МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ СВЯЗИ

ЗАЩИЩЕНА С ОЦЕНКОЙ _____

РУКОВОДИТЕЛЬ
ассистент А. Н. Головенков
(должность, уч. степень, звание) (подпись, дата) (инициалы, фамилия)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

«ИГРА В КАЛЛАХ»

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ гр. № 2455 Моисеенко Я.Г.
(подпись, дата) (инициалы, фамилия)

Санкт-Петербург 2025

Содержание

1 ВВЕДЕНИЕ

Игры с полной информацией, такие как шахматы, шашки или го, долгое время служили полигоном для исследований в области искусственного интеллекта. Одной из старейших известных человечеству логических игр является «Манкала» и ее наиболее популярная вариация — «Каллах».

Несмотря на простые правила, Каллах обладает значительной тактической глубиной. Для компьютера реализация соперника в этой игре требует применения алгоритмов поиска решений на дереве игры. В отличие от простых переборных задач, здесь необходимо учитывать противодействие соперника, который также стремится максимизировать свой выигрыш.

Целью данной курсовой работы является разработка программного приложения, реализующего игру Каллах в режиме «Человек против Компьютера». Ключевой задачей является реализация алгоритма искусственного интеллекта, способного принимать эффективные решения за ограниченное время, для чего необходимо использование методов оптимизации перебора, таких как альфа-бета отсечение.

Актуальность работы заключается в изучении фундаментальных алгоритмов теории игр и программирования искусственного интеллекта, которые находят применение не только в игровой индустрии, но и в задачах принятия решений, экономики и планирования.

2 ПОСТАНОВКА ЗАДАЧИ

Задачей курсовой работы является разработка консольного приложения на языке высокого уровня (например, C++), которое позволяет пользователю играть в Каллах против алгоритмического соперника.

Требования к программе:

1. Реализация стандартных правил игры Каллах (6 лунок у каждого игрока, по 4 камня в каждой на старте, наличие двух накопителей — «каллахов»).
2. Поддержка двух режимов ходов: обычный ход и бонусный ход (если последний камень попадает в свой каллах).
3. Реализация механики захвата камней соперника.
4. Интерфейс программы должен быть текстовым. После каждого хода игрока или компьютера игровое поле должно отрисовываться заново с новой строки для наглядного отображения текущего состояния.
5. Реализация искусственного интеллекта на базе алгоритма Минимакс.
6. Внедрение алгоритма оптимизации (альфа-бета отсечение) для увеличения глубины просчета ходов.
7. Корректная обработка конца игры и подсчет очков.

Входные данные программы — команды пользователя (выбор лунки от 1 до 6). Выходные данные — графическое (символьное) отображение доски и сообщения о ходе игры.

3 АЛГОРИТМ

3.1 Описание алгоритма

Для реализации искусственного интеллекта в игре Каллах был выбран алгоритм **Минимакс** (Minimax) с модификацией **Альфа-бета отсечение** (Alpha-Beta pruning).

Алгоритм Минимакс используется в играх с нулевой суммой и полной информацией. Он строит дерево игровых состояний, где уровни чередуются между «максимизирующим» игроком (компьютер, который стремится набрать максимум очков) и «минимизирующим» игроком (человек, действия которого, как предполагается, ведут к минимизации выигрыша компьютера).

Поскольку полное дерево игры Каллах слишком велико для перебора до конца партии, используется ограничение глубины поиска. Для оценки позиций на предельной глубине применяется эвристическая функция.

Альфа-бета отсечение позволяет существенно сократить количество просматриваемых узлов дерева, отбрасывая ветви, которые заведомо не повлияют на итоговое решение (например, если найден ход, который хуже уже найденной альтернативы).

3.2 Описание шагов алгоритма

Процесс принятия решения компьютером состоит из следующих шагов:

1. **Генерация ходов:** Определяются все допустимые ходы для текущего игрока (все непустые лунки на его стороне поля).

2. **Симуляция хода:** Для каждого возможного хода создается копия игровой доски. На ней выполняется процесс «посева» камней согласно правилам игры, включая обработку бонусных ходов и захватов камней противника.

3. **Проверка условий остановки:** Если достигнута заданная глубина рекурсии или игра окончена (у одной из сторон закончились камни), вычисляется оценка позиции.

4. **Эвристическая оценка:** Оценка позиции вычисляется по формуле:

$$Score = (\text{Камни в каллахе Компьютера}) - (\text{Камни в каллахе Человека})$$

5. **Рекурсивный спуск:**

   - Если ход делает Компьютер (Max), из оценок дочерних узлов выбирается максимальное значение.
   - Если ход делает Человек (Min), выбирается минимальное значение.

6. **Отсечение (Pruning):**

   (a) В процессе перебора обновляются параметры Alpha (лучший гарантированный результат для Max) и Beta (лучший гарантированный результат для Min).

   (b) Если на каком-то этапе становится известно, что $Beta \leq Alpha$, дальнейший перебор текущей ветви прекращается, так как этот путь не будет выбран оптимальным игроком.

3.3 Псевдокод алгоритма

Ниже представлен псевдокод основной функции выбора хода.

Инициализация:

```
Set locale to "ru_RU.UTF-8"
Declare:
    in_file     // поток файла ввода
    out_file    // поток файла вывода

Backup original streams:
    cout_backup <- cout.rdbuf()
    cin_backup  <- cin.rdbuf()

tee_buffer <- null // будет хранить указатель к TeeBuf, если понадобится

// Обработка аргументов:
If argc >= 3 then
    // Полностью автоматическая игра: файл -> файл
    Open in_file using argv[1]
    If in_file is open then
        Redirect cin to in_file
    Open out_file using argv[2]
    If out_file is open then
        Redirect cout to out_file
    Print "Режим: Автоматический (файл -> файл)"
Else if argc = 2 then
    // Игра с логированием
    Open out_file using argv[1]
    If out_file is not open then
        Print error to cerr
        Return 1
    // Создание TeeBuf для дублирования вывода на экран и в файл
    tee_buffer <- new TeeBuf(cout_backup, out_file.rdbuf())
    Redirect cout to tee_buffer
    Print "Режим: Интерактивный с записью в " + argv[1]
Else
    // Простая игра (вывод только на экран)
    Print "Режим: Только экран (логи не пишутся)"
```

Цикл игры:

```
game <- new KalahGame()
isPlayerTurn <- true
repeatTurn <- false

Print "Добро пожаловать в игру Каллах!"
game.printBoard()

While NOT game.isGameOver() do
    If isPlayerTurn then
        Prompt "Ваш ход (введите номер лунки 1-6): "
        Loop
            If cin >> choice succeeds then
                choice <- choice - 1
```

```
                If choice in [0..5] AND game.getBoard()[choice] > 0 then
                    Break loop
                Else if choice not in [0..5] then
                    Print "Неверный номер (нужно 1-6): "
                Else
                    Print "Лунка пуста! Выберите другую: "
            Else
                If cin.eof() then
                    Go to EndGame
                Print "Введите число: "
                Clear cin error state
                Ignore remaining input line
        End Loop

        If argc >= 3 then
            // Копирование из файла ввода в файл вывода
            Print (choice + 1)

        repeatTurn <- game.makeMove(choice, true)
        Print "Вы выбрали лунку " + (choice + 1) + "."
    Else
        Print "Компьютер думает..."
        aiMove <- getBestMove(game)
        If aiMove = -1 then
            Break loop

        repeatTurn <- game.makeMove(aiMove, false)
        Print "Компьютер выбрал лунку " + aiMove + " (по индексу 7-12)."
    End If

    game.printBoard()

    If repeatTurn then
        If isPlayerTurn then
            Print "Бонусный ход! Снова вы."
        Else
            Print "Бонусный ход ПК."
    Else
        isPlayerTurn <- NOT isPlayerTurn
    End If
End While
```

Окончание игры и очистка:

```
Label EndGame:
game.finishGame()
game.printBoard()
game.announceWinner()

// Восстановить оригинальные потоки ввода/вывода
cin.rdbuf(cin_backup)
cout.rdbuf(cout_backup)

If tee_buffer != null then
```

```
    delete tee_buffer

// Пауза перед выходом если игра была интерактивная
If argc < 3 then
    Print "Нажмите Enter, чтобы выйти..."
    Wait for Enter key

Return 0
```

3.4 Блок-схема алгоритма

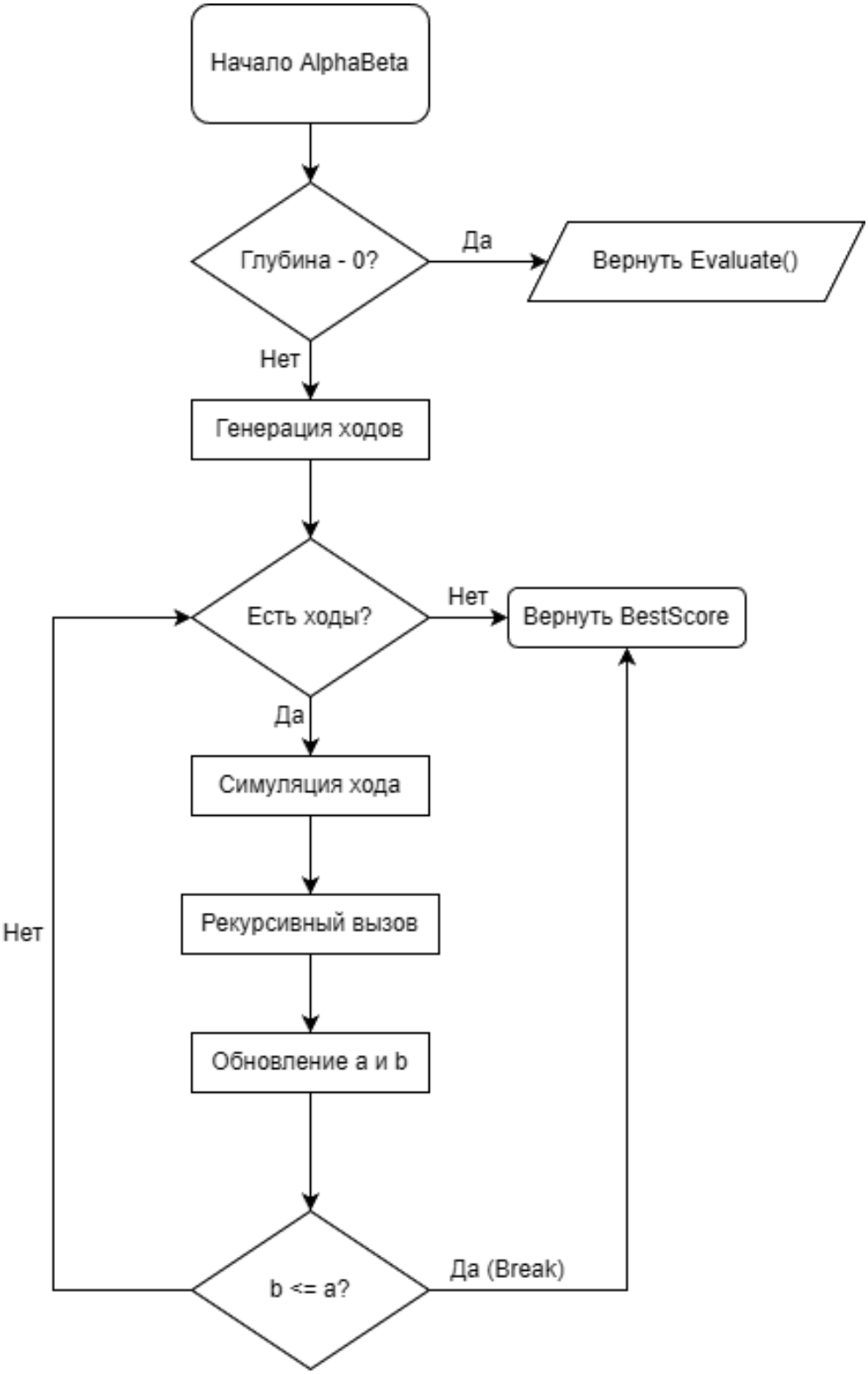

Рис. 1: Блок-схема алгоритма

3.5 Пример работы алгоритма по шагам

Рассмотрим упрощенный пример расчета на глубину 1 (анализ только ближайших последствий).

Ситуация: Ход компьютера. У него есть два возможных хода: из лунки №8 (4 камня) и лунки №12 (1 камень).

Таблица 1: Пример хода игры

Шаг	**Ход**	**Действие (Симуляция)**	**Оценка**
1	—	Инициализация. MaxEval = -Infinity.	—
2	№8	4 камня раскладываются в лунки 9, 10, 11, 12. Последний камень не в каллахе. Ход переходит к игроку.	—
3	—	Вычисление эвристики (разница камней в каллахах). Допустим, результат +2.	2
4	—	MaxEval обновляется: max(-inf, 2) = 2.	2
5	№12	1 камень кладется в каллах компьютера (№13).	—
6	—	Последний камень упал в каллах → Бонусный ход. Компьютер продолжает ходить.	—
7	—	Анализ продолжается (рекурсия не уменьшает глубину для бонусного хода). Допустим, итоговая оценка ветки +5.	5
8	—	MaxEval обновляется: max(2, 5) = 5.	5
9	—	Все ходы проверены. Выбирается ход №12.	—

3.6 Структуры данных

Для представления игрового поля в памяти программы используется массив (вектор) целых чисел.

1. **Тип данных:** `std::vector<int>`
2. **Размер массива:** 14 элементов.

Схема распределения индексов массива:

1. **Индексы 0–5:** Лунки Игрока (нижний ряд, слева направо).
2. **Индекс 6:** Каллах (накопитель) Игрока.
3. **Индексы 7–12:** Лунки Компьютера (верхний ряд, справа налево).
4. **Индекс 13:** Каллах (накопитель) Компьютера.

Такая структура позволяет легко реализовать круговое движение камней при «посеве» с помощью операции взятия остатка от деления:

```
index_next = (index_current + 1) % 14
```

3.7 Анализ сложности алгоритма

Эффективность программы зависит от временной и пространственной сложности используемого алгоритма поиска.

1. **Временная сложность:** В худшем случае (если отсечения не срабатывают) алгоритм Минимакс имеет сложность $O(b^d)$, где:

(a) b — коэффициент ветвления (количество доступных ходов, в Каллахе в среднем 4–6).

(b) d — глубина поиска.

При использовании альфа-бета отсечения в лучшем случае (при оптимальном порядке проверки ходов) сложность снижается до $O(b^{d/2})$. Это означает, что за то же время компьютер может просчитать ситуацию почти в два раза глубже.

2. **Пространственная сложность:** Алгоритм использует поиск в глубину (DFS), поэтому требуемая память линейно зависит от максимальной глубины стека рекурсии: $O(b \times d)$ (для хранения состояний доски на каждом уровне). Учитывая малый размер доски (14 чисел типа int), потребление оперативной памяти крайне мало.

4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Программа запускается через командную строку (консоль). Для обычной игры дополнительных аргументов не требуется. Для обычной игры с её записью требуется аргумент с названием файла, куда будет записываться результат. Для автоматической игры (для тестов) требуется два аргумента: название файла, куда будет записываться результат игры, и название входного файла.

Формат запуска:

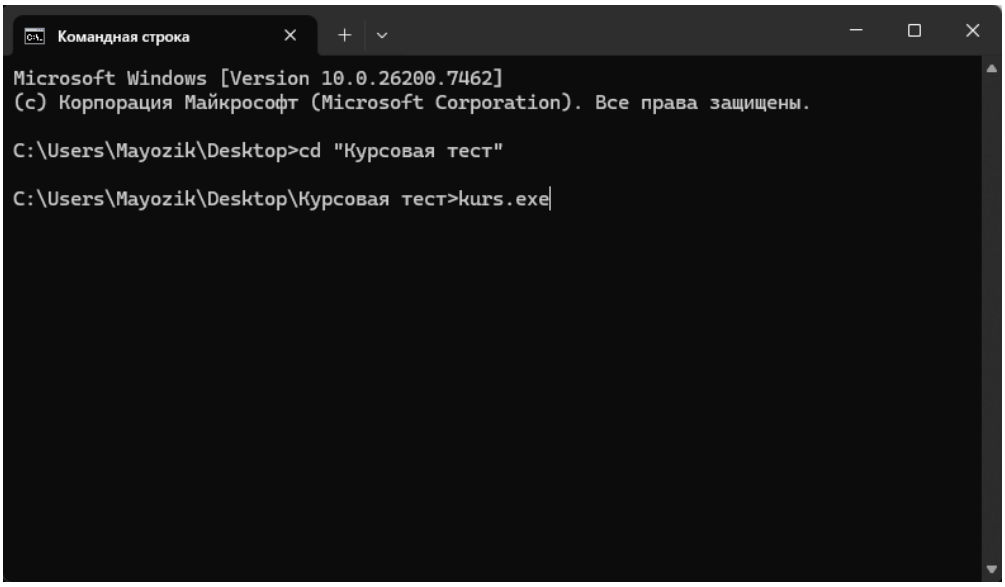

Рис. 2: Формат запуска

Управление: Игрок управляет нижним рядом лунок (индексы 1-6 слева направо). Для совершения хода необходимо ввести номер лунки и нажать Enter.

Отображение поля: Поле выводится в текстовом формате. Верхний ряд — лунки компьютера (индексы идут справа налево с точки зрения логики доски, но для удобства отображаются зеркально).

Крайние числа — каллахи (Слева — каллах Компьютера, Справа — каллах Игрока, в зависимости от реализации ориентации доски, обычно каллах игрока справа).

Пример вывода:

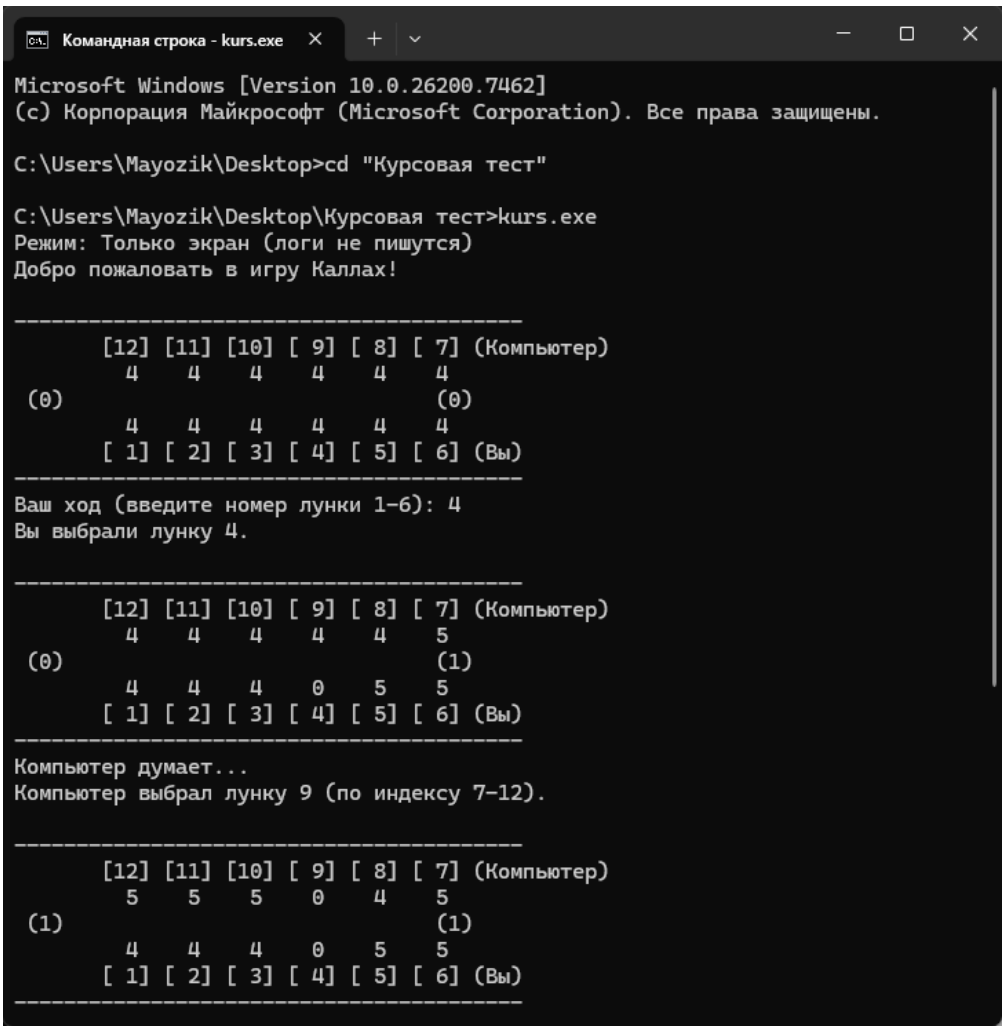

Рис. 3: Пример вывода

После каждого хода программа автоматически выводит обновленное состояние доски на новых строках. Если у игрока выпадает повторный ход (последний камень упал в каллах), программа предложит ввести ход снова.

5 ТЕСТОВЫЕ ПРИМЕРЫ

Для проверки корректности работы программы были проведены тесты ключевых механик игры.

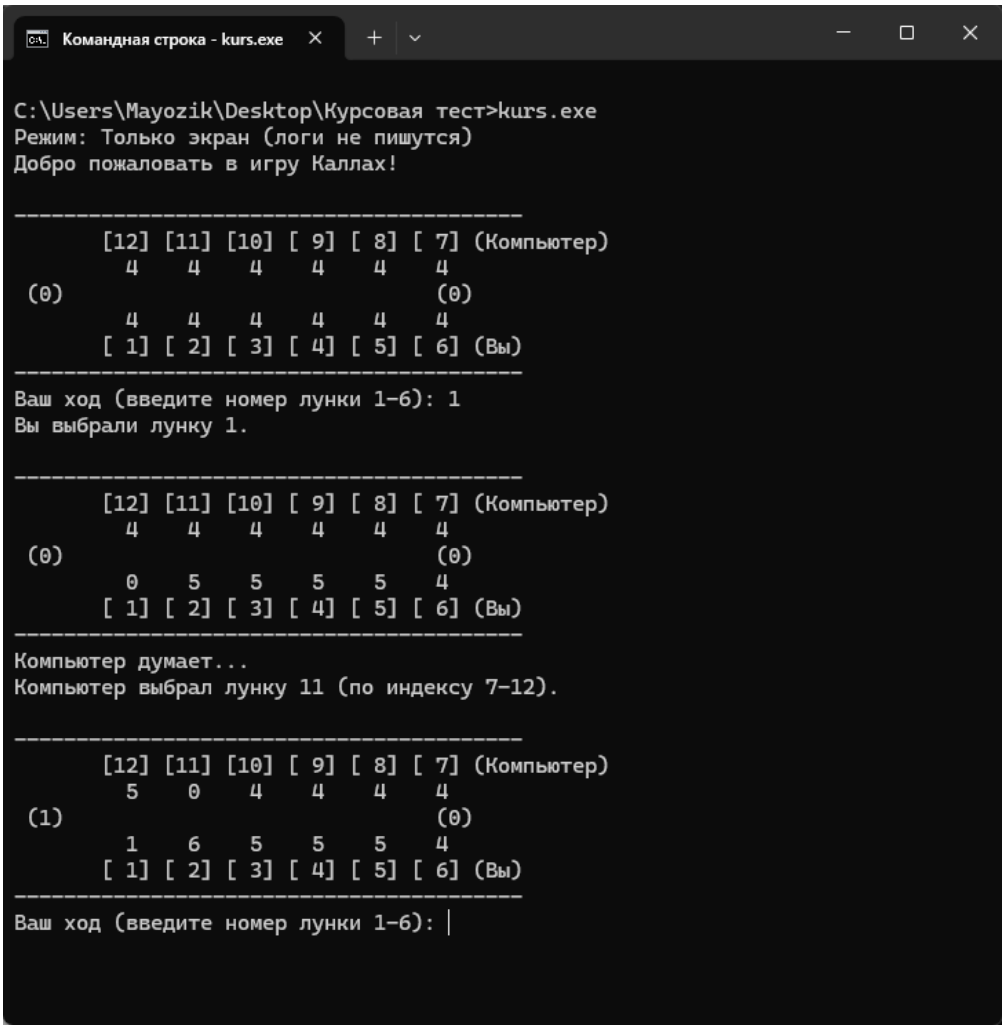

Рис. 4: Тест №1

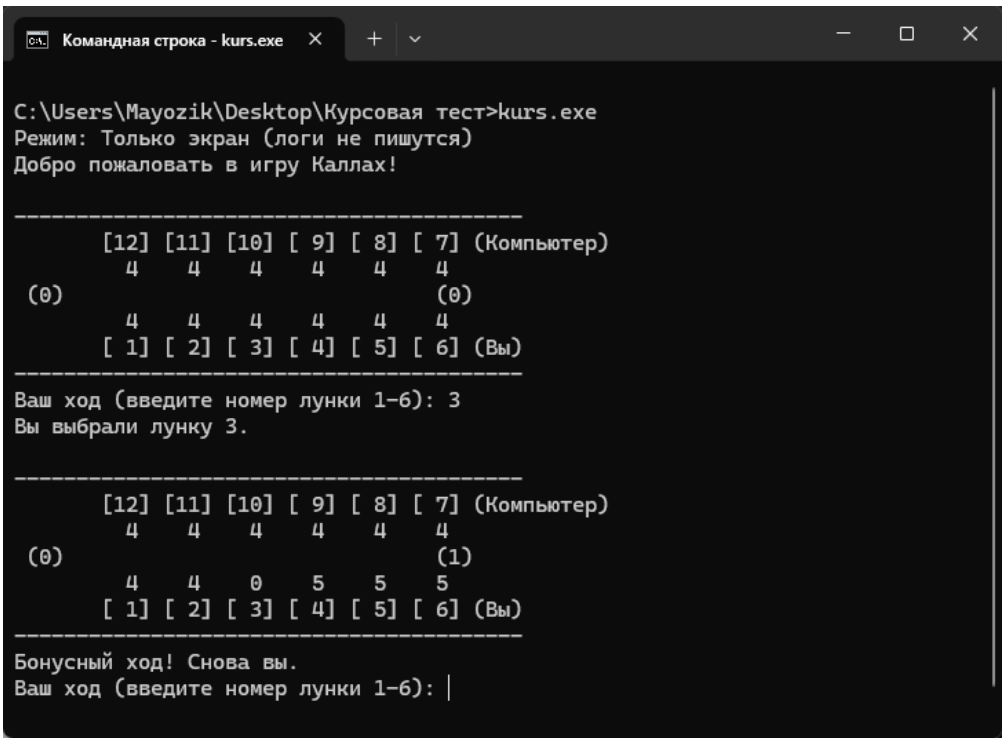

Рис. 5: Тест №2

Рис. 6: Тест №3

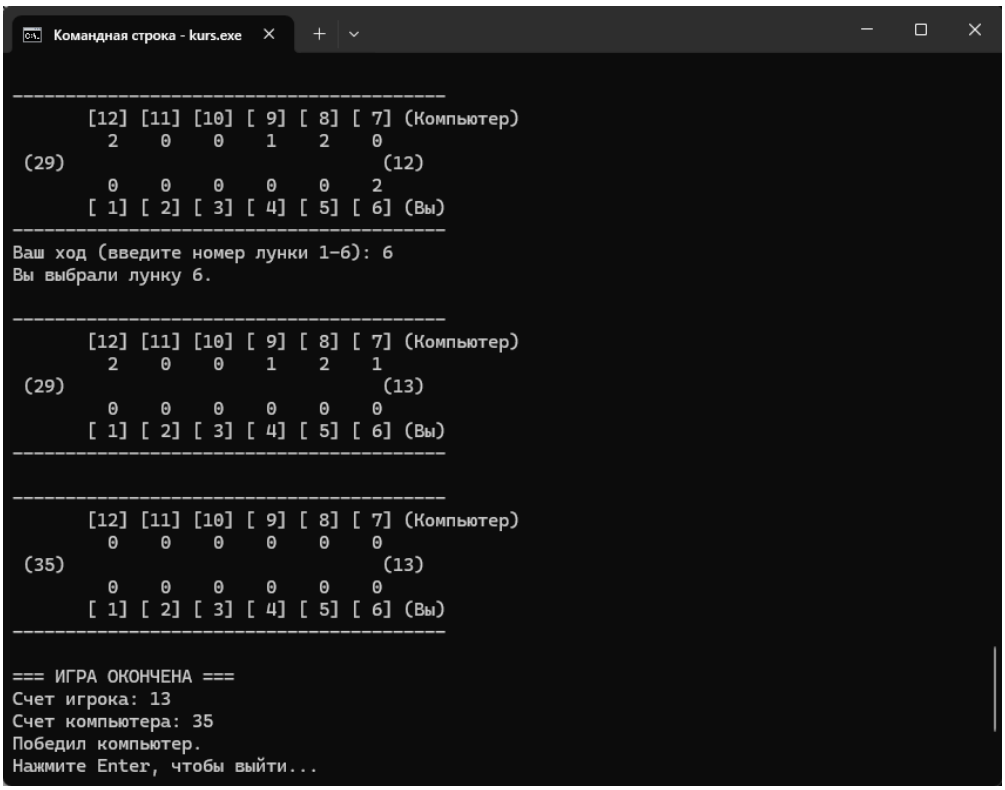

Рис. 7: Тест №4

Таблица 2: Результаты тестирования

№ теста	**Описание ситуации**	**Входные данные (Ход)**	**Ожидаемое поведение**	**Результат программы**
1	Начало игры. Первый ход игрока из 1-ой лунки	1	Камни из 1-й лунки (4 шт.) распределяются в 2, 3, 4, 5. Последний камень не в каллахе	Успешно
2	Бонусный ход. В лунке №3 лежит 4 камня.	3	Камни идут в 4, 5, 6 и в Каллах (№7). Последний камень в каллахе → Игрок ходит снова.	Успешно. Выведено сообщение "Бонусный ход! Снова вы."
3	Захват камней. В лунке №1 лежит 1 камень, в лунке №2 – 0 камней. Напротив (у ПК) есть камни.	5	Камень падает в пустую лунку №2. Забирается свой камень + все камни напротив в свой каллах.	Успешно. Счет в каллахе резко увеличился.
-	Конец игры. У игрока закончились камни.	-	Игра завершается. Все оставшиеся камни ПК перемещаются в каллах ПК. Объявляется победитель.	Успешно. Выведен финальный счет.

6 ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, реализующая игру Каллах с возможностью игры против компьютера.

Были решены следующие задачи:

1. Разработана структура данных для хранения состояния игрового поля.
2. Реализована логика правил игры, включая сложные моменты, такие как повторный ход и захват камней противоположной стороны.
3. Создан искусственный интеллект на основе алгоритма Минимакс.
4. Применена оптимизация альфа-бета отсечения, что позволило увеличить глубину просчета ходов с 4-5 до 8-10 полуходов без заметных задержек для пользователя.

Тестирование показало, что программа работает стабильно, корректно обрабатывает ввод пользователя и следует правилам игры. Алгоритмический соперник демонстрирует достойный уровень игры, способный обыграть новичка и составить конкуренцию опытному игроку. Цели курсовой работы достигнуты в полном объеме.

7 СПИСОК ЛИТЕРАТУРЫ

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: Вильямс, 2013. — 1328 с.

2. Рассел С., Норвиг П. Искусственный интеллект: современный подход. — 2-е изд. — М.: Вильямс, 2006. — 1408 с.

3. Страуструп Б. Язык программирования C++. — М.: Бином, 2011. — 1136 с.

4. Миллер Р. Теория игр для чайников. — М.: Диалектика, 2016. — 280 с.

8 ПРИЛОЖЕНИЕ

URL на репозиторий Github: https://github.com/Moisyarik/course-work-1