НИУ ИТМО

Отчет
по работе с эволюционными алгоритмами
с целью нахождения оптимальных начальных стратегий

Автор:

Береговенко Илья Игоревич M32341

2021 год

Цель работы: провести ряд исследований с эволюционных алгоритмов в области простейших настольных игр с целью выявления наиболее оптимальных начальных стратегий для игрока, в результате которых игру можно свести к выигрышу или же ничьей.

Задачи:

1) Написать код, реализующий среду какой-либо настольной игры/каких-либо игр, а также процесс эволюционирования программ;
2) Постараться подробно описать каждый из этапов разработки программы;
3) Провести ряд экспериментов против компьютерного игрока, чтобы определить на сколько рабочим оказался результат программы;
4) Провести серию экспериментов с целью выявления работоспособности программы при изменении ранее константных параметров.

Акт I

Пролог или изначальные мытарства

После определенного времени, отведенного на освоение основных постулатов эволюционных вычислений, работа над проектом началась. Изначально возник вполне банальный вопрос – вопрос о выборе игры, которая была бы в сравнительной степени простой, что могло бы ускорить написание кода и реализации мутаций и кроссинговера. После непродолжительных пленарных заседаний с научным руководителем проекта было решено выбрать не карточные игры (кои я хотел реализовать изначально), что могли вызвать ряд осложнений в определении фитнесс-функции, а всеми любимые и знакомые «крестики-нолики» за авторством великого и могучего народного творчества.

Сразу после этого было решено работать с проектом последовательно, что логично. В начале была реализована среда игры: поле, проверка выигрыша чьей-либо стороны, ходы каждой из них, создание случайных игроков. Среда создается с помощью таких функций, как *check* (проверка поля на победу одной из сторон или же взаимной ничьи), *randomMove* (генерирует случайный ход «нашего» игрока), *randomPlayerMove* (генерирует случайных ход оппонента), *singleGame* (запускает одну игру с новым очищенным полем 3x3, предоставляя ходы по очереди и выводя победителя), *starter* (запускает несколько серий игр), *points* (начисляет 0 в случае проигрыша, 1 в случае ничьи и 3 в случаи победы «нашему» игроку). Лишь, когда была создана работоспособная версия игры, в кою мог бы сыграть и обычный ничем не примечательный живой человек, настала очередь внедрения методов эволюционных преобразований.

Как известно, основополагающими в эволюционных вычислениях являются три функции: кроссинговер, мутация и фитнесс – которые позволяют выработать через ряд поколений работоспособную популяцию стратегий. Но прежде чем анализировать каждую из них, определим для себя, что из себя будет представлять стратегия нашего игрока, которую будет исполнять функция *singleStrategy*. Последняя на вход получает два массива: strMat и board. В board-е хранится поле значений клеток, которые могут быть свободны (0) либо заняты одним из игроков (1 – ход нашей стратегии; 2 – случайный ход «бесшабашного» игрока). Массив strMat содержит стратегию, которая представлена в виде 9-ти наборов из 5-ти чисел в каждом. Первые три цифры из данного набора используются для проверки условия, которое мы будем проверять в функции *singleStrategy*. Они задают координаты проверяемой клетки (первые два числа) и ее значение (третье число). При выполнении данного условия игрок с нашей стороны поставит крестик в клетку, заданную последними двумя числами (четвертым и пятым), если, конечно, она окажется свободной. Использую два данных массива, функция *singleStrategy,* пробегается по каждому из 9-ти наборов. В результате мы либо поставим крестик в соответствии с нашей стратегией, либо сделаем случайный ход, если ни одно из наших условий не имело место быть или же каждая клетка, в которую мы порывались поставить крестик была занята.

Теперь же можно рассказать об устройстве каждой из трех основополагающих функций, позволяющих нам совершать эволюционные процессы. Начнем с фитнесс-функции. Она определяет успешность каждой из наших стратегий (каждая из которых, как мы помним, состоит из 9-ти наборов). На вход она получает массив из «сидов», определяемых набором из произвольных чисел в промежутке от 1 до 1000, и набор стратегий. В начале она заводит массив из нулей по кол-ву вышеописанных стратегий. Затем каждой из стратегий она позволяет сыграть с каждым из «сидов», что определяет поведение «сумасбродного» игрока в функции *randomPlayerMove*. При этом для отдельно взятой пары сида и стратегии будет проведена серия из 10 игр (это делается для того, чтобы снизить случайность выигрыша данной стратегии. К тому же это позволит более точно заявлять, что две стратегии играют в сравнении друг с другом лучше или хуже против случайного игрока с данным «сидом»). Далее результат каждой из игр, определяемый функцией *points* в виде числа от 0 до 3-х, прибавляется к значению ячейки ранее заведенного массива, в которой лежат результаты игр той стратегии, что играла в данной партии. По итогу вышеописанных действий функция *fitness* выведет массив чисел, значение каждой из клеток в котором не будет превышать числа 30*(кол-во сидов).

Отвлечемся на некоторое время от описания функций, ибо я хотел бы уделить отдельное внимание вышеупомянутым «сидам». Делом в том, что

при начале работы над данным проектом я совершил в достаточной степени глупый шаг – мой случайный игрок совершал просто случайные ходы. С первого взгляда достаточно трудно уловить почему это является столь большой ошибкой, ибо оппонент нашей стратегии и должен быть… случайным. Но делом в том, что на не нужен совершенно случайный игрок, ибо это будет противоречить инварианту фитнесс-функции – она должна быть неизменной. Иными словами, наш оппонент, который совершает «случайные» ходы в игре с данным игроком, должен будет повторить их же и с другим. Если этого не сделать, наш эволюционный процесс будет напоминать не плановую селекцию, а беспорядочную мутацию сравни той, что подарила гулей в Fallout. Вдобавок ко всему данные ходы (ну, или же стратегии «случайных» игроков) должны будут совершаться и в следующем поколении. Однако в самом начале я этого не учел и пребывал в некотором смятении, когда иной раз поколения моих программ лишь изредка выигрывали хотя бы одну партию, а иногда поголовно побеждали абсолютно во всех играх! Масло в огонь подливало еще и то, что эти самые «победоносные» стратегии чаще всего либо были нерабочими (не позволяли сыграть даже в ничью), либо же в подавляющем большинстве требовали от вас исключительно случайных ходов, что как-то слабо походило на хорошую начальную стратегию. По этой причине перед запуском цикла поколений создается полноценный массив из «сидов», каждый из которых всегда будет задавать одну и ту же последовательность случайных чисел, что и позволит нам выработать работоспособную фитнесс-функцию, сохраняющую собственный инвариант.

Возвращаясь к двум оставшимся столпам эволюционных вычислений, рассмотрим мутацию, представленную в виде функции *mutation* (иногда я сам удивляюсь своей оригинальности). На вход она получает отдельно взятую стратегию и приходясь через каждый набор из 5-ти чисел с вероятностью в 4% меняет в нем что-то. Либо это может быть набор x и y, что мы используем для изначальной проверки как-либо клетки, либо же символ, наличие которого мы проверяем, либо координаты клетки (пару чисел, как и в прошлый раз), куда мы планируем поставить крестик, если условие выполняется. При этом числа, которыми мы заменяем изначальные значения в наборе, выбираются так же случайно без каких-либо закономерностей. Каждая новая особь подвергается этому процессу.

Наконец, рассмотрим функцию кроссинговера представленный функцией *cross*. Т.к. общее число популяции составляло 16 особей (для удобства дальнейших вычислений) было решено сделать так, чтобы по итогу кроссинговера новая популяция состояла из 14 новых особей, полученных в результате скрещивания, и 2-х лучших старых. Сделано это было с той целью, чтобы не попасть в «эволюционный тупик» - случай, когда, массово

эволюционировав, мы сделали «поворот не туда» и получили безвозвратное поколение твердолобых особей, что потенциально будут всегда делать не совсем корректные ходы. Сам процесс отбора 7-ти пар особей для дальнейшего скрещивания представлен в виде турнира с некоторыми оговорками. Каждая из пара особей соревнуется друг с другом по результатам фитнесс функции. При этом особь, что имеет более маленькое значение фитнесс функции так же имеет шанс выиграть на каждом из этапов турнира. Это обеспечивается тем, что особи с большим значением функции имеют шанс пройти на новый этап «соревнований» с вероятностью в 90%, а с меньшим – в 10%. Данные ухищрения совершаются с той целью, чтобы развивать «рецессивные» гены, которые в дальнейшем помогут улучишь стратегию. Далее после того, как остается лишь пара особей, совершается скрещивание стратегий функцией *crossing*. На вход она получает две ранее описанные особи, а также случайное число – место в массиве, после которого особи заменят свои гены (наборы из 5-ти чисел) на гены своего партнера. Далее новые пары, ранее уже скрещенные, мутирую (если повезет), сохраняются, как и их значения в фитнесс-функции, в новом наборе, а далее процедура повторяется еще 6-ть раз. Затем в набор стратегий добавляется еще 2 лучшие старые особи с заранее подсчитанными фитнесс-функциями, после чего возвращаются набор стратегий и значения их фитнесс-функций.

Последними действительно важными моментами кода, на которые хотелось бы обратить внимание являются функции *starter* и *final*. *Starter* создает стартовые наборы стратегий (совершенно пустые, где значение каждого из полей это ноль), массив значений «сидов», а так же запускает очередное поколение программ (при написания данной работы было решено ограничиться 1000-м поколением, хотя порой я не удерживался доходил аж до 10000-ого) и останавливает эволюционные процессы либо когда программа доходит до «крайнего поколения», либо когда одна из ячеек массива фитнесс-функции достигает значения 30*(кол-во сидов). *Final* же можно считать скорее за произвольную оценку. Функция на базе полученной популяции стратегий их в очередной раз запускает фитнесс функцию с ними правда уже на совершенно других «сидах», выбранных на промежутке от 1000 до 2000, в кол-ве 20 штук. Так же берется совершенно иная система начисления очков (2 в случае ничье и 5 в случае победы). Все это делается для того, чтобы в конечном счете преобразовать уже нам знакомый массив результатов фитнесс-функции в массив вероятностей выигрыша против случайного игрока. Это достигается за счет деления значения в каждой из ячеек массива на 10 (20*10*5 = 1000, т.к. 20 – кол-во «сидов»; 10 – кол-во партий сыгранных на одном «сиде»; 5 – максимальный балл за одну игру). Таким образом при желании мы можем выбрать наиболее часто «срабатывающую» стратегию.

На этом описание кода программы можно завершить.

Акт II

Создание работоспособного выводка или «Оно играет!!!»

Вышеописанный код, который будет представлен перед вами в программе, стал результатом долгих и в некоторой степени мучительных результатов работы. Показательной была ситуация с «сидами», которую я описал выше, однако, как вы можете догадаться она была не единственной.

Например, долгое время я не мог понять почему раз за разом программы выводила 0 результат в каждой графе при игре с оппонентом. Мне понадобилась парочка дней, чтобы понять, что стратегия отдельно взятой особи должна включать случайный ход, в обратном случае особи либо зависали, либо делали максимально невразумительные ходы, сохранявшие одиозность изначального расклада.

Не менее интересной была история и с начальным раскладом. Когда я только начал писать код, я подумал, что будет замечательным заранее пронумеровать каждую из клеток изначального набора (те, что были бы использованы для проверки условий) 0-0, 0-1 и т.д. На поверку это оказалось одним из самых невразумительных действий. С какой стати ход 1-2-0-0-0 это разумный ход на старте? Нет, конечно, мы не знаем хороший он или плохой. Если «хороший», то это отлично, но насколько мы можем быть уверены в оставшихся 8-ми? По этой самой причине не удивительно, что иной раз программы приходили в тупик. Эволюционно они честно пытались возвысится, но гены, которыми я их обременил вначале, неуклонно тянули этих бедолаг на дно, как наследственного молодого гопника к «Балтике 9».

Дальше больше. Возникли проблемы с оценкой. Не сказать, что она встала ребром, но много чего подпортила. Изначально функция *points* должна была выводить 2-ку на выигрыш и это было не самым хорошим ходом. С одной стороны, это было логичным: 2 ничье равны по силе 1 полностью выигрышной партии. Однако это было бы справедливо в том случае, если бы мы играли против такого же осознанного игрока, как если бы это был человек, но мы играем против «сумасброда», который подчиняется какой-то логике, которую мы характеризуем, как хаотичную. По этой самой причине мы должны поощрять особь в том, что она обыграла оппонента. Тем самым мы лучше развиваем «победоносные» гены, ибо «ничейные» нас волнует в сравнительной степени меньше.

Немаловажен и факт того, как была создана система стратегий. Изначально мы предполагали сделать ее с помощью автомата, однако вскоре поняли

можно будет обойтись и более простой моделью, которую мы нежно обозвали «палочки», ибо, если условие выполнялось на конце одной из «палочек», мы переходили к другому концу.

Можно было бы так же упомянуть тот факт, что изначально я предполагал не вносить условие на проверку «свободности» клетки, в которую мы бы поставили крестик. Изначально я предполагал, что программа сама сможет через годы эволюций понять, что ставить крест в уже занятую позицию нецелесообразно. В следствии этого нередки были случаи, когда мой крест наглейшим образом стирал ноль оппонента и почему-то (действительно почему?) стабильно выигрывал партию за партией, хотя ни одна из этих стратегий не выдерживала полевых условий в борьбе с «Крестиками-ноликами» компании Google на среднем уровне сложности.

Последнее, что можно ныне припомнить при создании работоспособной программы, это мое стремление к увеличению числа поколений. В моем понимание сложилось странное убеждение, что чем дальше заходит эволюция тем более приспособленными, становятся особи. Следуя данной логике, я долгое время выводил все дальние поколения особей. Однако учитывая тот факт, что код моей программы не был к тому моменту в достаточной степени отлажен, я столкнулся с проблемой, что программы вели себя архаично. Особенно в тот момент времени, когда я еще не успел ввести «сиды», даже на 2000 поколении результат был невразумительным. Числа скакали. Они приближались к 1000 (замечу, что тогда у меня существовала другая система начисления очков, которая в никакое сравнение не шла даже с вышеописанными), а в следующем поколении падали до 0. Понятное дело, что рост числа поколений ни к чему не приводил. С того момента было решено ограничиться 1000-м. Вдобавок ко всем мой научный руководитель порекомендовал мне ввести в тело эволюционного цикла проверку на 600 (напомню, что это максимальное число которое мы можем получить при игре с 20 оппонентами 10 раз и начислением 3-х очков за победу), ибо фитнесс-функция более не сможет присвоить какой-либо стратегии большее значение, т.к. мы уже достигли максимума.

После описания вышеописанных проблем, с которыми мне пришлось столкнуться, я бы хотел сконцентрировать ваше внимания на сами стратегии, чтобы вы смогли пронаблюдать за развитием популяции моих программ.

Первые стратегии были причудливыми.

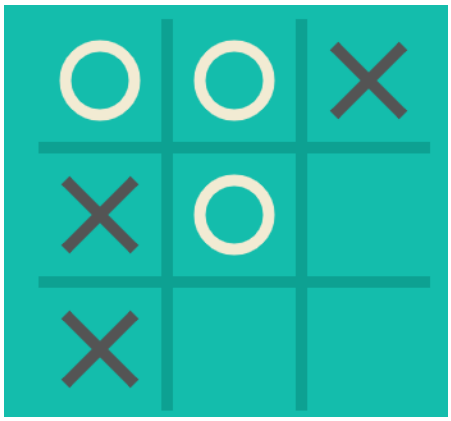
Они толком никак не отличались от исходной стратегии, где каждая из строк представлена в виде набора 0-0-0-0-0. Другие же были более странными. В них были ходы, которые противоречили сами себе, что приводило к блокировкам. Их здесь я прописывать не стану.

Первой стратегией, что смогла хотя бы делать в меру пристойные ходы, не скатывавшиеся в случайность, оказалась следующая:

x	y	symb	x_pl	y_pl
1	1	O	0	0
0	2	O	0	1
2	2	O	0	2
1	1	Nu	1	0
0	2	O	1	1
1	2	O	1	2
0	0	Nu	2	0
1	1	X	2	1
0	1	Nu	2	2

Однако она так и не стала «победоносной» или хотя бы «ничейной». Из 100 (при старой системе начисления очков) баллов она смогла набрать всего 14.

Первая игра: проигрыш



Оставшиеся проведенные две игры не стану записывать, ибо они повторяют вышеописанную. Все они проигрышные. Концентрировать внимание на ней не станем.

Вместо этого обратим внимание на более работоспособные стратегии. Первой из удачных оказалась та, что я вывел на 2000-ом поколении (этот был тот самый период, когда я полагал, что чем дальше зайдет эволюция, тем лучше окажутся особи).

x	y	symb	x_pl	y_pl
2	0	Nu	0	0
2	1	Nu	0	1
1	1	O	0	2
1	1	X	1	0

1 0 O 1 1

2 0 O 1 2

1 1 O 2 0

0 1 X 2 1

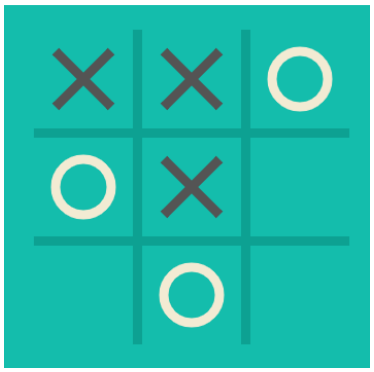
0 0 X 2 2

Проверкой стратегий займемся на google-ких «Крестиках-ноликах» на среднем уровне сложности.

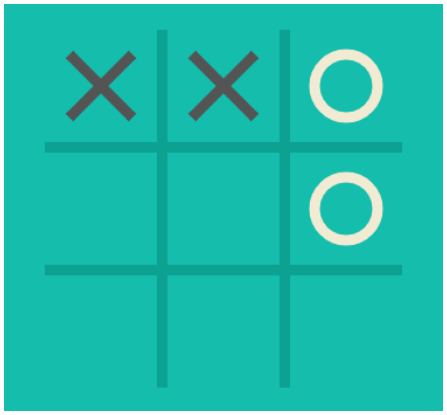
Т.к. мы стремимся выработать начальную, а не полноценную стратегию, логичным будет брать игру под собственный контроль в тех случаях, когда мы видим, что текущий (наш) ход является выигрышным.

Проведем серию из трех игр против компьютера.

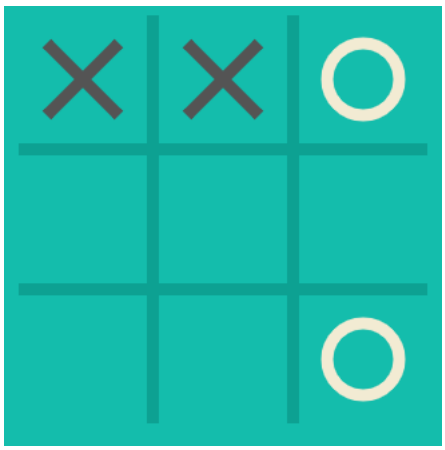
Первая игра: выигрышная



Вторая игра: проигрышная (если целиком и полностью следовать стратегии)



Третья игра: проигрышная (если целиком и полностью следовать стратегии)



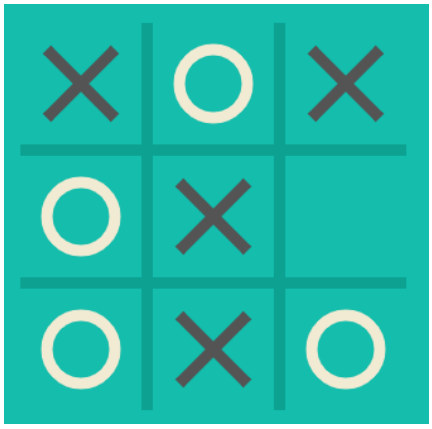
Видно, что лишь в одном случае из 3-х стратегия оказалась выигрышной.

Новым шагом эволюции стала следующая стратегия стратегией оказалась следующая:

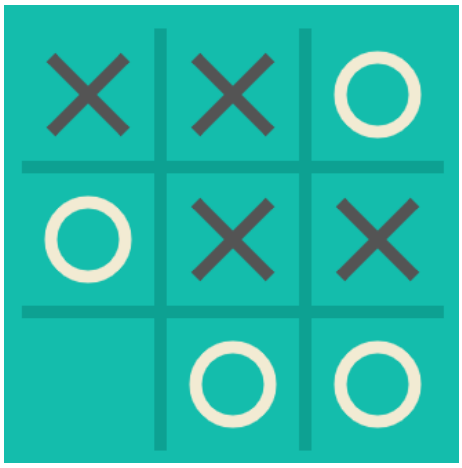
x	y	symb	x_pl	y_pl
1	1	X	2	2
0	1	X	1	2
2	0	X	2	0
0	0	O	0	2
0	1	O	1	1
1	1	X	0	2
0	0	Nu	0	0
1	1	O	0	1
0	1	Nu	0	2

При запуске нескольких игр с данной стратегией была обнаружена забавная закономерность.

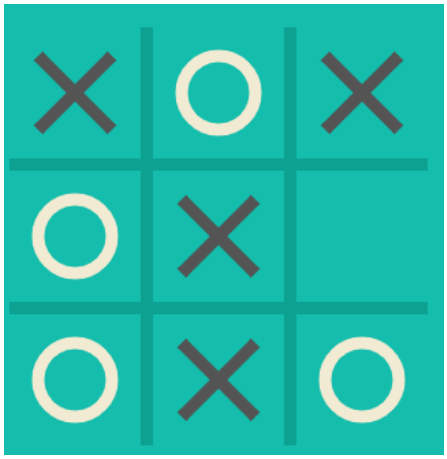
Первая игра: ничья



Вторая игра: ничья



Третья игра: ничья



Легко заметить, что стратегия эта стабильно может выводить игру в ничью, если в момент, когда появляется возможность сделать случайный ход, сделать осознанный шаг, который не будет восприниматься за поддавание оппоненту. Появилась она после введения «сидов», что позволило натаскивать программу на конкретного противника.

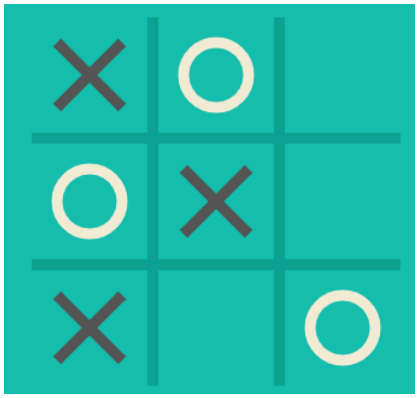
Но на этом я не хотел останавливаться. Мне казалось, что существует возможность обыграть противника на среднем уровне сложности, ибо программа до этой смогла выиграть, хотя и проиграла после этого дважды.

Благо вскоре выяснилось, что я не совсем корректно высчитывал фитнесс-функцию. Говоря более конкретно, каждый из стратегий играла не против каждого из 20-ти «сидов», а против одного конкретного определяемого по модулю 20 от номера итерации. Например, на 123 итерацию каждая играла 10 раз (предыдущая стратегия появилась именно за счет того, что было введено 10 игр с одним «сидом» для одной конкретной тактики) против «сида» под номером 3. Это приводило к тому, что каждая из стратегий не могла приспособиться ни к какой из тактик противника, ибо каждый последняя менялась.

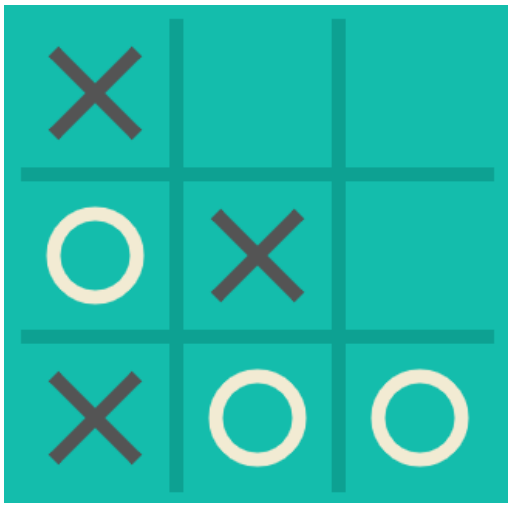
Осознав это, я решил исправить данную оказию. Ныне же каждая из стратегий имела возможность сыграть с каждым из «сидов» 10 раз при подсчете одной фитнесс-функции. В результате получилась следующая стратегия:

Запуск нескольких игр с данной стратегией оказался примечательным.

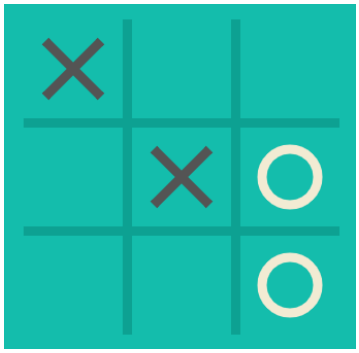
Первая игра: выигрыш



Вторая игра: выигрыш



Третья игра: выигрышная (если сделать один свой ход) / проигрышная (продолжить следовать стратегии)



В результате мы можем обнаружить, что данная стратегия оказалась наиболее рабочей из всех, ибо смогла пройти накопить необходимое кол-во знаний за период эволюционирования. Стратегия начала работать на столько хорошо, что появилась возможность с вероятность чуть более 2/3 выиграть осознанного противника на среднем уровне сложности.

Примечателен тот факт, что в момент выработки рабочего поколения программ меня, как и моего научного руководителя, стало смущать то, что программы в принципе доходили до значений 500 и даже 600. Дело в том, что, как нам всем известно, при безошибочной игре в «крестики-нолики» люди обычно сводят все в ничью. Иными словами, в нашем представлении программы не должны были выходить дальше значения 300 или же на худой конец 400. Однако лишь спустя какое-то время мы догадались, что данный вывод был бы справедливым, если бы мы играли против реального, осознанного игрока, коим просто на просто не мог явиться случайный. Другими словами ходы, которые совершал игрок, хоть и подчинялись как-то логике, но все же носили не стратегический, осознанный логикой характер, а более архаичный, как у обезьяны с гранатой.

По итогу, у меня родился работоспособный алгоритм действий, который можно будет спокойно применять (или сгенерировать снова) против среднестатистического игрока в крестики-нолики, не обладающего глубокими знаниями по математике и какой бы то ни было аналитике, одним словом, против паренька перешедшего из технического в гуманитарный вуз или против ребят учащихся на экономистов, искренне верящих, что они знают математику.

По итогу данного раздела подобно многим известному доктору Виктору Франкенштейну, сидя перед стратегией, что была собрана из кусков совершенно несвязанных друг с другом кусков, которая наконец-то зашевелилась и начала приносить плоды, я могу заорать во все горло:

«Оно работает!!! Оно живое!!! Оно играет!!!»

Акт III

Рубрика «эксперименты» или «Давай поковыряем это!»

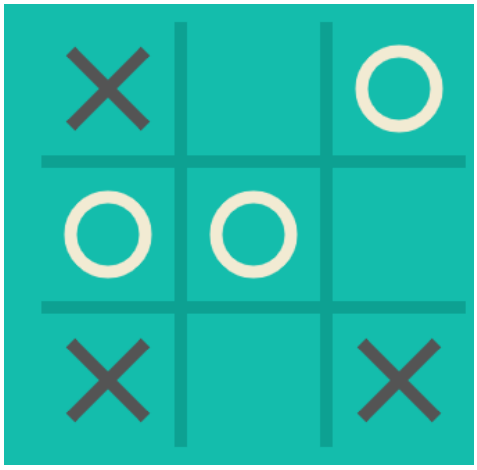
Финалом наших исследований должна стать серия экспериментов, направленная на выявление зависимостей в работоспособности программы при изменении некоторых параметров мною настолько тщательно описанной системы.

Начнем с изменения числа «сидов», против которых будет играть каждая из стратегий. Для начала увеличим их число с 20 до 50.

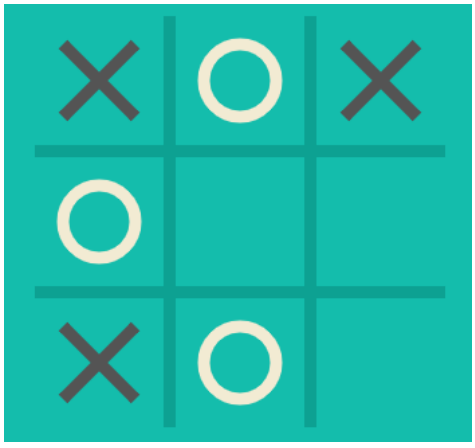
Получим следующую стратегию:

x	y	symb	x_pl	y_pl
1	2	O	0	1
2	0	X	1	0
0	1	Nu	0	0
1	0	Nu	2	0
0	0	Nu	2	2
2	2	X	1	1
2	0	O	2	2
2	0	O	0	0
2	1	Nu	0	2

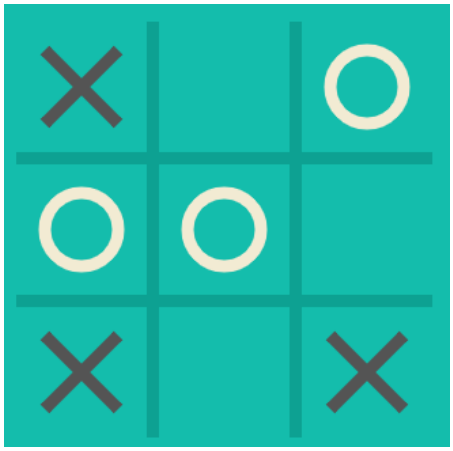
Первая партия: выигрыш (при «рациональном» случайном ходе)



Вторая партия: выигрыш (при «рациональном» случайном ходе)



Третья партия: выигрыш (при «рациональном» случайном ходе)



Можно заметить, что стратегия оказалась рабочей, однако в отличии от других случаев здесь приходиться совершать специальный «рациональный» ход в момент предоставления случайного шага, что будет не слишком удобным обстоятельством для пользователей, которые хотят получить обычную стратегию.

Теперь же уменьшим кол-во «сидов» до 4-х.

Тогда программа выведет следующую стратегию:

x y symb x_pl y_pl

2 2 O 2 1

0 0 X 0 0

2 0 Nu 0 0

0 0 Nu 0 0

2 0 Nu 2 0

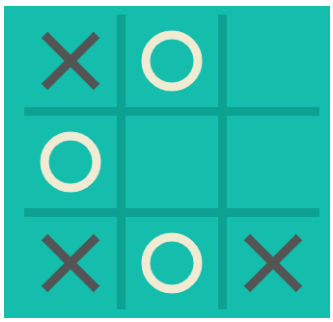
0 2 Nu 0 1

2 0 O 0 2

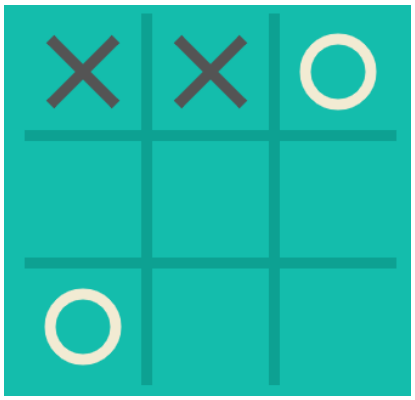
0 0 X 1 0

0 2 O 2 2

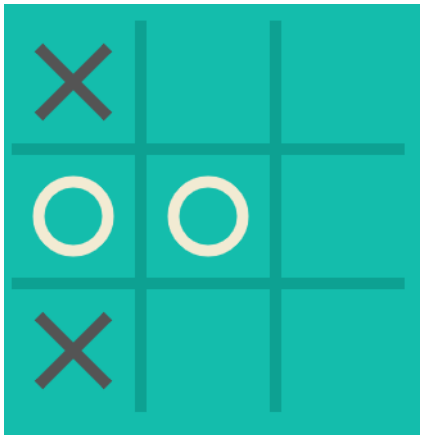
Первая игра: победа (при «рациональном» ходе)



Вторая игра: проигрыш



Третья игра: проигрыш (при точном следовании стратегии)



Нетрудно заметить, что программа при меньшем размере числа «сидов» стала выводить более «халтурные» стратегии, которые либо требуют от вас «рациональных» ходов, до коих не все могут догадаться, либо же стратегия, которую мы верно исполняем, приведет нас к проигрышу. Вывод напрашивается сам: при меньшем числе «сидов» вырабатывается менее качественная стратегия, ибо примеров, на которых ей нужно будет учиться, слишком мало.

Теперь же проанализируем фактор влияния длительности эволюционной цепочки. Снизим число итераций с 1000 до 400.

x y symb x_pl y_pl

2 0 X 1 2

0 0 Nu 2 2

0 2 Nu 1 2

0 0 O 1 2

1 1 X 0 0

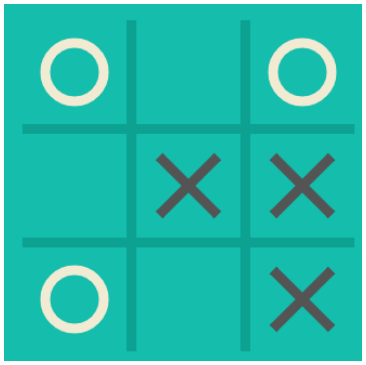
2 2 O 1 0

0 0 O 0 2

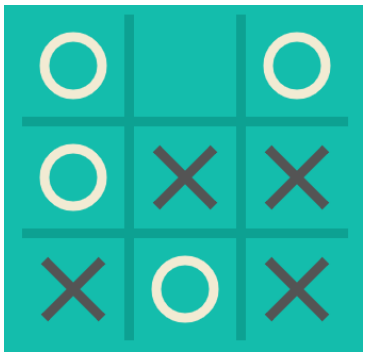
1 0 Nu 1 1

0 1 Nu 2 2

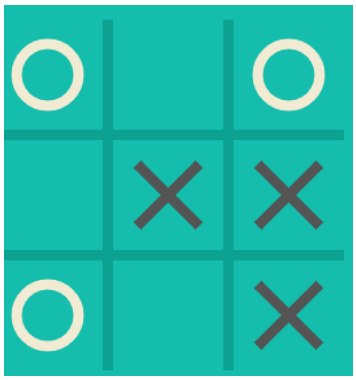
Первая игра: выигрыш



Вторая игра: ничья



Третья игра: проигрыш



Заметим, что в подобных условиях алгоритм ведет себя в достаточной степени архаично. В отличии от ранее рассмотренных мы можем заметить, что данный работает непредсказуемо. Если первые тактики из II-го акта стабильно проигрывали или же играли в ничью, то этот с равной верностью может как проиграть, так и выиграть. На лицо тот факт, что программа уже начала эволюционировать, однако не успела дойти до того момента, когда бы смогла выводить игры с большей вероятностью хотя бы в ничью.

В результате можно заключить, что уменьшение числа итераций цикла эволюции ведет к ухудшению качества конечных стратегий.

Вывод

В результате проведенной работы можно заключить, что была создана программа, которая способна выводить эволюционным путем качественные стратегии для людей, которые совершенно не разбираются в последних. Данное подтверждает серия экспериментов, проведенная мной ранее. Разумеется работа, как и программа, требует дальнейших исследований: так, например, нужно будет проверить влияние размера поля на качество выводимых стратегий и анализ схожих игр. Однако я полагаю, что эти темы заслуживают отдельного отчета, который должен будет продолжить мои микроскопические исследования.