СОДЕРЖАНИЕ

ВВЕДЕНИЕ

За последнее время искусственный интеллект стал использоваться в самых различных областях. Одним из направлений которого является генетический алгоритм. Он используется для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Отличительной особенностью данного алгоритма является акцент на использовании оператора скрещивания, который производит операцию рекомбинации решений-кандидатов. В качестве основных этапов работы генетического алгоритма можно выделить:

1) генерация начальной популяции;
2) расчет оценок приспособленности;
3) селекция;
4) применение оператора скрещивания;
5) применение оператора мутации;
6) генерация следующего поколения.

Работа алгоритма продолжается до удовлетворения критерия остонова. Более подробно ознакомиться с алгоритмом можно в книге «An Introduction to Genetic Algorithms» [2].

Цель данной курсовой работы – реализовать генетический алгоритм для игры в тетрис в системе Wolfram Mathematica [3].

1. СОЗДАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ТЕТРИСА

Тетрис – известная компьютерная игра, представляющая собой головоломку, построенную на размещении падающих геометрических фигур «тетрамино» (рисунок 1) на игровом поле.

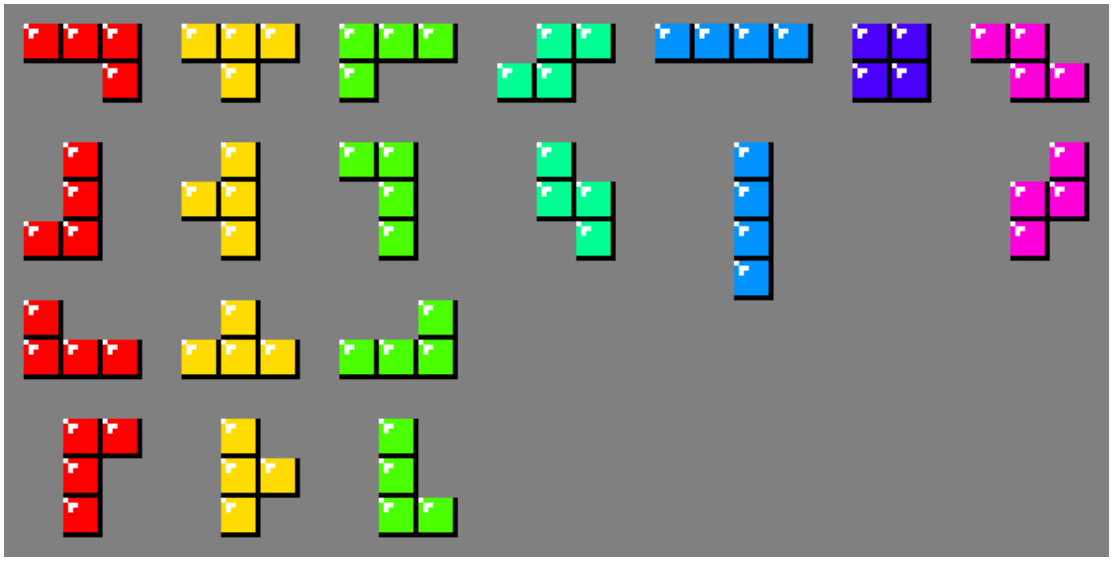

Рисунок 1 – Все возможные представления фигур

Изначально задается ширина и высота игрового поля, которые соответственно хранятся в переменных w и h. Игровое поле представлено матрицей $h \times w$, каждый элемент которой является списком, содержащим три элемента. Список соотносится с цветом согласно RGB модели. Изначально матрица заполнена следующим образом: первый столбец, последний столбец и первая строка, являющиеся границами, списком $br = \{0.3, 0.3, 0.3\}$ (серый цвет), а остальные элементы списком $bg = \{0, 0, 0\}$ (черный цвет). После чего создается список фигур $figs$, изображенный на рисунке 2, элементы которого являются списками из двух элементов: координаты фигуры и ее цвет, уровень, значение которого хранится в переменной $level$, увеличивается на один за каждые две удаленные строки, количество удаленных строк, значение которого хранится в переменной $lines$, счет, значение которого хранится в переменной $score$, увеличивается в зависимости от количества удаленных строк после одного хода:

1) при удалении одной строки $score \mathrel{+}= 40 * (level + 1)$;

2) при удалении двух строк $score \mathrel{+}= 100 * (level + 1)$;

3) при удалении трех строк $score \mathrel{+}= 300 * (level + 1)$;

4) при удалении четырех строк $score \mathrel{+}= 1200 * (level + 1)$.

```
figs = {(*figure:{coords,color}*)
   {{{0, -1}, {0, 0}, {0, 1}, {1, -1}}, {0.1, 0.1, 1.0}}(*J*),
   {{{0, -1}, {0, 0}, {0, 1}, {1, 1}}, {1.0, 0.5, 0.0}}(*L*),
   {{{1, 0}, {0, 0}, {1, -1}, {0, 1}}, {1.0, 0.0, 0.0}}(*Z*),
   {{{1, 0}, {0, 0}, {0, -1}, {1, 1}}, {0.1, 1.0, 0.1}}(*S*),
   {{{0, 1}, {0, 0}, {0, 2}, {0, -1}}, {0.1, 0.9, 1.0}}(*I*),
   {{{0, 0}, {1, 0}, {1, 1}, {0, 1}}, {1.0, 1.0, 0.1}}(*O*),
   {{{0, -1}, {0, 0}, {0, 1}, {1, 0}}, {0.9, 0.1, 1.0}}(*T*)};
```

Рисунок 2 – Список фигур

Изначально необходимо инициализировать переменные, для этого используется функция $init[]$. Генерируется матрица $glass$, матрица $nextglass$, которая необходима для отображения следующей фигуры, обнуляется счет, уровень и количество удаленных строк, в списке $fig7$ хранится случайная последовательность фигур, после чего в матрицу $nextglass[]$ помещается первый элемент списка $fig7$. Далее генерируется маска, которая представлена матрицей, содержащей логические значения, True, если клетка свободна, иначе False. Для генерации маски используется функция $newmask[]$, которая сравнивает каждый элемент матрицы $glass$ со списком bg и в случае равенства присваивает соответсвенному элементу матрицы $mask$ значение True, в противном случае значение False.

Для графического отображения используется встроенная функция $Graphics[]$, к которой применяется функция $DynamicModule[]$, для обновления изображения (рисунок 3). Поле “Moves” означает количество совершенных ходов в текущей партии.

Результат работы функции $Tetris[]$ изображен на рисунке 4.

```
Tetris[] := Dynamic[
   init[];
   Graphics[
    {
     Raster@Dynamic@glass[[ ;; -4]],
     Raster[Dynamic@nextglass, {{w, h - 7}, {w + 6, h - 3}}],
     Text[Style["Score", 24, White, Bold], {w, 17}, {-1, 0}],
     Text[Style[Dynamic@score, 24, White, Bold], {w + 6, 17}, {1, 0}],
     Text[Style["Lines", 24, White, Bold], {w, 15}, {-1, 0}],
     Text[Style[Dynamic@lines, 24, White, Bold], {w + 6, 15}, {1, 0}],
     Text[Style["Moves", 24, White, Bold], {w, 13}, {-1, 0}],
     Text[Style[Dynamic@movesTaken, 24, White, Bold], {w + 6, 13}, {1, 0}],
     Text[Style["Best", 24, White, Bold], {w, 11}, {-1, 0}],
     Text[Style[Dynamic@best, 24, White, Bold], {w + 6, 11}, {1, 0}],
     Text[Style["Worst", 24, White, Bold], {w, 9}, {-1, 0}],
     Text[Style[Dynamic@worst, 24, White, Bold], {w + 6, 9}, {1, 0}],
     Text[Style["Individual", 24, White, Bold], {w, 7}, {-1, 0}],
     Text[Style[Dynamic@individual, 24, White, Bold], {w + 6, 7}, {1, 0}],
     Text[Style["Generation", 24, White, Bold], {w, 5}, {-1, 0}],
     Text[Style[Dynamic@generation, 24, White, Bold], {w + 6, 5}, {1, 0}],
     Text[Style[Dynamic@msg, 24, White, Bold], {w + 3, 3}, {0, 0}]
    },
    PlotRange → {{0, w + 7}, {0, h - 2}},
    Background → RGBColor@br, ImageSize → 550]
  ];
```

Рисунок 3 – Визуализация тетриса

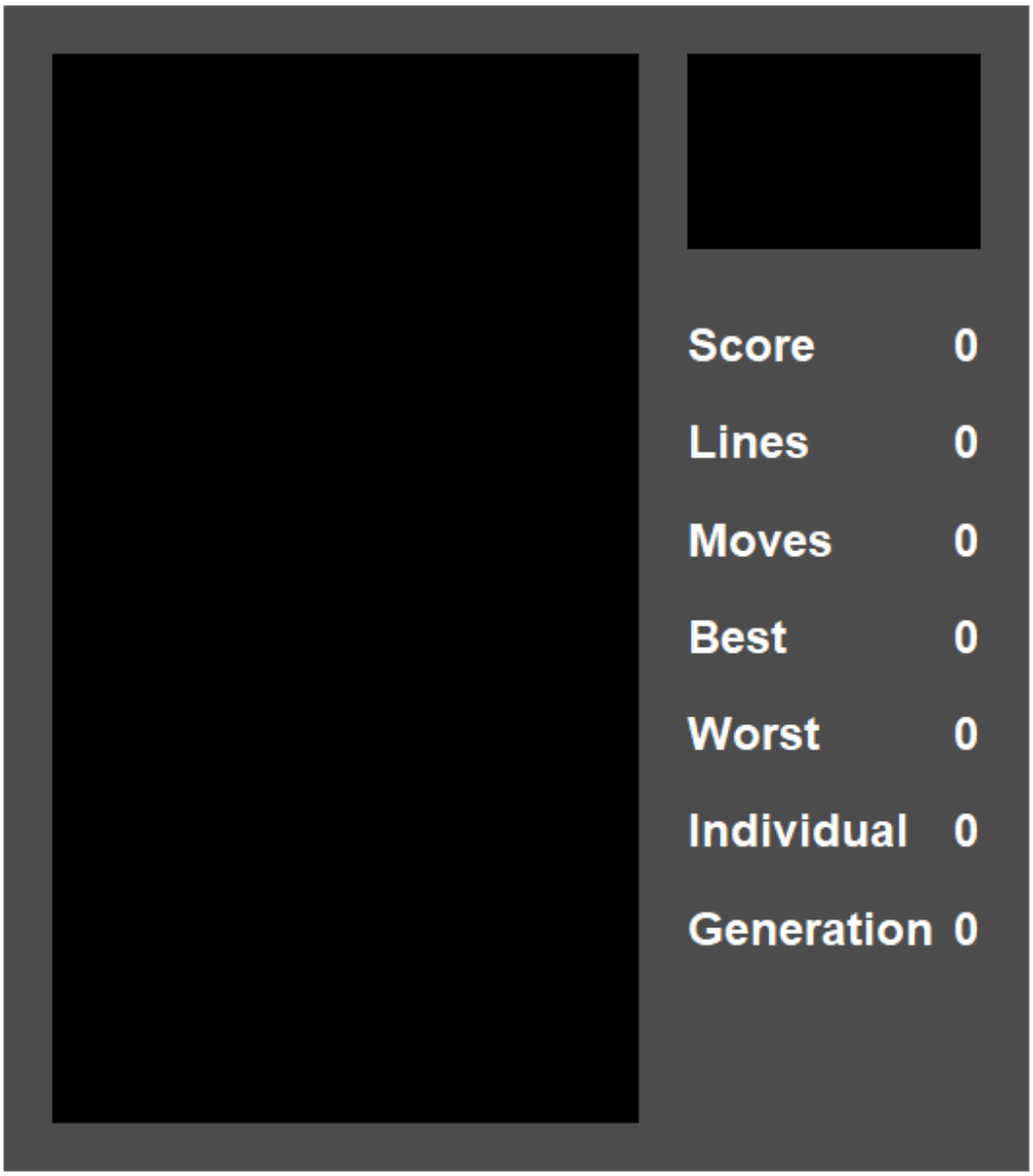

Рисунок 4 – Графический интерфейс тетриса

Значения «Best», «Worst», «Individual» и «Generation» относятся к процессу обучения алгоритма и будут описаны в следующем разделе.

2. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

2.1 Инициализация

Переменная $populationSize$ содержит количество индивидов в поколении, в данном случае обучение проходило при $populationSize = 50$. Переменная $generations$ содержит число поколений, отпущенных на эволюцию. Количество наиболее приспособленных индивидов, переходящих в следующее поколение, хранится в переменной $elites$, при этом разница между $populationSize$ и $elites$ должны быть четной. Переменная $movesLimit$ содержит максимальное количество ходов в партии, при достижении которого партия завершается. Шанс и шаг мутации соответственно хранятся в переменных $mutationRate$, $mutationStep$, в данном случае обучение проходило при $mutationRate = 0.02$, $mutationStep = 0.15$. Все промежуточные результаты хранятся в ассоциативном массиве $archive$, содержащем пять ключей:

1) «size» – количество прошедших поколений;
2) «totalFitness» – список, хранящий сумму оценок приспособленностей всех индивидов каждого прошедшего поколения;
3) «mostFintess» – список, хранящий оценки наиболее приспособленного индивида каждого прошедшего поколения;
4) «elites» – список, хранящий количество равное $elites$ наиболее приспособленных индивидов каждого прошедшего поколения;
5) «generations» – список всех индивидов каждого прошедшего поколения.

2.2 Генерация начальной популяции

Каждый индивид представлен ассоциативным списком, содержащим семь ключей, первые шесть из которых – гены, а седьмой – оценка приспособленности:

1) «rowsCleared» – вес количества строк, удаленных после хода;
2) «weightedHeight» – вес высоты самого высокого столбца;
3) «cumulativeHeight» – вес суммы высот всех столбцов;

4) «relativeHeight» – вес разницы между высотами самого высокого столбца и самого низкого столбца;
5) «holes» – вес количества заблокированных фигурой сверху клеток, образованных после хода;
6) «roughness» – вес суммы разниц высот каждого столбца с высотами всех следующих столбцов;
7) «fitness» – оценка приспособленности.

Эти и другие параметры описаны в статье «An Evolutionary Approach to Tetris» [1].

Для генерации используется функция $initPopulation[]$, которая в цикле создает $populationSize$ индивидов со случайными весами от -0.5 до 0.5 и добавляет их в список $genomes$, содержащий индивидов текущего поколения. Также увеличивается на один значение $archive["size"]$.

2.3 Обучение

Основной функцией, производящей обучение является $evolve[]$. Переменная $generation$ хранит номер текущего поколения, и ее значение отображается функцией $Tetris$, также отображается сообщение «Training», пока номер текущего поколения меньше значения $generations$, продолжается обучение.

Для каждого индивида текущего поколения вычисляется оценка приспособленности, для этого три раза выполняется функция $fitness$, после чего берется среднее значение. Номер индивида отображаются функцией $Tetris$.

Список индивидов сортируется по оценки приспособленности, наихудшая и наилучшая оценки отображаются функцией $Tetris$. Ассоциативный список $archive$ обновляется исходя из списка $genomes$.

Создается список $children$, который хранит указанное в переменной $elites$ количество наиболее приспособленных особей. После чего дополняется $populationSize - elites$ потомками, сгенерированными с помощью функции

$makeChildren$. В качестве родителей выбираются случайные индивиды из списка $genomes$, выбор происходит с учетом весов, в их роли выступают оценки приспособленности. Поскольку функция $makeChildren$ возвращает двух потомков, разница $populationSize - elites$ должна быть четной. После генерации потомков список индивидов текущего поколения заменяется на список потомков.

По завершении обучения сообщение «Training» меняется на «Complete».

3. ОЦЕНКА ПРИСПОСОБЛЕННОСТИ

3.1 Основная функция

Основной процесс оценки приспособленности происходит в функции $fitness$. На вход функция принимает два аргумента: индивид и пауза между размещением фигур на игровом поле. В качестве результата возвращается финальный счет.

Сперва происходит обновление переменных вызовом функции $init$, переменной $play$, которая означает, завершена партия или нет, присваивается True. Также обнуляется значение совершенных ходов. Пока количество совершенных ходов меньше ограничения и партия не завершена, переменным fig, fc, содержащие координаты текущей фигуры и ее цвет соответственно, присваивается первый элемент списка $fig7$, содержащий случайную последовательность фигур, после чего из списка $fig7$ удаляется первый элемент. Если список $fig7$ пустой, то ему присваивается новая случайная последовательность фигур. Далее происходит генерация всех возможных расположений фигуры fig на игровом поле посредством функции $getAllPossibleMoves$, результат сохраняется в переменной pm, после чего переменной mrm присваивается наиболее удачный ход, найденный с помощью функции $mostRatedMove$. Переменным y и x присваиваются координаты размещения фигуры, значению fig присваивается текущая фигура с некоторым поворотом. Если размещение фигуры не означает поражение, то fig размещается на игровом поле, удаляются заполненные строки и увеличивается счет, генерируется новая маска и увеличивается на один количество совершенных ходов, иначе партия заканчивается.

Размещение фигуры в матрице $glass$ происходит с помощью функции put, которая присваивает элементам матрицы на позициях, равных сумме соответсвенных координат фигуры fig и y, x, fc.

Удаление заполненных строк выполняется с помощью функции del, в которой содержатся три локальные переменные: sel – список из логических

переменных, означающих заполнена ли строка или нет; g – матрица glass, из которой были удалены заполненные строки; ln – количество удаленных строк. Если были удалены строки, то матрице $glass$ присваивается матрица g, дополненная в конце строками, количество которых равно количеству удаленных строк. После чего увеличивается значение переменной $lines$ на количество удаленных строк, счета и уровня. В конце генерируется новая маска.

3.2 Нахождение всех возможных ходов

Поиск всех возможных расположений фигуры происходит с помощью функции $getAllPossibleMoves$. Создается пустой список pm, который будет результатом работы функции. Переменной fg присваивается fig. Создается матрица $gl = glass$. Начальные координаты задаются следующим образом: $y1 = h - 3,\ x1 = Floor[w/2]$.

Основная работа функции заключается в цикле из трех итераций, в котором содержатся два вложенных цикла, в одном из которых уменьшается $x1$ на единицу, пока это возможно, а во втором увеличивается. В каждом вложенном цикле также расположен вложенный цикл, в котором $y1$ уменьшается на единицу, пока это возможно. Проверка на то, являются ли требуемые клетки свободными, выполняется функцией $check$, которая увеличивает соответственные координаты фигуры на $y1$ и $x1$, после чего проверяет, все ли элементы вызова функции get, которая возвращает список элементов маски на позициях равных сумме соответсвенных координат фигуры и $y1$, $x1$, являются логическими переменными True. Если y1 нельзя уменьшить, то в список pm добавляется список из трех элементов: матрица gl с размещенной фигурой fg, fg, $\{y1,\ x1\}$. В конце каждой итерации происходит поворот фигуры на 90º функцией $rotate$, которая умножает координаты фигуры fg на матрицу поворота.

3.3 Выбор наиболее удачного хода

Функция $mostRatedMove$ принимает два аргумента: список всех возможных ходов и индивида. Создаются локальные переменные: $rate = 0$ – наивысший рейтинг, res – ход с наивысшим рейтингом, $temp$ – рейтинг текущего хода, m – маска матрицы gl текущего хода, sel – список логических значений: True, если строка заполнена, иначе False. Основная работа происходит в цикле, изображенном на рисунке 5, который перебирает все ходы, и для них вычисляется temp.

```
getMostRatedMove[moves_, genome_] := (Module[
   {rate = 0, res, temp, sel, m},
   Do[
    m = Map[# == bg &, move[[1]], {2}];
    sel = Not[Or @@ #] & /@ m;
    sel[[1]] = False;
    temp = 0;
    temp += Total[Boole@sel] * genome["rowsCleared"];
    temp += getHoles[m] * genome["holes"];
    temp += getHeight[m] * genome["weightedHeight"];
    temp += getRelativeHeight[m] * genome["relativeHeight"];
    temp += getCumulativeHeight[m] * genome["cumulativeHeight"];
    temp += getRoughness[m] * genome["roughness"];
    If[temp > rate ∨ rate == 0, rate = temp; res = move],
    {move, moves}];
   If[ListQ[res], {res, rate}, res = {glass, fig, {y, x}}];
   {res, rate}]
 )
```

Рисунок 5 – Нахождение наилучшего хода

Для каждого гена индивида вычисляется нужный параметр в маске m. Функция $Total[Boole@sel]$ возвращает количество заполненных строк. Функция $getHoles[m]$ возвращает количество заблокированных клеток фигурами сверху. Функция $getHeight[m]$ возвращает высоту самого высокого столбца. Функция $getRelativeHeight[m]$ возвращает разницу между высотой самого высокого столбца и высотой самого низкого столбца. Функция $getCumulative[m]$ возвращает сумму высот всех столбцов. Функция

$getRoughness[m]$ возвращает сумму разниц высот каждого столбца с высотами всех следующих столбцов.

3.4 Вспомогательные функции

Функция $getHoles$ принимает на вход маску, возвращает значение счетчика, который хранится в локальной переменной $count$. Функция проходит по каждому элементу матрицы, если элемент равен $False$, то вычисляется высота данного столбца, и если номер строки, в которой расположен элемент, меньше высоты, то счетчик увеличивается.

Функция $getHeight$ принимает на вход маску, возвращает высоту самого высокого столбца, хранящуюся в локальной переменной $height$. Для нахождения высоты функция проходит по всем столбцам маски и находит номер строки последней занятой клетки.

Функция $getRelativeHeight$ принимает на вход маску, находит высоту самого низкого столбца, хранящуюся в локальной переменной $height$ и возвращает разницу вызова функции $getHeight$ и высоту самого низкого столбца. Для нахождения наименьшей высоты функция проходит по всем столбцам маски и находит номер строки первой занятой клетки.

Функция $getCumulativeHeight$ принимает на вход маску и возвращает сумму высот всех столбцов, хранящуюся в локальной переменной $total$. Для нахождения суммы функция проходит по всем столбцам маски и увеличивает $total$ на номер строки последней занятой клетки.

Функция $getRoughness$ принимает на вход маску и возвращает сумму разниц высот каждого столбца с высотами всех следующих столбцов, хранящуюся в локальной переменной dif. Для нахождения суммы функция проходит по всем столбцам маски, записывая в локальную переменную $temp$ высоту текущего столбца, после чего в цикле проходит по всем следующим столбцам, находит их высоту и увеличивает dif на модуль разницы между temp и высотой данного столбца.

4. ГЕНЕРАЦИЯ ПОТОМКОВ

Для генерации потомков используется функция $makeChildren$, на вход функция принимает два аргумента: первый родитель и второй. Создается новый индивид $child1$, гены которого наследуются случайным образом от одного из родителя, выбор происходит с учетом весов, в качестве которых выступают оценки приспособленности родителей. Создается индивид $child2$, который наследует гены, ненаследуемые первым потомком. После чего каждый ген потомков может мутировать с вероятностью $mutationRate$. При мутации значение гена умножается на число, выбранное случайным образом из промежутка $[1 - mutationRate; 1 + mutationRate]$. Мутация изображена на рисунке 6. Функция возвращает список из двух потомков.

```
If[RandomReal[] < mutationRate, child1["rowsCleared"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child1["weightedHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child1["cumulativeHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child1["relativeHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child1["holes"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child1["roughness"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];

If[RandomReal[] < mutationRate, child2["rowsCleared"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child2["weightedHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child2["cumulativeHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child2["relativeHeight"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child2["holes"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
If[RandomReal[] < mutationRate, child2["roughness"] *= RandomReal[{1 - mutationStep, 1 + mutationStep}]];
```

Рисунок 6 – Мутация потомков

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы цель задачи была выполнена в полном объеме. Посредством генетического алгоритма были найдены веса для параметров, которые позволяют алгоритму набирать в среднем одну тысячу игровых очков в игре тетрис. На эволюцию было отпущено пятьдесят два поколения.

График на рисунке 7 отображает изменение оценки приспособленности наилучшего и наихудшего индивида поколения, среднюю оценку приспособленности всех индивидов поколения. График на рисунке 8 отображает оценки приспособленности каждого индивида поколения. На основе данных графиков можно сделать выводы об успешности применения генетического алгоритма. Заметен рост средней и худшей оценки, также резкое улучшение лучшей и средней оценки в первых десяти поколениях.

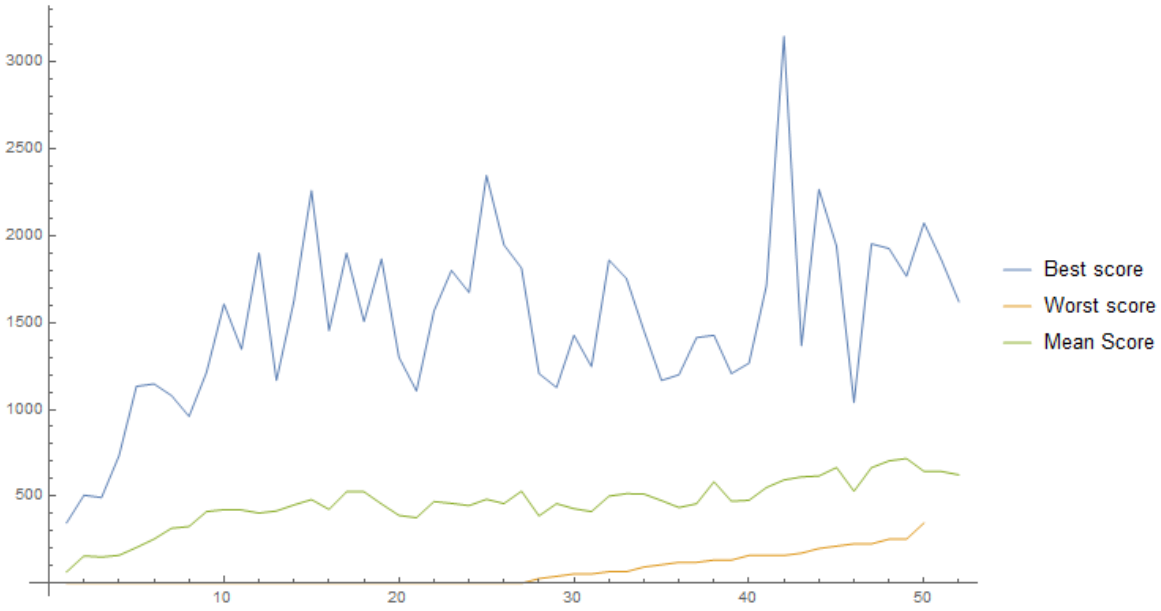

Рисунок 7 – Изменение оценки приспособленности

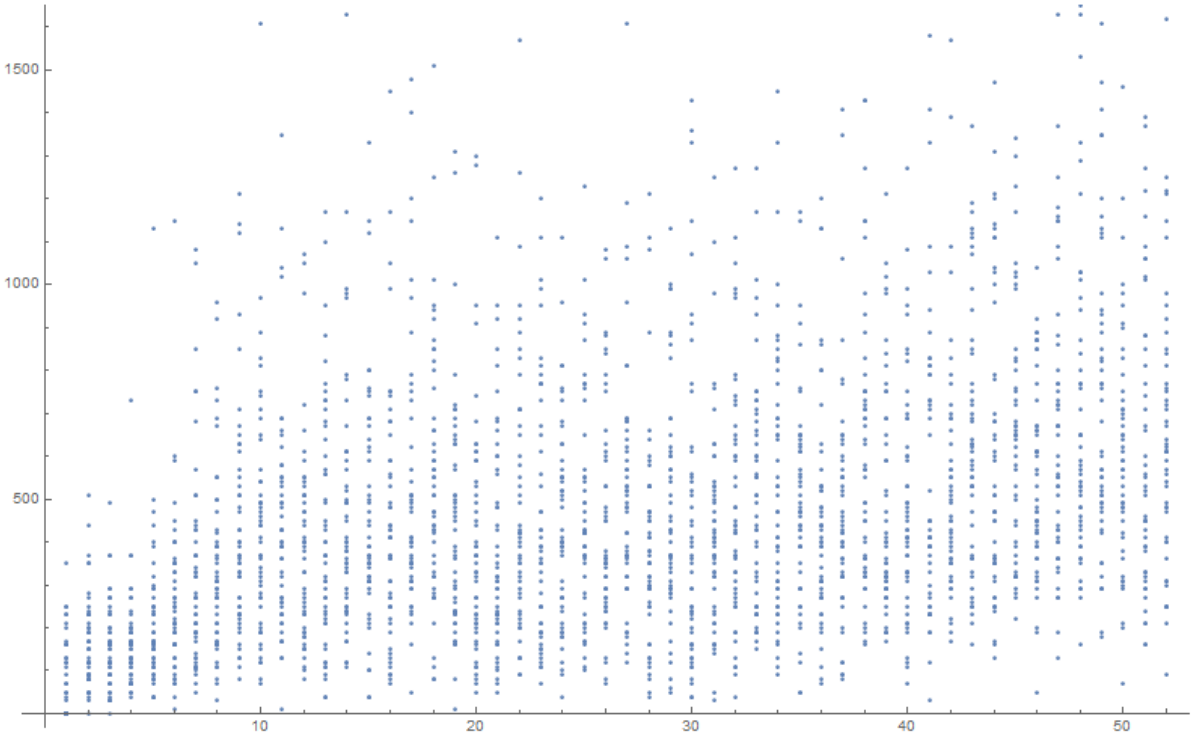

Рисунок 8 – Оценка приспособленности всех индивидов

Для улучшения результатов обучения возможно добавление большего количества параметров, изменение способа вычисления оценки приспособленности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Böhm N., An Evolutionary Approach to Tetris / Niko Böhm, Gabriella Kokai, Stefan Mandl / MIC2005: The Sixth Metaheuristics International Conference, Vienna, Austria, August, 22-26, 2005

2. Melanie M., An Introduction to Genetic Algorithms / Mitchell Milanie / First MIT Press paperback edition, 1998.

3. Wolfram Documentation [Электронный ресурс] / Documentation center. URL: https://reference.wolfram.com/language/ (дата обращения – 17.05.21).