Искусственный интеллект для игры “Гомоку”

2020

Гомоку

- Гомоку — настольная логическая игра для двух игроков. На квадратной доске размером 19×19 (в традиционном варианте) или 15×15 (в современном спортивном варианте) пунктов игроки поочерёдно выставляют камни двух цветов.
- Выигрывает тот, кто первым построит непрерывный ряд из пяти (и более) камней своего цвета по вертикали, горизонтали или диагонали

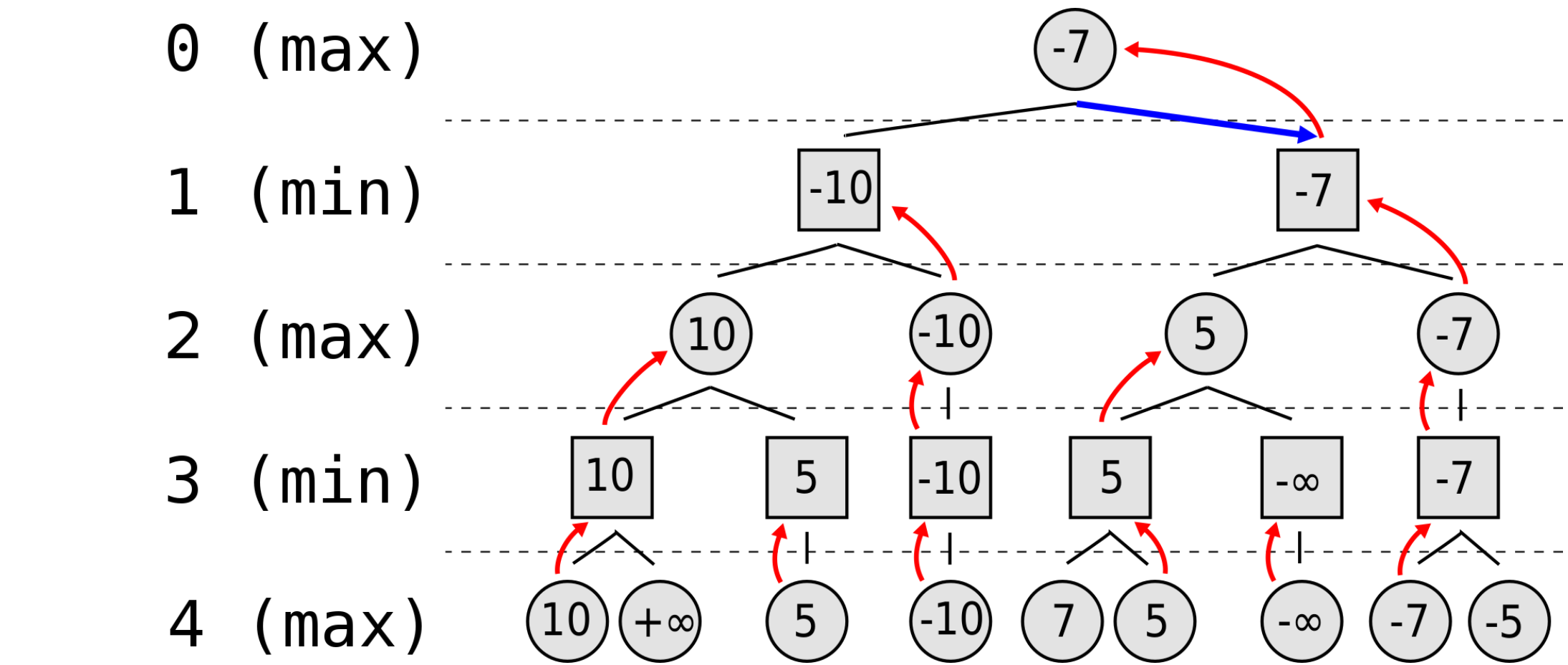
Сложности

- Большой коэффициент ветвления ~210
- 5-10 секунд на вычисления
- Невозможность проводить вычисления и поддерживать интерфейс отзывчивым в одном потоке

Minimax

- Минимакс — правило принятия решений, используемое в теории игр, теории принятия решений, исследовании операций, статистике и философии для минимизации возможных потерь из тех, которые лицу, принимающему решение, нельзя предотвратить при развитии событий по наихудшему для него сценарию
- Критерий минимакса первоначально был сформулирован в теории игр для игры двух лиц с нулевой суммой в случаях последовательных и одновременных ходов, впоследствии получил развитие в более сложных играх и при принятии решений в условиях неопределённости

Minimax



Evaluation function

- Функция оценки – оценивает текущее положение фигур на доске и возвращает число – на сколько хороша позиция для игрока.

Minimax

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

```
(* Initial call *)
minimax(origin, depth, TRUE)
```

Negamax

```
function negamax(node, depth, color) is
    if depth = 0 or node is a terminal node then
        return color × the heuristic value of node
    value := −∞
    for each child of node do
        value := max(value, −negamax(child, depth − 1, −color))
    return value
```

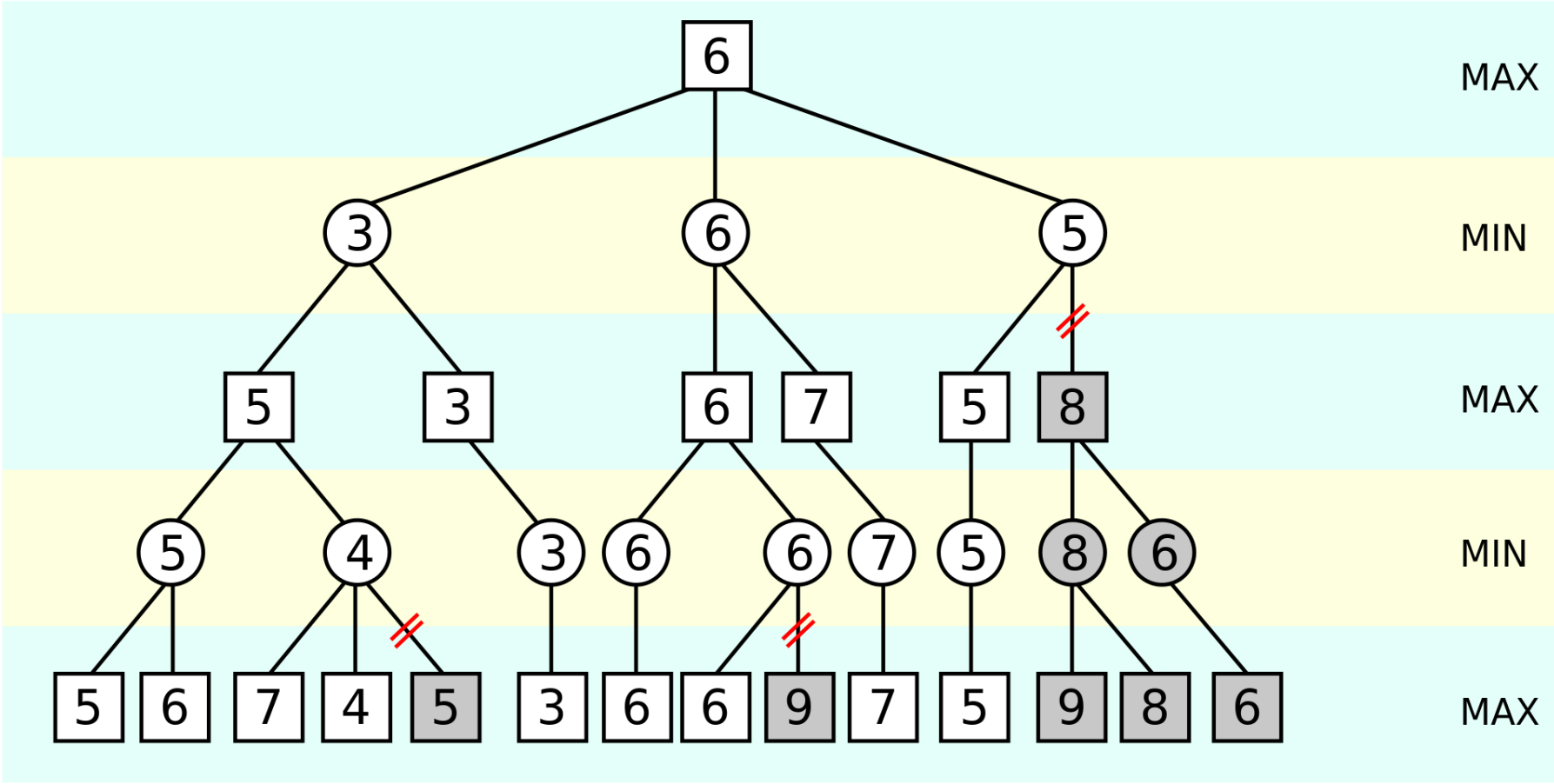
- Алгоритм Negamax идентичен Minimax, но чаще используется на практике ввиду его краткости и отсутствию лишнего if.
- Это возможно благодаря следующему равенству:

$$\max(a, b) = -\min(-a, -b)$$

Alpha–beta pruning

- **Альфа-бета-отсечение** — оптимизация минимакса, стремящаяся сократить количество узлов, оцениваемых в дереве поиска.
- В основе алгоритма лежит идея, что оценивание ветви дерева поиска может быть досрочно прекращено (без вычисления всех значений оценивающей функции), если было найдено, что для этой ветви значение оценивающей функции в любом случае хуже, чем вычисленное для предыдущей ветви.
- Альфа-бета-отсечение является оптимизацией, так как результаты работы оптимизируемого алгоритма не изменяются.

Alpha–beta pruning



Minimax with alpha beta pruning

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if α ≥ β then
                break (* α cut-off *)
        return value
```

```
(* Initial call *)
alphabeta(origin, depth, −∞, +∞, TRUE)
```

Negamax with alpha beta pruning

```
function negamax(node, depth, α, β, color) is
    if depth = 0 or node is a terminal node then
        return color × the heuristic value of node

    childNodes := generateMoves(node)
    childNodes := orderMoves(childNodes)
    value := −∞
    foreach child in childNodes do
        value := max(value, -negamax(child, depth − 1, -β, -α, -color))
        α := max(α, value)
        if α ≥ β then
            break (* cut-off *)
    return value
```

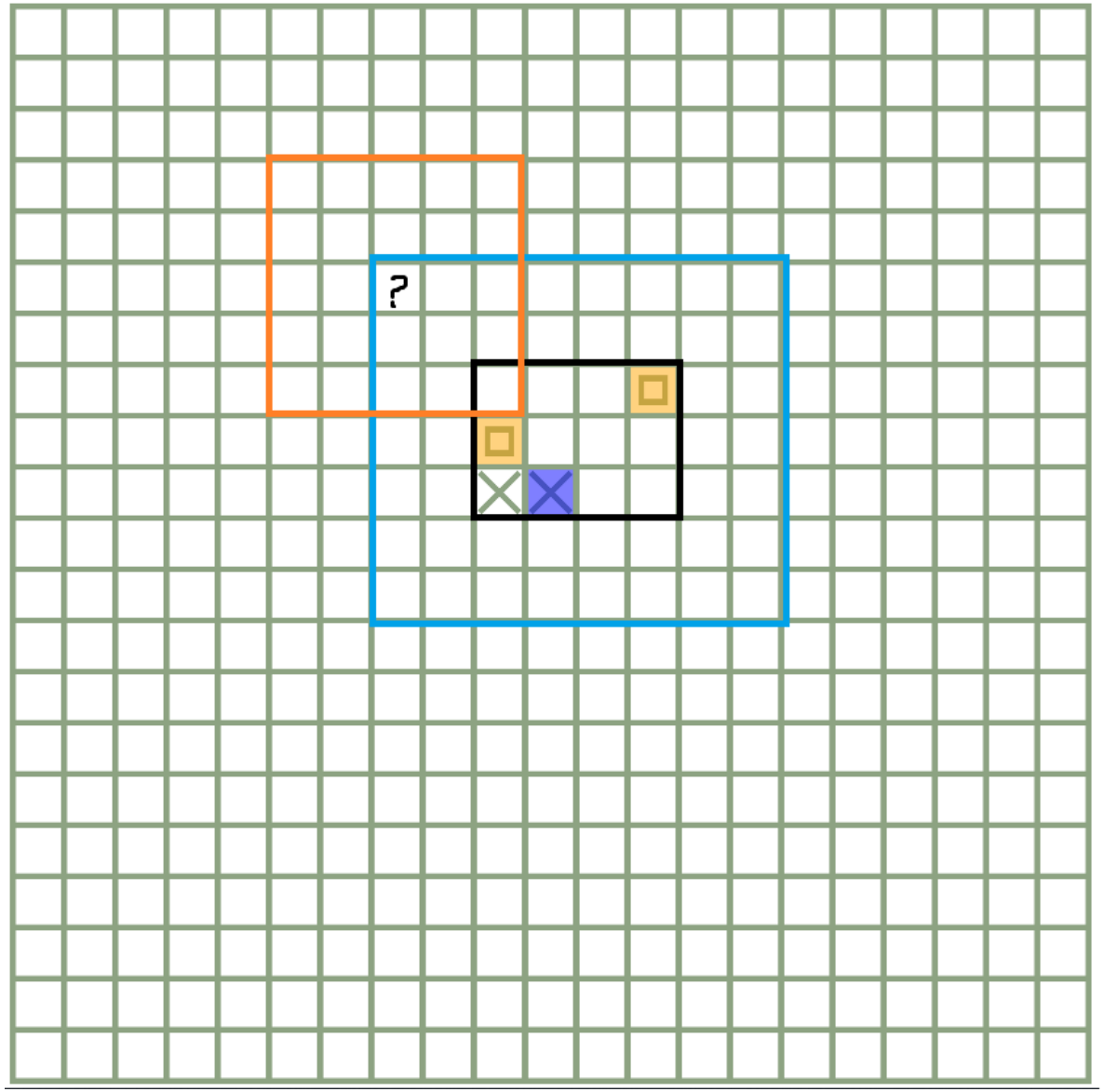
```
(* Initial call for Player A's root node *)
negamax(rootNode, depth, −∞, +∞, 1)
```

generateMoves

- С помощью функции оценки мы определяем насколько ход перспективен, присваиваем ему score. Затем сортируем ходы по score, так что наиболее перспективные ходы рассматриваются первыми.

```
function BoardGenerator(restrictions, Board, player) {
    let availSpots_score = []; //c is j  r is i;
    let min_r = restrictions[0];
    let min_c = restrictions[1];
    let max_r = restrictions[2];
    let max_c = restrictions[3];
    let move = {};
    for (var i = min_r - 2; i <= max_r + 2; i++) {
        for (var j = min_c - 2; j <= max_c + 2; j++) {
            if (Board[i][j] === 0 && !remoteCell(Board, i, j)) {
                move = {}
                move.i = i;
                move.j = j;
                move.score = evaluate_move(Board, i, j, player)
                if (move.score === WIN_DETECTED) {
                    return [move]
                }
                availSpots_score.push(move)
            }
        }
    }
    availSpots_score.sort(compare);
    //  return availSpots_score.slice(0,20)
    return availSpots_score;
}
```

- Рассматривать все возможные ходы слишком затратно, чтобы увеличить производительность, нужно отбросить некоторые неперспективные позиции.
- Алгоритм поиска перспективных позиций:
    1. Вписываем все фигуры на доске в минимально возможный прямоугольник (черный прямоугольник)
    2. Начинаем поиск в радиусе двух клеток от этого прямоугольник (синий прямоугольник)
    3. Если рассматриваемая клетка не имеет соседей в радиусе двух клеток, клетка считается неперспективной и отбрасывается (оранжевый квадрат)



Transposition table

- Таблица транспозиций- это кэш ранее виденных позиций и связанных с ними оценок в игровом дереве, созданном компьютерной игровой программой.
- Если позиция повторяется через другую последовательность ходов, извлекается полезная информация (a,b).

Zobrist hashing

- Для того чтобы получить хэш текущего состояния доски необходимо:
- Заранее сгенерировать таблицу, содержащую N больших случайных чисел,где N=Количество столбцов*Количество Строк*Количество игроков
- Применить операцию xor к элементам таблицы, соответствующим определенным позициям на доске
- Полученный хэш имеет свойство аддитивности – нет необходимости каждый раз вычислять хэш с нуля, достаточно лишь сложить по модулю два текущий хэш и значение из таблицы которое соответствует новой фигуре на доске

```
function Table_init() {
    for (var i = 0; i < Rows; i++) {
        Table[i] = [];
        for (var j = 0; j < Columns; j++) {
            Table[i][j] = []
            Table[i][j][0] = random32();
            Table[i][j][1] = random32();
        }
    }
}
```

```
function hash(board) {
    let h = 0;
    let p;
    for (var i = 0; i < Rows; i++) {
        for (var j = 0; j < Columns; j++) {
            let Board_value = board[i][j];
            if (Board_value !== 0) {
                if (Board_value === -1) {
                    p = 0
                } else {
                    p = 1
                }
                h = h ^ Table[i][j][p];
            }
        }
    }
    return h;
}
```

```
function update_hash(hash, player, row, col) {
    if (player === -1) {
        player = 0
    } else {
        player = 1
    }
    hash = hash ^ Table[row][col][player];
    return hash
}
```

Negamax with alpha beta pruning and transposition table

```
function negamax(node, depth, α, β, color) is
    alphaOrig := α

    (* Transposition Table Lookup; node is the lookup key for ttEntry *)
    ttEntry := transpositionTableLookup(node)
    if ttEntry is valid and ttEntry.depth ≥ depth then
        if ttEntry.flag = EXACT then
            return ttEntry.value
        else if ttEntry.flag = LOWERBOUND then
            α := max(α, ttEntry.value)
        else if ttEntry.flag = UPPERBOUND then
            β := min(β, ttEntry.value)

        if α ≥ β then
            return ttEntry.value

    if depth = 0 or node is a terminal node then
        return color × the heuristic value of node

    childNodes := generateMoves(node)
    childNodes := orderMoves(childNodes)
    value := −∞
    for each child in childNodes do
        value := max(value, −negamax(child, depth − 1, −β, −α, −color))
        α := max(α, value)
        if α ≥ β then
            break

    (* Transposition Table Store; node is the lookup key for ttEntry *)
    ttEntry.value := value
    if value ≤ alphaOrig then
        ttEntry.flag := UPPERBOUND
    else if value ≥ β then
        ttEntry.flag := LOWERBOUND
    else
        ttEntry.flag := EXACT
    ttEntry.depth := depth
    transpositionTableStore(node, ttEntry)

    return value
```

StateCache

- Для того чтобы увеличить производительность, следует избегать повторного вызова функции оценки.
- Так как одну и ту же конфигурацию доски можно получить разными путями, мы сохраняем значение функции оценки каждый раз при её вычислении и при достижении листьев (depth===0) проверяем наличие данных в кеше, если их нет, вызываем функцию оценки, иначе возвращаем значение из кеша

```
if (depth === 0) {
    if (StateCache[hash] !== undefined) {
        cch_hts++
        return StateCache[hash]
    }
    return evaluate_state(newBoard, player, hash, restrictions)
}
```

Поиск с нулевым окном

- Поиск с нулевым окном – это вызов алгоритма альфа бета со значениями alpha == beta – 1.
- Данная техника позволяет увеличить количество отсечек и соответственно скорость поиска

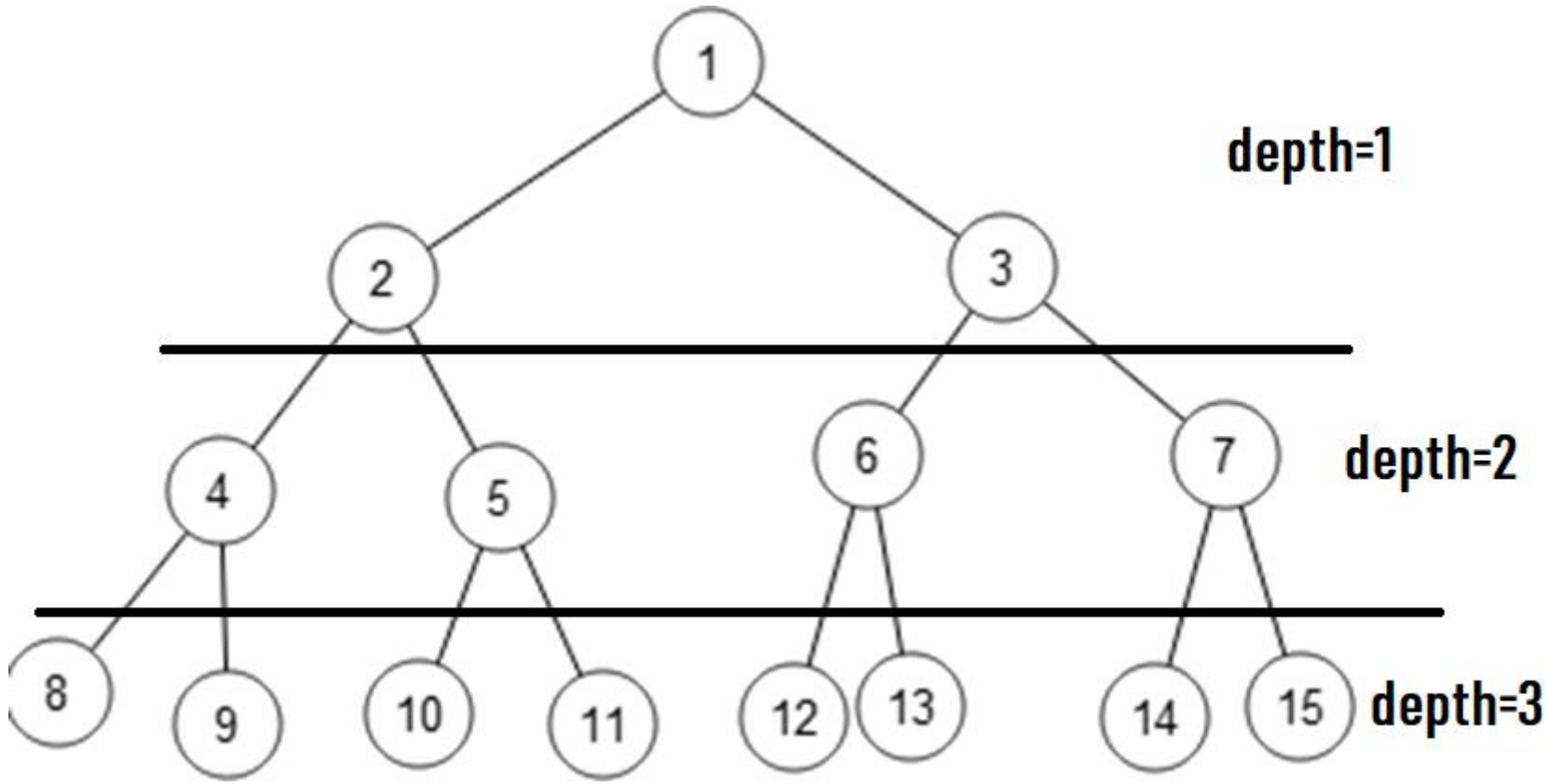
NegaScout

- NegaScout – оптимизация альфа-бета-отсечения, использует метод поиска с нулевым окном. Использовался в шахматном компьютере Deep Blue

```
int NegaScout ( position p; int alpha, beta )
{                       /* compute minimax value of position p */
   int b, t, i;
   if ( d == maxdepth )
      return quiesce(p, alpha, beta);           /* leaf node */
   determine successors p_1,...,p_w of p;
   b = beta;
   for ( i = 1; i <= w; i++ ) {
      t = -NegaScout ( p_i, -b, -alpha );
      if ( (t > a) && (t < beta) && (i > 1) )
         t = -NegaScout ( p_i, -beta, -alpha ); /* re-search */
      alpha = max( alpha, t );
      if ( alpha >= beta )
         return alpha;                              /* cut-off */
      b = alpha + 1;                    /* set new null window */
   }
   return alpha;
}
```

Iterative deepening depth-first search (IDDFS)

- Поиск с итеративным углублением — это оптимизация поиска в глубину и в ширину, которая гарантированно позволяет найти самое близкое к начальному состоянию решение, избегая экспоненциальной сложности.
- Каким образом реализуется этот алгоритм? Мы ищем в глубину с ограничением глубины константой N. Нашли решение – хорошо. Не нашли — повторяем поиск в глубину с константой N+1 и так далее, пока не отыщется.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
depth=1
depth=2
depth=3

MTD(f)

- MTD(f)-Memory-enhanced Test Driver with node *n* and value *f,* алгоритм использующий *только* поиск с нулевым окном для определения лучшего хода. В среднем он превосходит Negascout и Negamax.
- Чаще всего реализуется как поиск с итеративным углублением

MTD(f)

```
function MTDF(root : node_type; f : integer; d : integer) : integer;
      g := f;
      upperbound := +INFINITY;
      lowerbound := -INFINITY;
      repeat
            if g == lowerbound then beta := g + 1 else beta := g;
            g := AlphaBetaWithMemory(root, beta - 1, beta, d);
            if g < beta then upperbound := g else lowerbound := g;
      until lowerbound >= upperbound;
      return g;
```

```
function iterative_deepening(root : node_type) : integer;

      firstguess := 0;
      for d = 1 to MAX_SEARCH_DEPTH do
            firstguess := MTDF(root, firstguess, d);
            if times_up() then break;
      return firstguess;
```

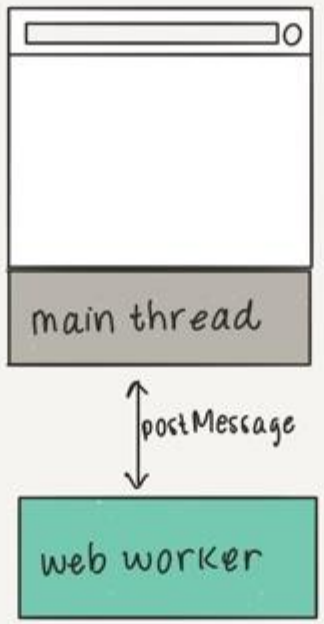
MTD(f) 10

- Для того чтобы увеличить производительность еще больше, можно рассматривать не все перспективные ходы, а только 10 самых перспективных из них.
- Коэффициент ветвления становится фиксированным (10) существенно увеличивая производительность алгоритма.
- Данная оптимизация рискованна тем, что опасный ход соперника может оказаться не рассмотренным, но на практике она существенно увеличивает скорость и процент побед.

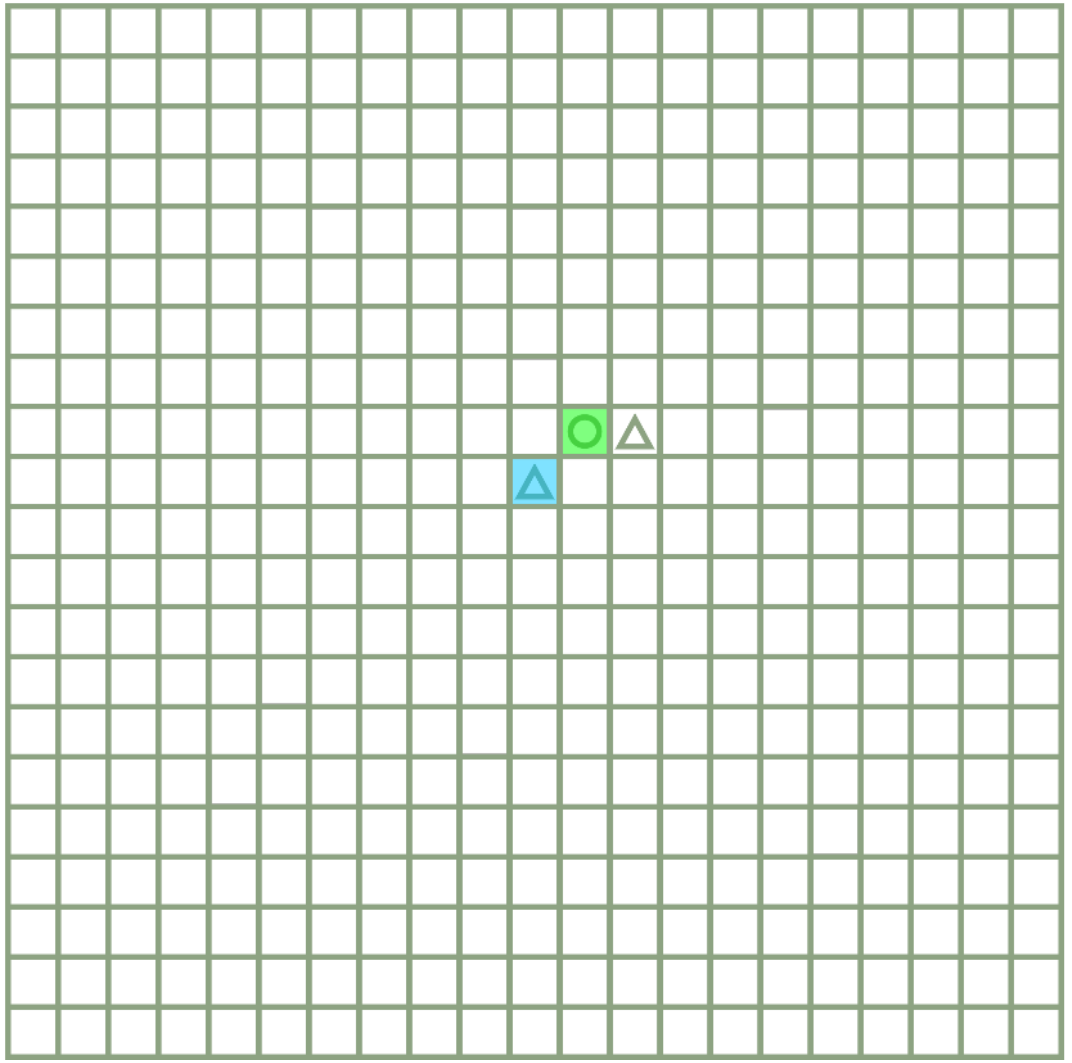
```
function BoardGenerator(restrictions, Board, player) {
    let availSpots_score = []; //c is j  r is i;
    let min_r = restrictions[0];
    let min_c = restrictions[1];
    let max_r = restrictions[2];
    let max_c = restrictions[3];
    let move = {};
    for (var i = min_r - 2; i <= max_r + 2; i++) {
        for (var j = min_c - 2; j <= max_c + 2; j++) {
            if (Board[i][j] === 0 && !remoteCell(Board, i, j)) {
                move = {}
                move.i = i;
                move.j = j;
                move.score = evalute_move(Board, i, j, player)
                if (move.score === WIN_DETECTED) {
                    return [move]
                }
                availSpots_score.push(move)
            }
        }
    }
    availSpots_score.sort(compare);
    return availSpots_score.slice(0,10)
    // return availSpots_score;
}
```

Web Workers API

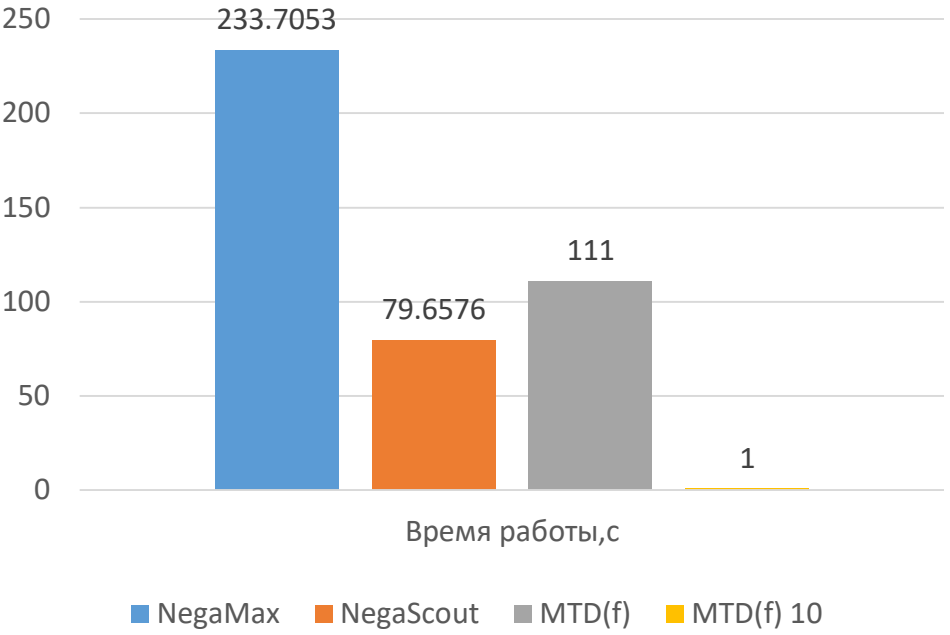
- Для того чтобы поддержать отзывчивой работу интерфейса при выполнении алгоритма, необходимо выполнять его в отдельном потоке.
- Web Workers API позволяет легко создать новый поток и обмениваться с ним данными.

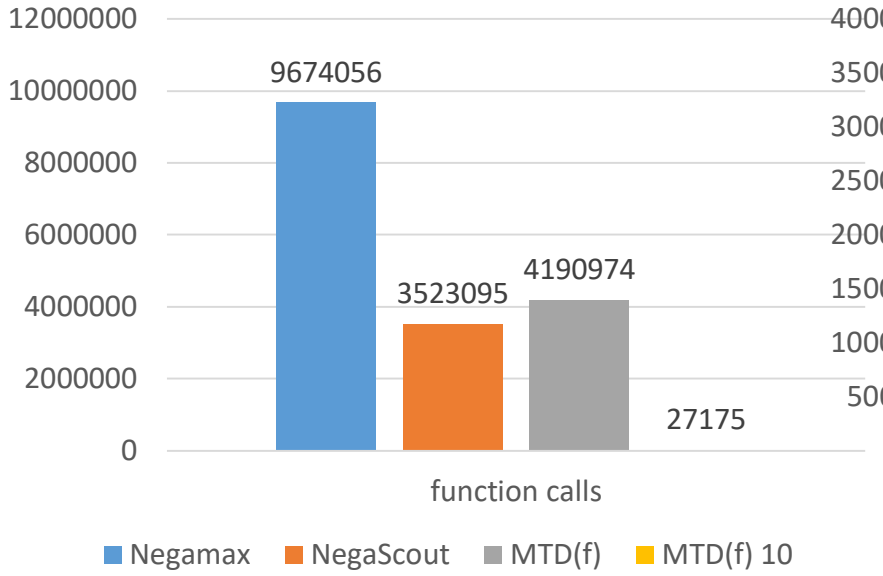


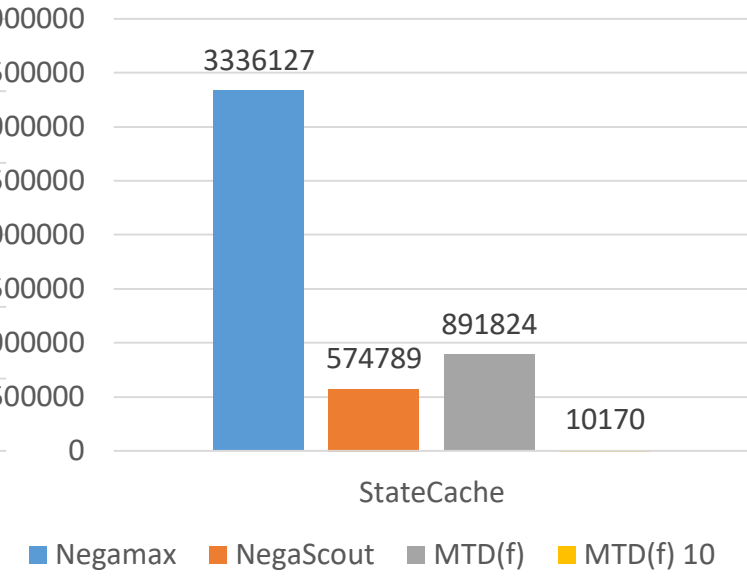
- Сравним производительность алгоритмов на примере поиска на глубину 8 из данной позиции



Тесты (depth=8)







Тесты (depth=8)

- NegaScout оказался быстрее, чем MTD(f) в данной позиции, но возможно это лишь неудачный пример
- Проведем турнир между всеми алгоритмами до трех побед.
- После каждой игры алгоритмы меняются сторонами
- Все алгоритмы реализованы как IDDFS
- 1 – победа, 0.5 – ничья

Турнир (bo3) (60 s)

	Negamax	NegaScout	MTD(f)	MTD(f) 10
Negamax		0:2	0:2	0:2
NegaScout	2:0		1:2	0:2
MTD(f)	2:0	2:1		0:2
MTD(f) 10	2:0	2:0	2:0	

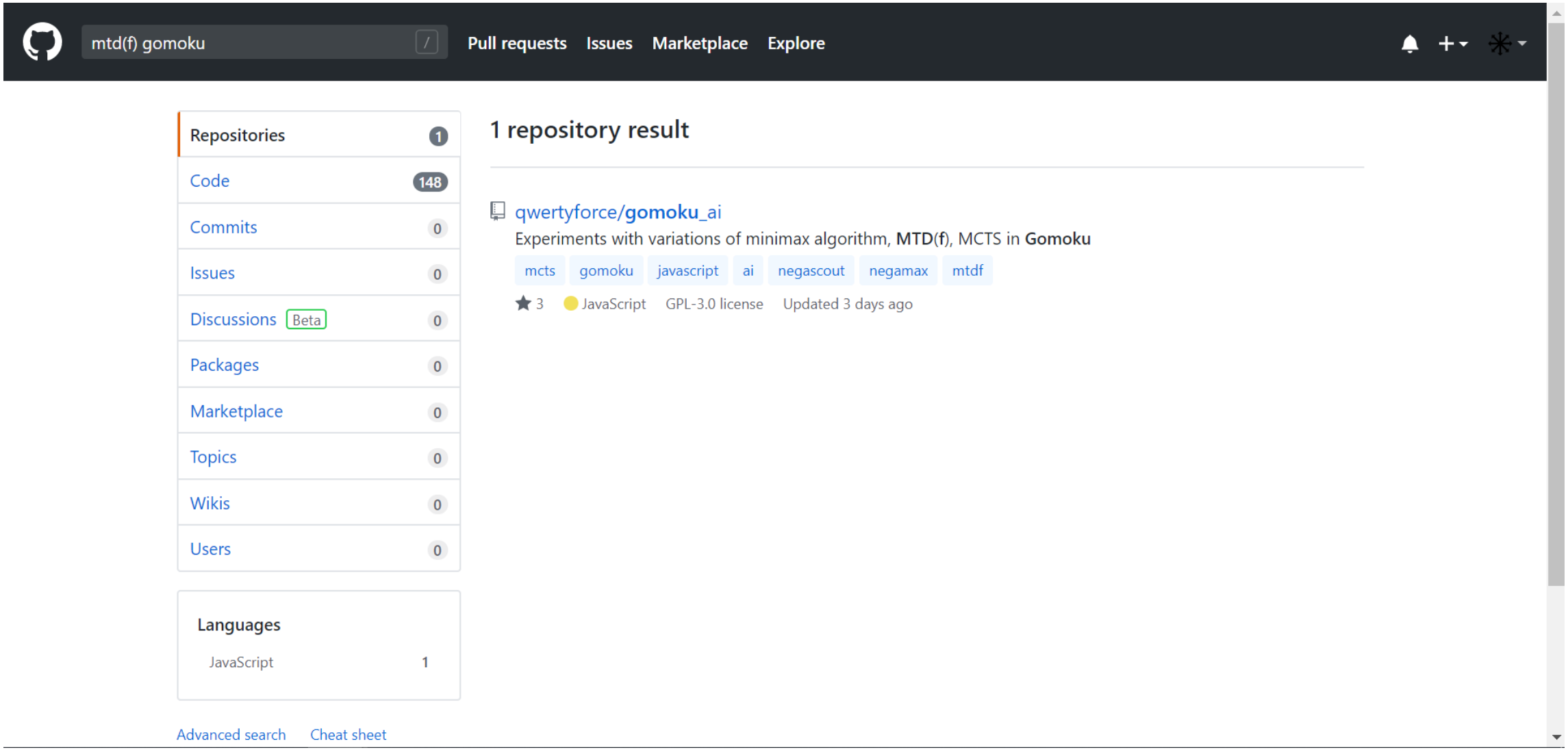
MTD(f) 10 1.5:0.5 NegaScout 10

Сравнение с другими алгоритмами на javascript

Сайт	Результат (MTD(f) 10)
http://gomoku.yjyao.com/	2:0 (10s на ход)
https://github.com/lihongxun945/gobang	2:0 (10s на ход)
https://logic-games.spb.ru/gomoku/	2:0 (10s на ход)
https://ixjamesyoo.github.io/Gomoku/	2:0 (10s на ход)

Выводы

- Получившийся алгоритм – MTD(f) 10, успешно справился со своими соперниками и благодаря повышенной производительности смог одержать победу во всех сыгранных матчах, не проиграв ни одной игры.
- Можно заметить, что если заменить функцию оценки, алгоритм определения победы и алгоритм генерации возможных ходов, данный алгоритм можно использовать для любой игры с полной информацией и нулевой суммой, где два игрока ходят поочередно.
- Данная работа является единственной реализацией алгоритма mtd(f) для игры гомоку на Github


mtd(f) gomoku
/
Pull requests
Issues
Marketplace
Explore
Repositories 1
Code 148
Commits 0
Issues 0
Discussions Beta 0
Packages 0
Marketplace 0
Topics 0
Wikis 0
Users 0
Languages
JavaScript 1
Advanced search
Cheat sheet
1 repository result
qwertyforce/gomoku_ai
Experiments with variations of minimax algorithm, MTD(f), MCTS in Gomoku
mcts
gomoku
javascript
ai
negascout
negamax
mtdf
3
JavaScript
GPL-3.0 license
Updated 3 days ago

mtd(f) gomoku

Sign in

All | Images | Shopping | Videos | News | More | Settings | Tools

About 64,100 results (0.29 seconds)

github.com › qwertyforce › gomoku_ai

qwertyforce/gomoku_ai: Experiments with variations ... - GitHub

gomoku_ai. Experiments with variations of minimax algorithm, **MTD**(**f**), MCTS in **Gomoku** (WIP). Main **mtd**(**f**) ai is in worker.js. You can play against ai here ...

github.com › mtrazzi › gomoku

mtrazzi/gomoku - GitHub

-p2 {human,minimax,alpha_beta,alpha_beta_memory,alpha_beta_basic,**mtdf**}, --player2 {human,minimax,alpha_beta,alpha_beta_memory,alpha_beta_basic ...

books.google.ru › books

Advances in Computer Games: 13th International Conference, ...

H. Jaap van den Herik, Aske Plaat - 2012 - Computers

... like Othello, **GoMoku**, and Amazons, can be described with the help of QIPs, ... 1MTD(**f**) or **MTD**(**f**,n): Memory-enhanced Test Driver with node n and value **f**.

www.gamedev.net › forums › topic › 415225-gomoku...

Gomoku Solution - Artificial Intelligence - GameDev.net

Список использованной литературы

- Aske Plaat: A Minimax Algorithm faster than NegaScout, [Электронный ресурс] https://arxiv.org/abs/1404.1511
- Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (3rd ed.). Upper Saddle River, New Jersey: Pearson Education, Inc. p. 167. ISBN 978-0-13-604259-4.
- [Электронный ресурс] https://www.chessprogramming.org/

Код

- https://github.com/qwertyforce/gomoku_ai

Сыграть

- https://4battle.ru/play_offline

Спасибо за внимание!