

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

«Альфа-бета отсечение в игре Щёлк»

(наименование темы проекта или работы)

ОТЧЁТ

по второму этапу курсовой работы

по дисциплине

технологии программирования

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

(подпись)

Капранов С.Н.

(фамилия, и.,о.)

СТУДЕНТ:

(подпись)

Куликова Е.А.

(фамилия, и.,о.)

18-ИСТ-4

(шифр группы)

Работа защищена «__» _____

С оценкой _____

Нижний Новгород 2020

Вариант 23

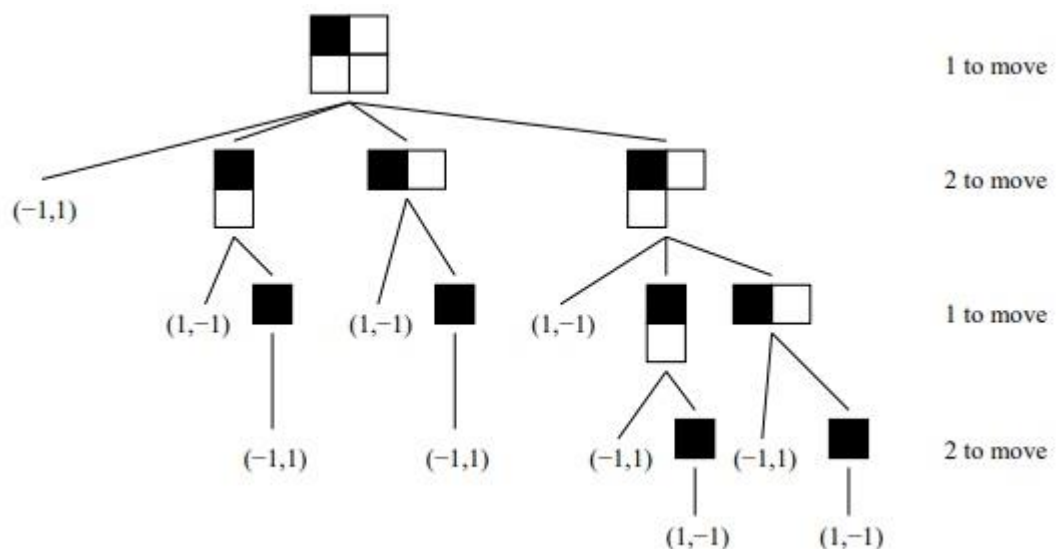
Игра «Щёлк»

Двое по очереди "откусывают" от прямоугольной доски. Игрок выбирает любое поле доски и снимает все фишки, которые находятся не выше и не левее избранного поля. Снявший последнюю фишку - проигрывает.

Начало игры	Первый игрок	Второй игрок	Первый игрок	Второй игрок
● ● ● ● ●	● ● ● ● ●	● ● ● ● ●	● ○ ○ ○ ○	●
● ● ● ● ●	● ● ● ● ○	● ● ● ● ●	● ○ ○ ○ ○	○
● ● ● ● ●	● ● ● ● ○	● ○ ○ ○ ○	●	○

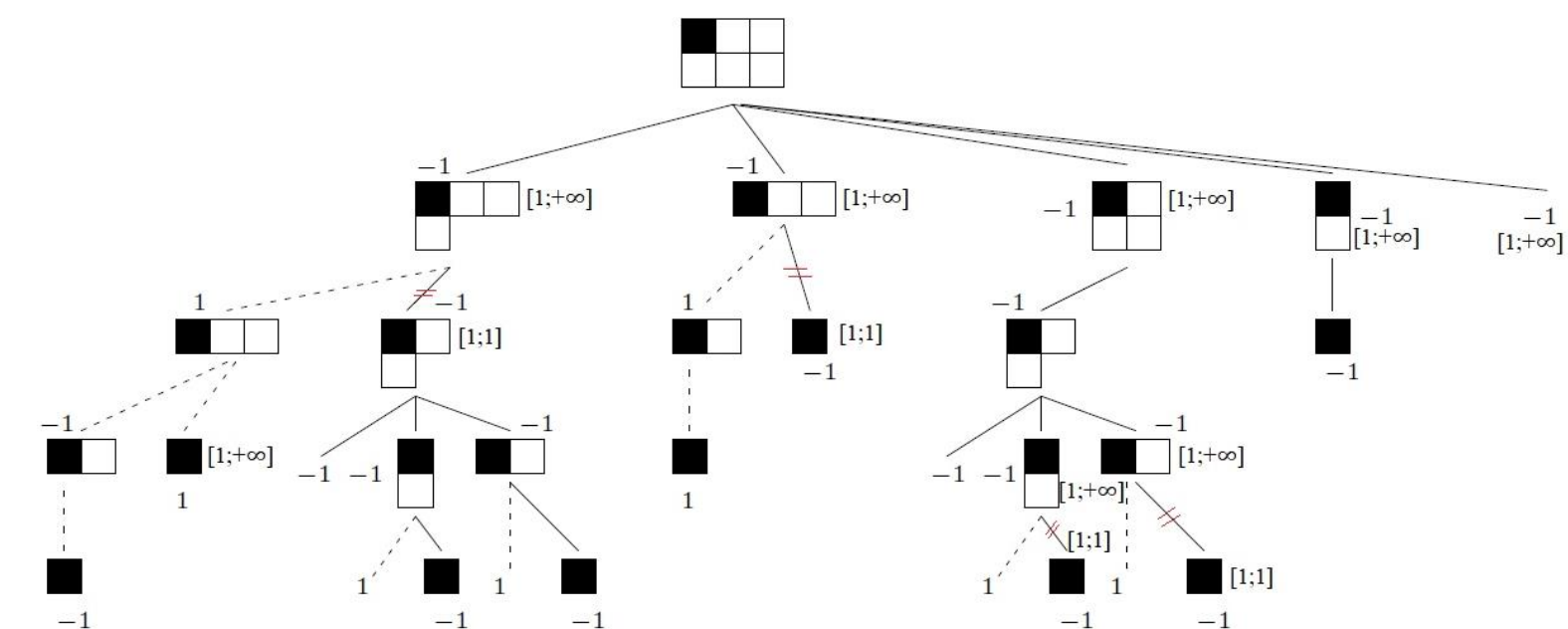
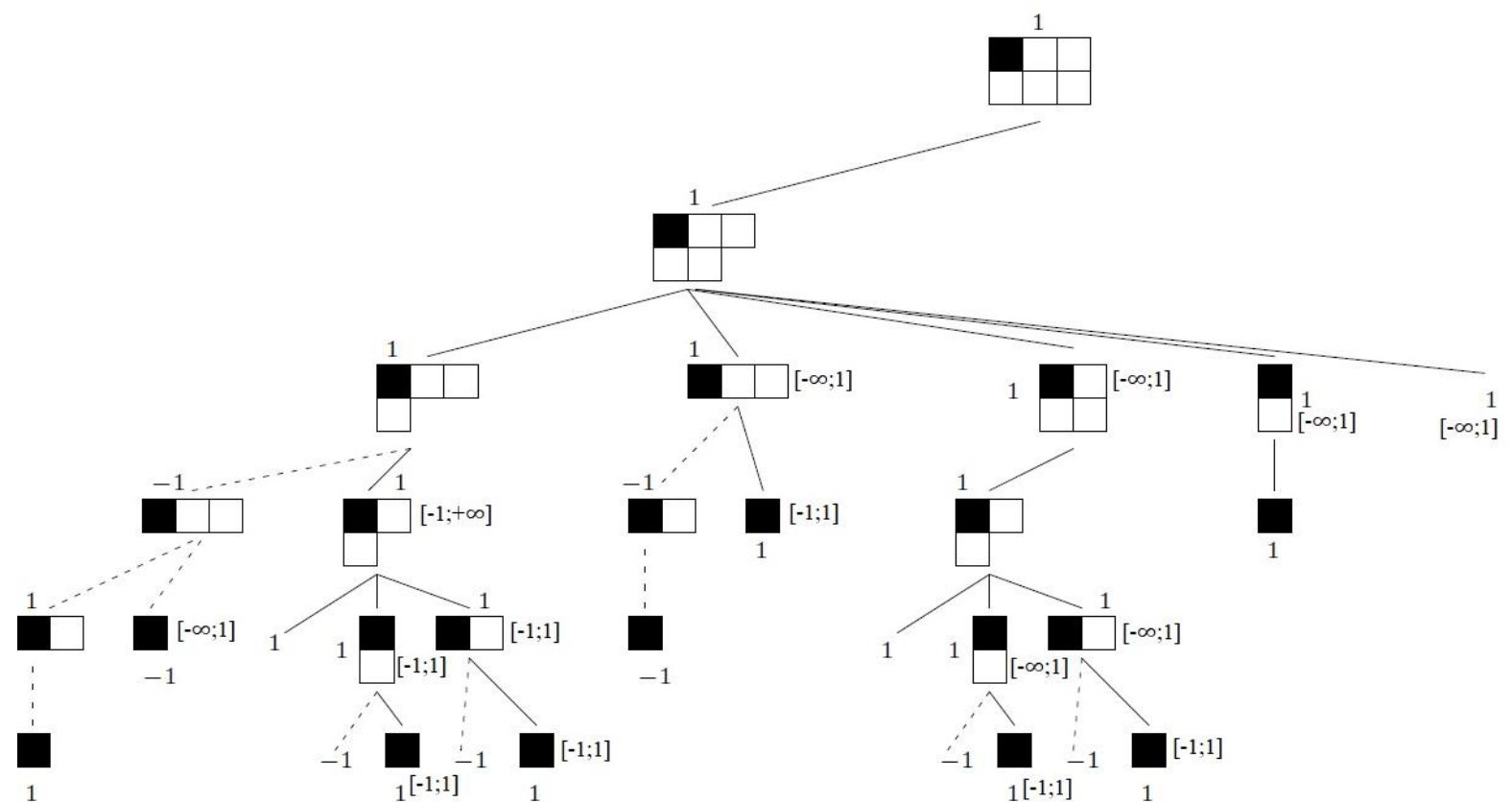
Алгоритм

Оценка игрового поля производится для «крайнего случая» рекурсии, т.е. когда осталась только «отравленная» фишка, с помощью оценочной функции, и для узлов дерева игры по стратегии *minimax*. Для игрока-*max*, если он победил, то оценка 1, если проиграл, то $-\infty$. Для игрока-*min* при победе оценка -1, при проигрыше $+\infty$.



Альфа-бета отсечение. Основная его идея заключается в том, чтобы после «прохода» по одной из ветвей дерева решений «отсекать» ветви, заведомо не имеющие оптимального решения, относительно коэффициентов альфа и бета, полученных на первом проходе. Альфа — это минимально возможная оценка, которую может получить максимизирующий игрок, инициализируется значением минус бесконечности и наоборот для беты. Если

в одном из узлов ветви значения альфа больше либо равно бета, найден ход, который гарантирует победу максимизирующему игроку. Если значения этих параметров в текущем узле не улучшаются, ветвь не следует рассматривать, поскольку все ее «потомки», как и она сама, не содержат оптимального решения.



Листинг

```
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <ctime>

int w, h;

// Вызов функции вернет доступные ходы для конкретного игрового поля
std::vector<std::pair<int, int>> available_moves(bool ** board)
{
    std::vector<std::pair<int, int>> moves;
    for (int i = 0; i < w; i++)
        for (int j = 0; j < h; j++)
            if (board[i][j] && (i != 0 || j != 0))
                moves.push_back(std::make_pair(i, j));
    return moves;
}

// Вызов данной функции вернет true если победил игрок, иначе false
bool has_won(bool ** board, bool is_maximizing)
{
    bool sum = false;
    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            if (i == 0 && j == 0)
                continue;
            sum = sum || board[i][j];
        }
    }
    if (!sum && board[0][0] && !is_maximizing)
        return true;
    return false;
}

// Данная функция возвращает true если один из игроков победил
bool game_is_over(bool ** board, bool is_maximizing)
{
    return has_won(board, is_maximizing);
}

// Данная функция возвращает оценку при достижении "крайнего случая"
int evaluate_board(bool ** board, bool is_maximizing)
{
    if (has_won(board, is_maximizing))
        return 1;
    return -1;
}

// Создает копию доски
bool ** deepcopy(bool ** board)
{
    bool ** res = new bool*[w];
    for (int i = 0; i < w; i++)
    {
        res[i] = new bool[h];
        for (int j = 0; j < h; j++)
            res[i][j] = board[i][j];
    }
    return res;
}
```

```

}

// Вызов функции означает выполнение хода игроком на конкретной игровой доске
bool select_space(bool ** board, std::pair<int, int> move)
{
    if (move.first >= w || move.second >= h)
        return false;
    if (board[move.first][move.second])
    {
        for (int i = move.first; i < w; i++)
            for (int j = move.second; j < h; j++)
                board[i][j] = false;
        return true;
    }
    return false;
}

std::pair<int, std::pair<int, int>> minimax(bool ** input_board, int alpha, int beta,
bool is_maximizing)
{
    // Крайний случай рекурсии - игра окончена
    if (game_is_over(input_board, is_maximizing))
        return std::make_pair(evaluate_board(input_board, is_maximizing),
std::pair<int, int>());
    // Инициализируем значения best_value и best_move
    std::pair<int, int> best_move;
    int best_value;
    // Случай, когда ход максимизирующего игрока
    if (is_maximizing)
        best_value = alpha;
    // Случай, когда ход минимизирующего компьютера
    else
        best_value = beta;
    /* Пройдём циклом по всем возможным ходам, для того чтобы выбрать наилучший
    путем рекурсивных вызовов функции minimax с копией игровой доски.
    Как только рекурсия достигнет "крайнего" случая, она вернет значения из [1, -1]
    для функции, которая ее вызвала, до тех пор, пока самая "верхняя" функция в стеке
    вызовов (minimax с текущей игровой доской) не получит свое значение */
    for (std::pair<int, int> move : available_moves(input_board))
    {
        bool ** new_board = deepcopy(input_board);
        select_space(new_board, move);
        int hypothetical_value;
        if (is_maximizing)
        {
            hypothetical_value = minimax(new_board, best_value, beta,
!is_maximizing).first;
            if (hypothetical_value > best_value)
            {
                best_value = hypothetical_value;
                best_move = move;
            }
            if (best_value >= beta)
                return std::make_pair(best_value, best_move);
        }
        else
        {
            hypothetical_value = minimax(new_board, alpha, best_value,
!is_maximizing).first;
            if (hypothetical_value < best_value)
            {
                best_value = hypothetical_value;
                best_move = move;
            }
        }
    }
}

```

```

        if (best_value <= alpha)
            return std::make_pair(best_value, best_move);
    }
    for (int i = 0; i < w; i++)
        delete new_board[i];
    delete new_board;
}
return std::make_pair(best_value, best_move);
}

int main()
{
    setlocale(LC_ALL, "Russian");

    std::cout << "Размеры поля - ";
    std::cin >> h >> w;

    int start = clock();
    bool ** board = new bool*[w];
    for (int i = 0; i < w; i++)
    {
        board[i] = new bool[h];
        for (int j = 0; j < h; j++)
            board[i][j] = true;
    }
    std::pair<int, std::pair<int, int>> res = minimax(board,
    std::numeric_limits<int>::min(), std::numeric_limits<int>::max(), true);
    std::cout << "оценка " << res.first << " ход " << res.second.second << " " <<
    res.second.first << " время " << clock() - start;

    for (int i = 0; i < w; i++)
        delete board[i];
    delete board;

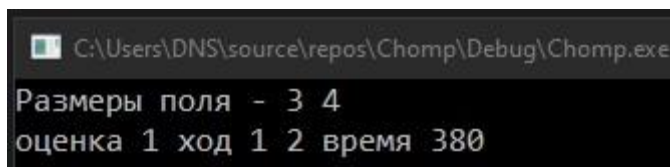
    return 0;
}

```

Результат выполнения программы

ход компьютера

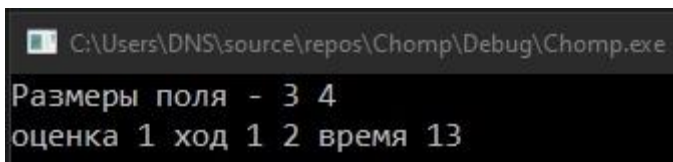
без альфа-бета отсечения и с ним



```

C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 3 4
оценка 1 ход 1 2 время 380

```



```

C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 3 4
оценка 1 ход 1 2 время 13

```

ход человека (0, 2)

ход компьютера

без альфа-бета отсечения и с ним

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 3 2
оценка 1 ход 2 1 время 1
```

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 3 2
оценка 1 ход 2 1 время 1
```

ход человека (2, 0)

ход компьютера

без альфа-бета отсечения и с ним

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 2 2
оценка 1 ход 1 1 время 0
```

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 2 2
оценка 1 ход 1 1 время 0
```

ход человека (1, 0)

ход компьютера

без альфа-бета отсечения и с ним

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 1 2
оценка 1 ход 0 1 время 0
```

```
C:\Users\DNS\source\repos\Chomp\Debug\Chomp.exe
Размеры поля - 1 2
оценка 1 ход 0 1 время 0
```

проигрыш человека

Как можно заметить при первом ходе компьютер затратил гораздо меньше времени на поиск нужного хода при помощи альфа-бета отсечения. При игре на поле большего размера эта разница становится колоссальной.