot } Shoot Backward Backward If\_Enemy\_In\_Visible\_Area{ TurnLeft } Forward TurnLeft Shoot Forward Shoot Backward Backward Forward If\_Enemy\_In\_Visible\_Area{ TurnLeft } If\_Enemy\_In\_Visible\_Area{ } Forward TurnLeft Shoot Forward Shoot Backward Backward Forward If\_Enemy\_In\_Visible\_Area{ TurnLeft } If\_Enemy\_In\_Visible\_Area{ } Forward Forward If\_Enemy\_In\_Visible\_Area{ Forward Shoot } Shoot Shoot Shoot If\_Enemy\_In\_Visible\_Area{ } If\_Ememy\_Back{ Forward Shoot } Shoot Backward Shoot If\_Enemy\_Front{ TurnRight } Backward Forward If\_Enemy\_In\_Visible\_Area{ Shoot }*

# Генетический алгоритм

Работу генетического алгоритма упрощенно можно представить в следующем виде [1]:

1. Создание начальной популяции и их оценка целевой функцией.
2. Создание новых особей, путем
   1. скрещивания
   2. мутации
3. Селекция
4. Если достигнуто значение целевой функции в какой-либо особи, то выход, иначе перейти к шагу 2.

Рассмотрим по порядку все шаги.

## Создание начальной популяции

Для создания начальной популяции стратегий был использован вероятностный метод на основе двух параметров: CloseBracketCoefficient – вероятность уменьшения уровня вложенности функции и NewFunctionCoefficient – вероятность создания более «глубоких» функций. Данные параметры, а также размерность популяции и максимальный размер стратегии задаются в файле конфигурации[[4]](#footnote-4). С фрагментом кода программы можно ознакомиться в приложении 2.

## Скрещивание

В данной работе использовано 3 метода:

1. Свободное скрещивание (Panmixia).
2. Внутривидовое скрещивание(Inbreed).
3. Межвидовое скрещивание(Outbreed).

В файле конфигурации параметр CrossoverProb определяет количество новых особей, которые появились в результате скрещивания. Параметры PanmixiaRatio, InbreedRatio, OutbreedRatio определяют в каком соотношении будут добавляться новые особи каждого вида скрещивания. В свободном скрещивании – случайно выбираются две особи из популяции. Во внутривидовом скрещивании случайно выбирается одна особь и из оставшихся особей выбирается наиболее похожая особь. В межвидовом выбирается наиболее непохожая особь.

Сходство особей определяется с помощью аналога расстояния Хэмминга для стратегий. Для его вычисления необходимо сравнить две стратегии и посчитать количество совпавших команд. Получившуюся величину и назовем расстоянием Хэмминга для стратегий.

Рассмотрим, как происходит скрещивание двух стратегий. Сначала необходимо выбрать две стратегии одним из методов, описанных выше. Выберем случайным образом для первой стратегии подстратегию, которая представляет собой первую часть исходной стратегии случайной длины. А для второй стратегии подстратегию, представляющую вторую часть исходной стратегии случайной длины. Если суммарная длина выбранных стратегий превышает максимально допустимую длину стратегии, выбрасываем из стратегии последние и первые команды для первой и второй стратегии, соответственно. Осуществляем эту операцию до тех пор, пока исходная суммарная длина подстратегий не станет равна максимально допустимой длине стратегии.

Далее выбранные подстратегии необходимо привести к состоянию допустимой стратегии. Под допустимой стратегией мы понимаем такую стратегию, у которой не нарушено правило вложенности функций. Для первой стратегии: если она заканчивается на середине некоей функции «If\_Condition {», то дополним эту стратегию символами «}» до тех пор, пока глубина входа в функции не станет равна нулю. Для второй стратегии, которая может начинаться внутри нескольких функций, будем сдвигать ее на величину минимальной глубины входа. Таким образом, из двух недопустимых стратегий получаем две допустимые стратегии, которые можно соединить, и они составят также допустимую стратегию.

## Мутации

Мутации будем производить следующим образом: в файле конфигурации параметр MutationProb определяет сколько новых особей появится в результате мутаций. Далее случайным образом выбирается особь из популяции, к которой будут применяться мутации, и количество мутаций из диапазона от 1 до размера выбранной стратегии. Далее производим мутацию по правилу: выбираем случайно команду из стратегии. Если это терминал, то меняем его на случайный терминал. Если это функция, то меняем его на случайную функцию.

## Селекция

Селекция в ходе работы производится двумя методами:

1. Максимальная селекция. Все особи упорядочиваются по значению функции приспособленности. Выбираются особи с максимальным значением функции приспособленности.
2. Турнирная селекция. Выбираются 2 особи. Особь с наименьшей функцией приспособленности выбывает из популяции. Повторяется пока размер популяции не станет равен максимально допустимому.

В результате проведенных вычислительных экспериментов было установлено, что использование селекции второго типа более приемлемо в силу меньших затрат по ресурсам.

## Функция приспособленности

Для вычисления функции приспособленности было смоделировано поведение танка и всех противников на поле боя. После смоделированного боя функция приспособленности вычисляется следующим образом:

public void CountFitness (Battle battle)

{

Result = Steps + Killed \* 50 + countMetric(battle) \* 10;

if (FinishAchieved)

Result += 10000;

if (! IsAlive)

Result -= 100;

}

Здесь Steps – количество шагов, пройденных стратегией. Killed – количество танков убитых стратегией. Функция countMetric находит на какое число клеток сдвинулся от начальной точки танк в сторону конечной точки. FinishAchieved – указывает достигла ли стратегия финиша. IsAlive показывает остался ли танк в живых после выполнения стратегии.

## Полученные результаты

В процессе решения поставленной задачи был разработан программный комплекс, написанный на языке C# 5.0 на платформе .Net 4.5.2. Наблюдение за процессом работы генетического алгоритма происходит с помощью вывода информации о текущей популяции в лог-файл. Для этого производится сбор статистических данных по каждой популяции: максимальное, минимальное и среднее значение функции приспособленности. Ознакомиться с примером выводимых данных и их изменениями от поколения к поколению можно в Приложении 3.

Для оценки эффективности предложенного алгоритма был выполнен вычислительный эксперимент, в процессе которого были получены следующие результаты. Для поля 15\*10 клеток с размером популяции в 100 особей, размером максимальной стратегии 1000 элементов, с 20 препятствиями на карте и 10 танками противника произведено 100 запусков программы. С помощью описанного метода генетического программирования, решение было найдено в 99% случаях. В среднем по времени поиск решения занимает 19580 миллисекунд. В среднем решению требуется 12 поколений для нахождения правильной стратегии[[5]](#footnote-5).

На других начальных данных, поля 30\*20 клеток с размером популяции в 100 особей, размером максимальной стратегии 1000 элементов, с 100 препятствиями на карте и 30 танками противника также было произведено 100 запусков программы. Кроме того, было добавлено дополнительное ограничение: если решение не находится в течение 5 минут, то считаем, что алгоритм не справился с задачей. С помощью описанного метода генетического программирования, решение находится в 77% случаях. В среднем по времени занимает 144697 миллисекунд или 2.5 минуты. В среднем решению требуется 87 поколений для нахождения правильной стратегии.

Следует отметить, что для получения наилучшей производительности были использованы параллельные вычисления с помощью библиотеки для языка C# TPL (Task Parallel Library) [16]. Подсчет функции приспособленности – затратная по времени операция. Пересчет функции приспособленности для одного поколения представляет собой нахождение значения функции приспособленности для множества стратегий. Эти операции не взаимодействуют друг с другом и могут выполняться параллельно, что и было сделано. Для популяции размером в 100 особей на карте размером 30\*20 получили ускорение в среднем с 2100 миллисекунд до 1400 миллисекунд.

# ­Визуализация данных

Для визуализации данных и наглядного просмотра стратегий, получаемых методом генетического программирования, была написана программа на языке C# 5.0 и XNA Game Framework 4.4 [11]. В программу можно загрузить файл карты, файл со стратегией противника и стратегией танка. Данные файлы отображаются в программе генетического программирования вместе с логами, их можно загрузить в визуализатор и наглядно посмотреть, как изменяется стратегия танка из поколения в поколение. В частности, на Рис. 1 и Рис. 2 изображено окно программы визуализации для различных стратегий движения танка.



Рис. 1. Пример работы программы визуализации. Желтый танк – танк со стратегией, полученной методом генетического программирования. Фиолетовые танки – танки противника. Белый квадрат – начальная позиция танка. Фигурка орла – клетка, достижение которой – главная задача танка. При данной стратегии танк смог просуществовать 28 ходов и подстрелить 3 других танка. Текущее значение функции приспособленности 238.



Рис. 2. Пример работы программы визуализации. При данной стратегии танк смог просуществовать 72 хода, никого не подстрелил, и сам был подбит. Текущее значение функции приспособленности -8.

# Заключение

В процессе работы было получено решение для поставленной задачи – разработан алгоритм генетического программирования для нахождения стратегии движения танка по игровому полю боя. Создан «язык» для стратегий танка, который оказался вполне работоспособен, и который смог справиться с поставленной задачей. Предложен вероятностный метод генерации начальной популяции стратегий, разработаны и применены различные методы скрещивания, селекции. Был разработан программный комплекс, включающий модуль визуализации, модуль вычислительных функций, модуль построения карт, модуль генерации стратегий, модуль, реализующий генетические функции. В программном комплексе предусмотрено ведение подробного вывода внутренней информации и параметров алгоритма, по которым можно проанализировать характер генетических изменений в популяциях. Для тестирования разработанного алгоритма был произведен вычислительный эксперимент, в результате которого было установлено, что оптимальное решение для поля 15\*10 в среднем находится за 20 секунд на 12 поколении стратегий. Оптимальное решение для поля 30\*20 в среднем находится за 144 секунды на 87 поколении стратегий.

Исходные коды программного комплекса можно найти по электронному адресу: https://github.com/dantre/CourseWork

# Список литературы

1. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. — 2-е изд. — М: Физматлит, 2006. — С. 320
2. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. И.Д. Рудинского.–М.: Горячая линия – Телеком, 2006. –452с.: ил.
3. Танковый биатлон [Электронный ресурс]. URL: http://tankbiatlon.ru/ (дата обращения 06.06.2016)
4. Шилдт Г. C# 4.0: полное руководство: Пер. с англ. — М.: ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ.
5. Шилдт Г. C# Учебный курс. — М: Питер, 2003, 471 стр.
6. Evolutionary robotics [Электронный ресурс].

URL: http://www.evolutionaryrobotics.org/ (дата обращения 06.06.2016)

1. Kober J., Peters J. Learning motor skills: from algorithms to robot experiments.–Cham: Springer, 2014. –201p.
2. Koza J. Genetic Programming on the programming of computer by means of natural selection. –MIT Press, 1998. –609p.
3. Mitchell M. An introduction to genetic algorithms. –MIT Press, 1996. –162p.
4. RB Whitaker XNA Tutorials [Электронный ресурс]. URL: http://rbwhitaker.wikidot.com/xna-tutorials (дата обращения 06.06.2016)
5. Reed A. Learning XNA 4.4. Game development for the PC, XBOX 360 and Windows Phone 7. –O’Reilly Media, 2011. –540p.
6. Robocode [Электронный ресурс] URL: http://robocode.sourceforge.net/ (дата обращения 06.06.2016)
7. Some expert system need common sense [Электронный ресурс] URL: http://www-formal.stanford.edu/jmc/someneed/someneed.html (дата обращения 10.06.2016)
8. Google Self-Driving Car Project [Электронный ресурс] URL: http://www.google.com/selfdrivingcar/ (дата обращения 10.06.2016)
9. Melomiks [Электронный ресурс] URL: http://geb.uma.es/melomics (дата обращения 10.06.2016)
10. Albanary J. C# 6.0 in a Nutshell –O’Reilly Media, 2015. –1133p.

# Приложение 1

Конфигурация для задачи генетического программирования в формате XML.

<Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<GeneticConfig>

<PopulationSize>100</PopulationSize>

<CrossoverProb>0.5</CrossoverProb>

<MutationProb>0.1</MutationProb>

<PanmixiaRatio>0.4</PanmixiaRatio>

<InbreedRatio>0.3</InbreedRatio>

<OutbreedRatio>0.3</OutbreedRatio>

<MaxStrategySize>1000</MaxStrategySize>

</GeneticConfig>

<MapConfig>

<Width>30</Width>

<Height>20</Height>

<ObstaclesCount>100</ObstaclesCount>

<EnemiesCount>30</EnemiesCount>

<ViewArea>10</ViewArea>

<FireArea>5</FireArea>

</MapConfig>

<StrategyGeneratorConfig>

<CloseBracketCoefficient>0.5</CloseBracketCoefficient>

<NewFunctionCoefficient>0.7</NewFunctionCoefficient>

</StrategyGeneratorConfig>

</Configuration>

# Приложение 2

Вероятностная программа генерации стратегий для начальной популяции.

public Strategy GenerateProgram ()

{

var result = new List<string>();

var functionDepth = 0;

var rnd = new Random(Guid.NewGuid().GetHashCode());

var i = 0;

int length = rnd.Next(1, maxLength);

while (i < length)

{

var p = rnd.NextDouble();

if (functionDepth != 0 && p < closeBracketCoeff)

{

result.Add("}");

functionDepth--;

i++;

continue;

}

int index;

if (p < newFunctionCoeff)

{

index = rnd.Next(StrategyTokens.FunctionSet.Count);

result.Add(StrategyTokens.FunctionSet[index]);

functionDepth++;

i++;

}

index = rnd.Next(StrategyTokens.TerminalSet.Count);

result.Add(StrategyTokens.TerminalSet[index]);

i++;

}

for (i = 0; i < functionDepth; i++)

result.Add("}");

return new Strategy(result);

}

# Приложение 3

Вывод статистических данных для карты, данной в приложении 1.

2016-06-12 04:47:40,135 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Genetic Programming Started

2016-06-12 04:47:55,782 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 104

2016-06-12 04:47:55,782 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Finish creating initial population

2016-06-12 04:47:55,954 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 104

2016-06-12 04:47:55,954 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #1

2016-06-12 04:47:55,954 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -110, Max fitness: 276, Average: 21,58

2016-06-12 04:47:56,453 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 246

2016-06-12 04:47:56,453 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #2

2016-06-12 04:47:56,453 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 276, Average: 46,83

2016-06-12 04:47:57,716 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 246

2016-06-12 04:47:57,732 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #3

2016-06-12 04:47:57,732 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 363, Average: 58,63

2016-06-12 04:47:58,340 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 245

2016-06-12 04:47:58,356 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #4

2016-06-12 04:47:58,356 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 363, Average: 72,67

2016-06-12 04:47:58,590 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 107

2016-06-12 04:47:58,590 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #5

2016-06-12 04:47:58,590 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 363, Average: 87,53

2016-06-12 04:47:59,136 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 334

2016-06-12 04:47:59,152 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #6

2016-06-12 04:47:59,152 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -110, Max fitness: 363, Average: 99,2

2016-06-12 04:47:59,776 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 185

2016-06-12 04:47:59,776 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #7

2016-06-12 04:47:59,776 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 347, Average: 126,17

2016-06-12 04:48:00,524 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 412

2016-06-12 04:48:00,524 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #8

2016-06-12 04:48:00,524 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -126, Max fitness: 347, Average: 131,33

2016-06-12 04:48:01,024 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 255

2016-06-12 04:48:01,024 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #9

2016-06-12 04:48:01,024 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -102, Max fitness: 352, Average: 142,56

2016-06-12 04:48:01,601 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 285

2016-06-12 04:48:01,601 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #10

2016-06-12 04:48:01,601 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -90, Max fitness: 362, Average: 141,19

2016-06-12 04:48:02,287 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 329

2016-06-12 04:48:02,287 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #11

2016-06-12 04:48:02,287 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -116, Max fitness: 362, Average: 129,76

2016-06-12 04:48:02,802 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 262

2016-06-12 04:48:02,802 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #12

2016-06-12 04:48:02,802 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -116, Max fitness: 362, Average: 118,03

2016-06-12 04:48:03,520 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 270

2016-06-12 04:48:03,520 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #13

2016-06-12 04:48:03,520 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 384, Average: 125,76

2016-06-12 04:48:03,800 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 136

2016-06-12 04:48:03,800 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #14

2016-06-12 04:48:03,800 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 384, Average: 126,49

2016-06-12 04:48:04,705 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 228

2016-06-12 04:48:04,721 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #15

2016-06-12 04:48:04,721 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -106, Max fitness: 352, Average: 128,27

2016-06-12 04:48:05,360 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 361

2016-06-12 04:48:05,360 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #16

2016-06-12 04:48:05,360 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -126, Max fitness: 352, Average: 134,42

2016-06-12 04:48:05,688 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 139

2016-06-12 04:48:05,688 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #17

2016-06-12 04:48:05,688 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -126, Max fitness: 352, Average: 120,4

2016-06-12 04:48:06,421 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 223

2016-06-12 04:48:06,421 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #18

2016-06-12 04:48:06,421 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -126, Max fitness: 360, Average: 125,41

2016-06-12 04:48:07,014 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 158

2016-06-12 04:48:07,014 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #19

2016-06-12 04:48:07,014 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -126, Max fitness: 360, Average: 135,79

2016-06-12 04:48:07,498 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 137

2016-06-12 04:48:07,498 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #20

2016-06-12 04:48:07,498 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -102, Max fitness: 378, Average: 136,18

2016-06-12 04:48:08,683 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 142

2016-06-12 04:48:08,683 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #21

2016-06-12 04:48:08,683 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -90, Max fitness: 378, Average: 137,04

2016-06-12 04:48:09,385 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 369

2016-06-12 04:48:09,416 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #22

2016-06-12 04:48:09,416 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -102, Max fitness: 402, Average: 156,47

2016-06-12 04:48:10,181 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 127

2016-06-12 04:48:10,181 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #23

2016-06-12 04:48:10,181 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -102, Max fitness: 10379, Average: 267,07

2016-06-12 04:48:12,396 [1] INFO GeneticProgramming.Program – 98

2016-06-12 04:48:12,396 [1] INFO GeneticProgramming.Program – Average time: 19676,7272727273

2016-06-12 04:48:12,396 [1] INFO GeneticProgramming.Program – Success: 98,989898989899%

2016-06-12 04:48:12,396 [1] INFO GeneticProgramming.Program – Average Generations: 12,5757575757576

2016-06-12 04:48:12,396 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Genetic Programming Started

2016-06-12 04:48:22,177 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 79

2016-06-12 04:48:22,177 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Finish creating initial population

2016-06-12 04:48:22,302 [1] INFO GeneticProgramming.Genetic.Engine.Population - Parallel update strategies: 86

2016-06-12 04:48:22,302 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Generation #1

2016-06-12 04:48:22,302 [1] INFO GeneticProgramming.Genetic.GeneticLogger - Min fitness: -116, Max fitness: 10195, Average: 257,68

2016-06-12 04:48:22,489 [1] INFO GeneticProgramming.Program - 99

2016-06-12 04:48:22,505 [1] INFO GeneticProgramming.Program - Average time: 19580,87

2016-06-12 04:48:22,505 [1] INFO GeneticProgramming.Program - Success: 99%

2016-06-12 04:48:22,505 [1] INFO GeneticProgramming.Program - Average Generations: 12,47

2016-06-12 04:48:22,910 [1] INFO GeneticProgramming.Program - 6102 484 5126 3847 3807 2547 360 50700 19133 1021 1020 1151 975 926 1112 2696 239101 53016 300584 76806 8399 1873 2265 1446 267 989 1023 13958 9465 35572 22289 2122 5753 1167 1380 2411 4571 4633 1410 19256 7118 4661 10441 5044 838 5656 3174 2541 2398 5235 33115 10982 43830 13993 21077 4515 8807 15543 7479 1284 195794 56356 1961 1635 1481 26468 13044 51571 38992 19707 91432 33069 1658 7422 1104 1040 1054 1257 15508 11165 4182 1507 2317 2207 70770 22094 3726 2347 18694 5004 8517 5598 5602 27749 12063 1913 872 30370 32252 10091

2016-06-12 04:48:22,910 [1] INFO GeneticProgramming.Program - True True True True True True True True True True True True True True True True True True False True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True True

2016-06-12 04:48:22,942 [1] INFO GeneticProgramming.Program - 2 2 18 2 17 5 2 66 2 2 2 2 2 2 2 2 126 2 198 2 14 2 5 2 2 2 2 17 2 30 7 2 2 2 2 2 8 4 2 18 2 4 8 2 2 6 2 3 2 2 32 2 29 2 14 2 11 17 2 2 132 2 2 2 2 13 6 27 29 7 71 2 2 2 2 2 2 2 16 5 2 2 2 3 40 7 2 2 13 2 9 2 4 22 7 2 2 39 24 2

1. См. Приложение 1. [↑](#footnote-ref-1)
2. Рассматриваются все танки на поле боя: главный танк и танки соперника. [↑](#footnote-ref-2)
3. См. Приложение 1. [↑](#footnote-ref-3)
4. См. Приложение 1. [↑](#footnote-ref-4)
5. См. Приложение 3 [↑](#footnote-ref-5)