Реализация и сравнение алгоритмов поиска в дереве позиций игры

А. А. Синкевич

Задачи

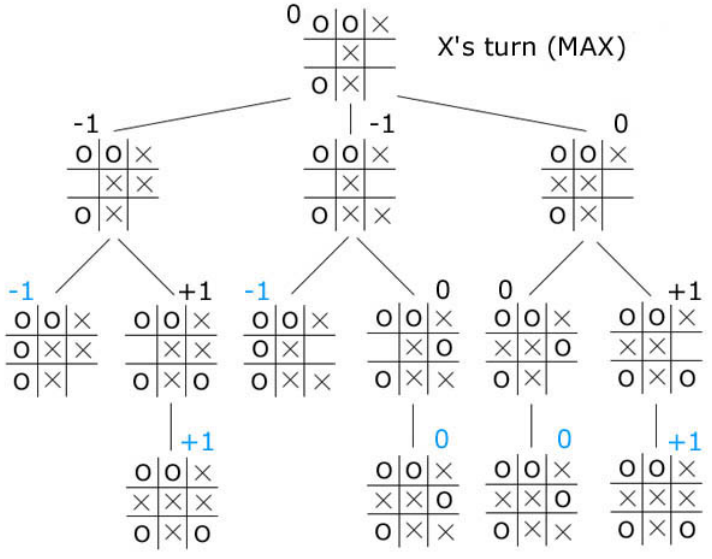
- Создание клиент-серверного приложения для игры в шашки;
- Реализация различных алгоритмов поиска в игровом дереве позиций;
- Нахождение различных функций оценки позиций;
- Сравнение реализованных алгоритмов, использующих созданные оценочные функции.

Введение

- Конечная игра — у игроков есть конечное число возможных действий для достижения победы;
- Игра с полной информацией — позволяет просчитать как свои действия, так и действия соперника;
- Игра с нулевой суммой (антагонистическая) — игра двух и более игроков с прямо противоположными интересами, выигрыши игроков противоположны (например, +1 и -1);
- Детерминированная игра — позиция зависит только от действий игроков, а не от каких-либо случайных событий.

Для таких игр (шахматы, шашки, крестики-нолики, го и многие другие) можно построить дерево позиций.

Дерево позиций игры



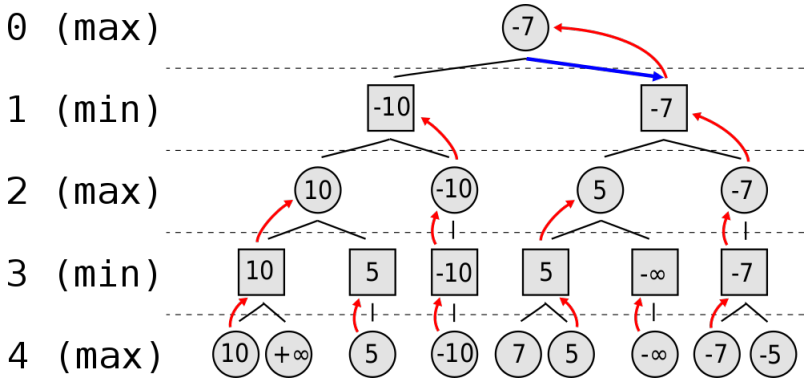
Алгоритм минимакса

Алгоритм минимакса для двух игроков: рекурсивный спуск по дереву игровых позиций, пока не будет достигнуто ограничение по глубине или позиция будет конечной. В таких случаях алгоритм возвращает значение функции оценки этой позиции. Иначе:

- максимизируется результат игрока, тогда рекурсивно вычисляются значения для всех позиций, в которые игрок может перейти из данной, и возвращается максимум из них;
- минимизируется результат игрока, тогда рекурсивно вычисляются значения для всех позиций, в которые игрок может перейти из данной, и возвращается минимум из них.

Такой алгоритм запускается для заданной позиции и игрока, результат которого требуется максимизировать, и возвращает значение минимакса. Также, перед завершением алгоритма, можно сохранить лучший ход.

Алгоритм минимакса



Алгоритм негамакса

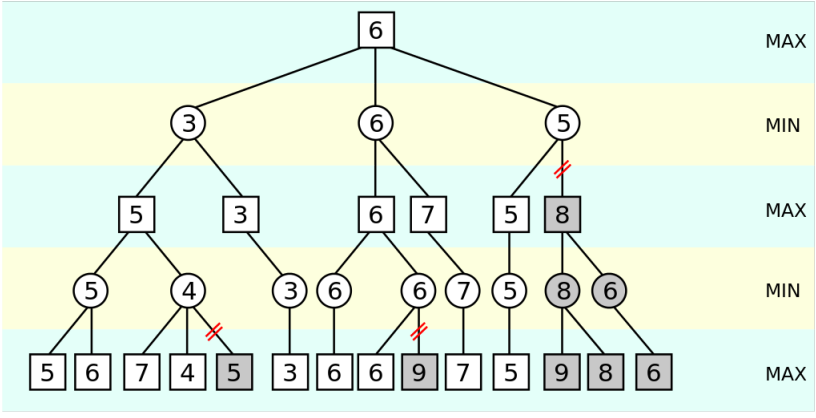
Для игр двух игроков с нулевой суммой выполняется равенство $\max(a, b) = -\min(-a, -b)$, где a, b — значения минимакса для двух позиций. Это свойство используется для более простой реализации алгоритма минимакса — алгоритма негамакс (negamax): не требуется отдельно рассматривать случай максимизируемого и минимизируемого игроков, а достаточно изменить знак у вычисленных рекурсивно значений, и всегда выбирать максимум.

Алгоритм альфа-бета отсечения

Алгоритм поддерживает два значения — α и β — представляющих минимальное значение, которое может получить максимизируемый игрок, и максимальное значение, которое может получить минимизируемый игрок. Изначально $\alpha = -\infty$ и $\beta = +\infty$. Когда максимальное значение, получаемое минимизируемым игроком, становится меньше, чем минимальное значение, получаемое максимизируемым игроком, то есть $\beta < \alpha$, то дальнейшие ходы из позиции рассматривать не имеет смысла.

Алгоритм выполняет рекурсивный спуск по дереву игровых позиций, аналогичный алгоритму негамакса, но при рекурсивном переходе в вершину передаёт $\alpha' = -\beta$, $\beta' = -\alpha$, и после рассмотрения этого хода обновляет α максимумом из текущего значения и значения рассмотренной вершины. Если окажется, что $\alpha \geqslant \beta$, то рассмотрение ходов для данной позиции завершается.

Алгоритм альфа-бета отсечения



Алгоритм NegaScout

Основной вариант — последовательность ходов, оптимальная для обоих игроков.
При рассмотрении хода (кроме первого) из позиции, проверяется, может ли он быть в основном варианте. Для этого выполняется запуск алгоритма с окном $\alpha' = -\alpha - 1$, $\beta' = -\alpha$ (при таких ограничениях будет значительно больше отсечений) и, если получен результат, не увеличивающий α, то можно считать, что ход неоптимален, и не требуется проводить полный обход поддерева. Иначе, если для результата (*res*) выполняется $\alpha < res < \beta$, то вариант, включающий данный ход, более оптимален, чем предыдущий основной вариант, и требуется обход поддерева с обычным окном ($\alpha' = -\beta$, $\beta' = -res$).

Таблица транспозиций

Транспозиции — последовательности ходов, приводящие к одной позиции. Для того, чтобы избежать многократной обработки одной и той же позиции, используются таблицы транспозиций — хеш-таблицы, в которых для некоторых позиций хранятся значения минимакса. Перед обработкой позиции алгоритм минимакса может проверить её наличие в таблице и использовать уже вычисленное значение. Но из-за ограничения поиска по глубине можно использовать это значение только в том случае, если оно было вычислено при меньшей глубине. Использование таблиц транспозиций с алгоритмом альфа-бета отсечения затруднено тем, что некоторые ветви отсекаются из-за рассмотренных ранее позиций, поэтому не всегда можно напрямую использовать значение из таблицы. Поэтому вместе со значением позиции сохраняется его тип: точная оценка, оценка снизу (позволяет обновить α), оценка сверху (позволяет обновить β).

Оценочные функции

Оценочная функция — функция, позволяющая приближённо численно выразить вероятность игрока победить из данной позиции.
Простейшая оценочная функция для шашек вычисляет разницу количеств шашек у игроков. Можно заметить, что дамка обладает большими возможностями, чем обычная шашка, и её стоит оценивать дороже. Например, можно взять стоимость обычной шашки как 1, а дамки — 2, и модифицировать вычисление разницы так, чтобы учитывалась цена шашки.
Для дальнейшего улучшения функции оценки можно заметить, что чем ближе шашка к последнему (для игрока) ряду, тем больше у неё вероятность стать в будущем дамкой. Чтобы это использовать, можно считать стоимость обычной шашки не константой, а суммой какого-либо числа и числа рядов до последнего.

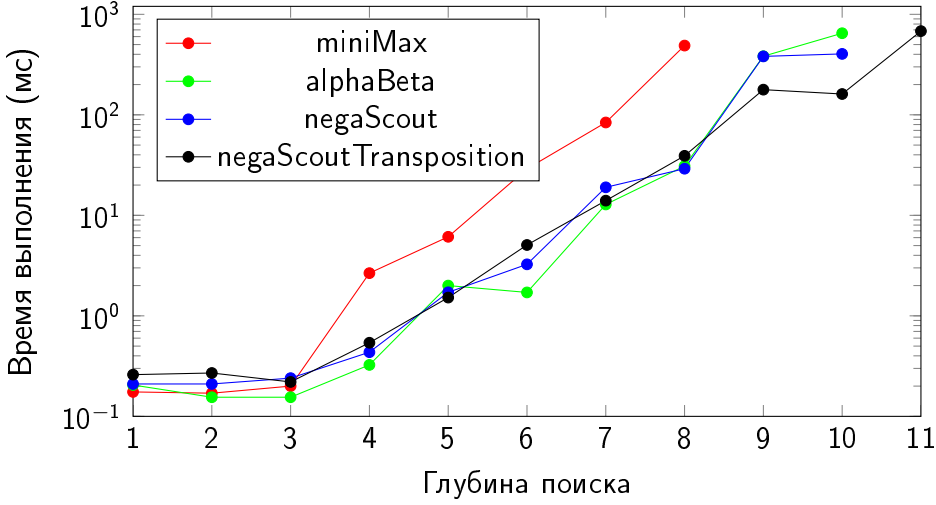


Рис.: Среднее время работы ботов в зависимости от глубины поиска

Реализация

Проект состоит из нескольких частей:

- общей библиотеки `CheckersLib`, содержащей основные классы (для доски, хода, клиента);
- клиента с графическим интерфейсом `CheckersClient`;
- сервера игры `CheckersServer`;
- клиента-бота `CheckersBot`, который содержит всех ботов (`random`, `miniMaxWeak`, `miniMax`, `alphaBeta`, `negaScout`, `negaScoutTransposition`);
- `CheckersBotTest` — небольшой программы для сравнения ботов.

Все программы написаны на языке C# и используют платформу .NET Core 3.1. Для взаимодействия по сети используется библиотека WebSocketSharp, реализующая протокол WebSocket.

Интерфейс клиента

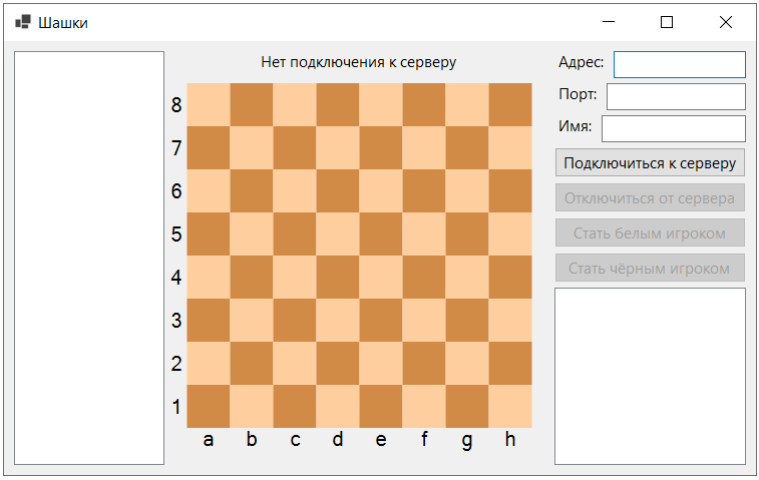


Рис.: Окно клиента после запуска

Интерфейс клиента

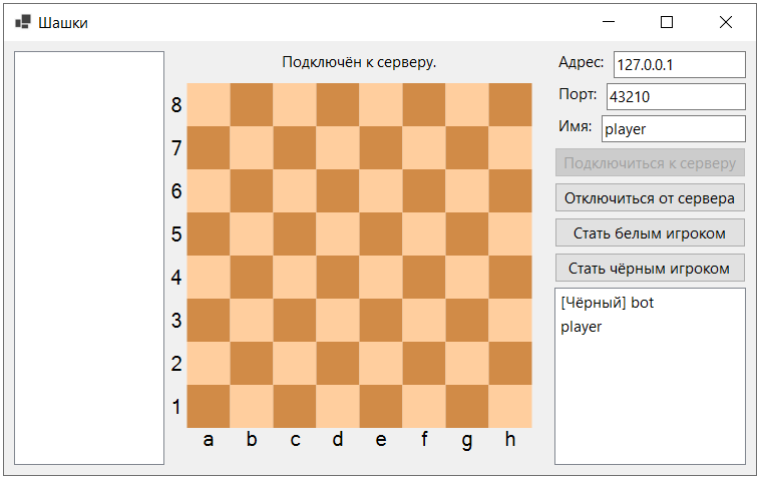


Рис.: Окно клиента после подключения к серверу

Интерфейс клиента

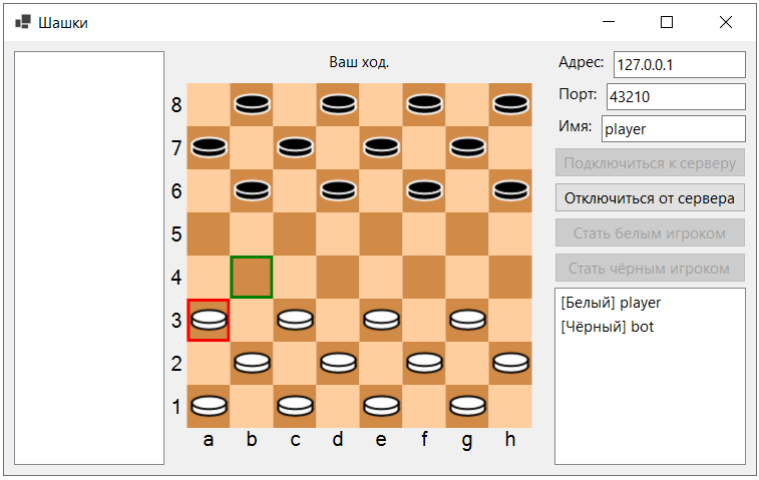


Рис.: Окно клиента при совершении хода

Интерфейс клиента

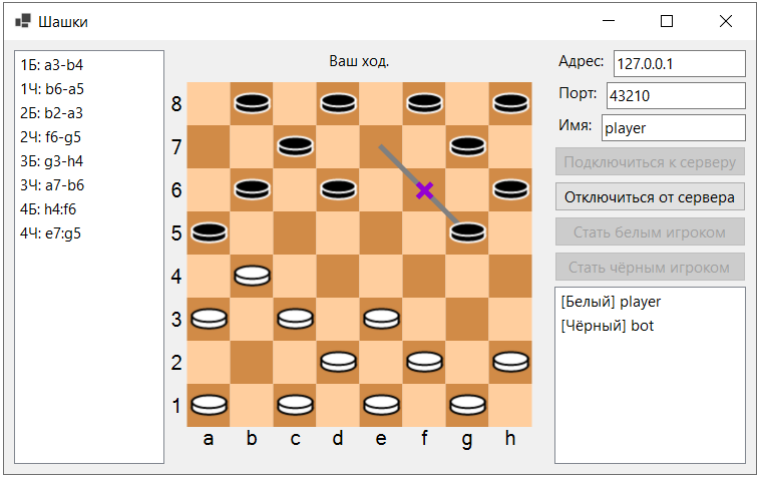


Рис.: Окно клиента при демонстрации хода соперника