МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий и искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Курсовая работа по дисциплине
Алгоритмизация и программирование
«Алгоритм принятия решений для игры Крестики-Нолики на поле 15×15»

Студент,

группа 5130201/40001 _____ **Ови Мд Шамин Ясир**

Преподаватель,

_____ **Сеннов В. Н.**

«_____»_____ **2025г.**

Санкт-Петербург, 2025

Содержание

Введение **3**

1 Постановка задачи **4**
1.1 **Структура проекта** . . . 5

2 Математическое описание **6**
2.1 Игровое поле и цель . . . 6
2.2 Стратегия принятия решения . . . 6
2.2.1 Стратегия первого хода . . . 6
2.2.1 Стратегия принятия решения . . . 6
2.2.2 Оценочная функция (эвристика) . . . 7
2.2.2 Оценочная функция (эвристика) . . . 7
2.2.3 Поиск по минимаксу с отсечением альфа-бета . . . 8
2.2.4 Выбор хода . . . 8
2.2.5 Запасной план . . . 8

3 Особенности реализации **9**
3.1 Структура класса и его ответственность . . . 9
3.2 Ключевые алгоритмы и логика . . . 9
3.3 Генерация случайности . . . 10
3.4 Интерфейс и поля класса . . . 11
3.5 Интеграция в структуру проекта . . . 11

4 Результаты работы программы **12**
4.1 Против простого BaselineEasy (500 игр X, затем 500 игр O) . . . 12
4.2 Против усиленного BaselineHarder (500 игр X, затем 500 игр O) . . . 12
4.3 Сводный анализ . . . 13
4.4 Против MyPlayer (self-play) . . . 13
4.5 Против человека (human vs bot) . . . 14

Заключение **15**

Приложение A. Полный исходный код класса MyPlayer **18**
Приложение A1. Заголовочный файл (my_player.hpp) . . . 18
Приложение A2. Реализация (my_player.cpp) . . . 19

Приложение B. Результаты тестирования **27**
Приложение B1. Против простого BaselineEasy . . . 27
Приложение B2. Против усиленного BaselineHarder . . . 28
Приложение B3. Тестирование алгоритма . . . 29

Введение

Игра «Крестики-Нолики» хорошо известна многим людям с детства. Обычно она играется на маленьком поле 3×3, но в этой курсовой работе используется более сложный вариант — поле 15×15 и победа достигается, если собрать 5 символов в ряд. Такой формат делает игру намного интереснее и позволяет применять алгоритмы из области искусственного интеллекта.

Цель моей работы — создать игрового бота, который может принимать разумные ходы и побеждать как слабого, так и сильного базового игрока. Бот должен уметь играть и первым (X), и вторым (O), а также укладываться в ограничение по времени на обдумывание хода. Это важно, потому что в настоящих играх нельзя думать слишком долго.

Для создания такого игрока я использовал идеи из теории игр, а также простые и понятные эвристики. В том числе использовался алгоритм минимакс и метод отсечения альфа-бета, чтобы бот мог предсказывать будущие ходы и принимать оптимальные решения.

Такая работа полезна не только для обучения, но и как пример того, как можно применять алгоритмы искусственного интеллекта в реальных задачах, например, в планировании, управлении или играх.

1 Постановка задачи

В рамках данной курсовой работы мне было предложено создать игрового бота для игры «Крестики-Нолики» на поле размером 15×15, где победа достигается при выстраивании 5 символов в ряд (по горизонтали, вертикали или диагонали).

Проект был предоставлен в виде уже готовой структуры. Структура проекта и описание были получены с официального репозитория преподавателя: https://github.com/vsennov/tictactoe-course. В папке `src/core/` находится основная логика игры, включая управление состоянием поля и обработку ходов. В папке `libs/` лежит скомпилированная библиотека с базовым игроком, с которым нужно соревноваться. Также в проекте есть папка `tests/` с тестами, где можно запускать моего бота против других ботов или человека.

Моя задача состояла в том, чтобы:

- Понять устройство проекта и как всё работает;
- Написать своего игрока (бота) в файлах `src/player/my_player.cpp` и `my_player.hpp`;
- Сделать так, чтобы бот мог играть разумно, понимать игровую ситуацию, принимать хорошие решения и выигрывать;
- Протестировать своего бота против других — как в одиночных играх, так и в пакетных тестах;
- Убедиться, что мой бот работает быстро (около 50–100 мс на ход) и стабильно;
- Научить его побеждать как слабого, так и усиленного базового игрока, как в роли первого игрока (X), так и второго (O).

1.1 Структура проекта

```
tictactoe-div/
|- CMakeLists.txt               # Главный файл сборки проекта
|- README.md                    # Описание проекта и инструкции
|- shell.nix                    # Конфигурация окружения Nix (опционально)
|- libs/
|    \- libtttcore.a            # Предкомпилированная библиотека с базовым
|
|- build/
|   |- CMakeCache.txt           # Кэш сборки
|   |- CMakeFiles/              # Вспомогательные файлы CMake
|   |- compile_commands.json    # Файл для поддержки автодополнения в IDE
|   |- libtttplayer.a           # Скомпилированный файл с моим игроком
|   \- tests/                   # Каталог с тестовыми исполняемыми файлами
|- src/
|   |- core/                    # Основная логика игры
|   |   |- baseline.hpp         # Интерфейс к базовому игроку
|   |   |- event.cpp/hpp        # События в игре
|   |   |- game.cpp/hpp         # Основной игровой цикл
|   |   \- state.cpp/hpp        # Состояние игрового поля
|   |- player/                  # Моя реализация
|   |   |- my_player.cpp        # Файл с реализацией моего алгоритма
|   |   |- my_player.hpp
|   |   |- my_observer.cpp/hpp  # Вывод отладочной информации
|   \- remote/                  # Код для игры по сети (опционально)
|       |- cli_client.cpp
|       |- cli_server.cpp
|       |- client.cpp/hpp
|       |- dto.proto
|       |- dto_utils.cpp/hpp
|       \- zmq_utils.hpp
|- tests/
    |- CMakeLists.txt              # Файл сборки тестов
    |- human_player.cpp/hpp        # Игрок-человек для ручного тестирования
    |- test_my_player.cpp          # Мой игрок против самого себя
    |- test_baseline.cpp           # Мой игрок против базового игрока
    |- test_my_player_vs_human.cpp # Мой игрок против человека
    |- test_stats.cpp              # Серия игр между моими ботами
    |- test_stats.hpp
    \- test_stats_vs_baseline.cpp  # Серия игр против базового игрока
```

2 Математическое описание

Для решения поставленной задачи была разработана стратегия принятия решений, основанная на сочетании простых правил, оценки позиции и метода минимакса с отсечением альфа-бета. Алгоритм работы игрока включает в себя следующие этапы:

2.1 Игровое поле и цель

Игровое поле представляется в виде сетки B размером $M \times N$, где каждая ячейка $B_{x,y} \in \{\mathrm{X}, \mathrm{O}, \varnothing\}$. Здесь символы X и O представляют игроков, а $\varnothing$ — пустую ячейку.

Цель — построить последовательность из $L = 5$ одинаковых символов по горизонтали, вертикали или диагонали. Линия:

$$L_i = \{B_{x_0,y_0}, B_{x_1,y_1}, \ldots, B_{x_4,y_4}\}$$

считается *победной*, если:

$$\forall k \in \{0,1,2,3,4\}, \quad B_{x_k,y_k} = s$$

для некоторого $s \in \{\mathrm{X}, \mathrm{O}\}$.

2.2 Стратегия принятия решения

Игровой бот начинает партию, размещая символ в центре поля — этот ход обеспечивает равный доступ ко всем направлениям и служит хорошей отправной точкой. Далее, на каждом ходу бот последовательно анализирует ситуацию: определяет возможные угрозы (как свои, так и со стороны противника), оценивает позиции с помощью специальной эвристической функции и применяет упрощённый алгоритм минимакс с отсечением альфа-бета [5] для выбора наиболее перспективного хода. Такой подход позволяет сочетать тактическую защиту и нападение с эффективным использованием времени на расчёт.

2.2.1 Стратегия первого хода

Если это первый ход в игре, бот выбирает центральную ячейку игрового поля. Это означает, что координаты хода определяются как:

$$(x, y) = \left(\left\lfloor \frac{M}{2} \right\rfloor, \left\lfloor \frac{N}{2} \right\rfloor \right)$$

Такой ход обеспечивает равный доступ ко всем направлениям и создаёт сбалансированную стартовую позицию для последующего стратегического анализа и развития атаки.

2.2.2 Оценочная функция (эвристика)

Каждой потенциальной позиции присваивается числовая оценка, отражающая её стратегическую ценность. Вводится функция $f(B, s)$, где:

- B — текущее состояние игрового поля;
- s — символ бота (X или O);
- k — количество возможных пятёрок (последовательностей длиной 5) на поле;
- i — индекс текущей пятёрки;
- $h(c, e)$ — функция, оценивающая конкретную пятёрку;
- c — количество символов игрока в рассматриваемой линии;
- e — количество открытых концов в линии (возможные значения: 0, 1, 2).

Формула для оценки одной линии:

$$h(c,e) = \begin{cases} 1000000 & \text{если } c \geq 5 \\ 3500 & \text{если } c = 4 \wedge e = 2 \\ 1500 & \text{если } c = 4 \wedge e = 1 \\ 400 & \text{если } c = 3 \wedge e = 2 \\ 120 & \text{если } c = 3 \wedge e = 1 \\ 60 & \text{если } c = 2 \wedge e = 2 \\ 20 & \text{если } c = 2 \wedge e = 1 \\ 1 & \text{в остальных случаях} \end{cases}$$

Итоговая оценка всей позиции:

$$f(B, s) = \sum_{i=1}^{k} h(c_i, e_i)$$

Для учёта активности противника используется скорректированная формула общей оценки:

$$\text{Score} = f(B, s) - 2 \cdot f(B, s')$$

где s' — символ соперника (противоположный s).

2.2.3 Поиск по минимаксу с отсечением альфа-бета

Для оценки будущих ходов используется упрощённый алгоритм минимакса на глубине $d = 1$ с отсечением по альфа-бета [5].

Пусть $V(B, d, \alpha, \beta, \max)$ — оценка доски B на глубине d:

- Если игрок стремится *максимизировать* результат:

$$V(B, d, \alpha, \beta, \text{true}) = \max_{m \in M} V(B_m, d-1, \alpha, \beta, \text{false})$$

- Если *минимизирует* (противник):

$$V(B, d, \alpha, \beta, \text{false}) = \min_{m \in M} V(B_m, d-1, \alpha, \beta, \text{true})$$

Здесь B_m — доска после хода $m \in M$, а M — множество всех допустимых ходов. Если $\beta \leq \alpha$, ветвь отсекается (больше не рассматривается).

2.2.4 Выбор хода

1. Генерируются все допустимые ходы.
2. Каждому ходу присваивается эвристическая оценка.
3. Отбираются 5–6 лучших по оценке ходов.
4. На каждый из них применяется минимакс с глубиной 1.
5. Выбирается ход с наилучшей ожидаемой оценкой.

2.2.5 Запасной план

Если не найдено сильных ходов, бот делает первый доступный допустимый ход.

Такой подход позволяет боту эффективно играть, соблюдая ограничение по времени на ход (около 50–100 мс), быть тактически гибким и учитывать как свои, так и вражеские угрозы.

3 Особенности реализации

Логика игрока была реализована на языке C++ с использованием принципов объектно-ориентированного программирования. Реализация находится в файлах `my_player.cpp` и `my_player.hpp`, расположенных в каталоге `src/player/`. Класс игрока называется `MyPlayer` и наследуется от интерфейса `IPlayer`, определённого в проекте.

3.1 Структура класса и его ответственность

- `MyPlayer` хранит:
  - Символ игрока `m_sign` — `X` или `O`;
  - Имя игрока `m_name` — для отображения в игре.
- Основная логика принятия решения реализована в методе:
  - `Point make_move(const State& state);` — анализирует игровое состояние и выбирает лучший ход.

3.2 Ключевые алгоритмы и логика

1. **Центральный ход первым**

   Если ход первый (`state.get_move_no() == 0`), игрок пытается занять центральную ячейку:

```
1       int cx = cols / 2, cy = rows / 2;
2       if (state.get_value(cx, cy) == Sign::NONE) return {cx, cy};
3
```

2. **Проверка победы и защиты**

   До оценки по эвристике бот проверяет, есть ли выигрышный ход или необходимость блокировки соперника:

```
1       if (is_winning_move(state, x, y, m_sign)) return {x, y};
2       if (is_winning_move(state, x, y, enemy)) return {x, y};
3
```

3. **Обнаружение угроз**

   - `is_open_four_threat(...)` — ищет шаблоны типа _OOOO_;
   - `is_critical_threat(...)` — ищет двойные угрозы (вилки);
   - `is_manual_winning_opportunity(...)` — симуляция потенциальных победных линий.

   Проверка проводится по 4 направлениям с использованием массива направлений:

```
    const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};

```

4. **Оценка позиции (эвристика)**

   Каждая возможная позиция оценивается по числу символов в линии и наличию открытых концов. Положительные шаблоны для игрока добавляют очки, опасные шаблоны для противника — уменьшают.

```
    if (count == 4 && open_ends == 2) base = 3500;
    else if (count == 3 && open_ends == 2) base = 400;

```

5. **Выбор лучших кандидатов**

   - Все пустые ячейки оцениваются;
   - Отбираются топ-5–6 наилучших;
   - Проводится случайная перестановка (shuffle) и частичная сортировка.

6. **Минимакс с отсечением (глубина 1)**

   Для отобранных кандидатов применяется минимакс с альфа-бета отсечением:

```
    int score = minimax(next, depth - 1, alpha, beta, !maximizing,
  self, enemy);

```

   Это помогает смоделировать следующий возможный ход противника и выбрать оптимальный ответ.

3.3 Генерация случайности

Для разнообразия поведения при равных оценках используется стандартная функция `std::rand()`:

```
  int j = std::rand() % (i + 1);

```

3.4 Интерфейс и поля класса

Методы и поля класса MyPlayer

- `Sign m_sign;` — символ игрока: X или O
- `const char *m_name;` — имя игрока
- `void set_sign(Sign sign);` — установка символа игрока
- `Point make_move(const State&);` — логика принятия решения
- `int evaluate_board(...);` — эвристическая оценка
- `int minimax(...);` — минимакс с отсечением
- `bool is_open_four_threat(...);` — обнаружение шаблона _OOOO_
- `bool is_critical_threat(...);` — обнаружение двойной угрозы

3.5 Интеграция в структуру проекта

- Игрок интегрирован через тесты в папке `tests/`:
  - `test_my_player.cpp`
  - `test_baseline.cpp`
  - `test_stats_vs_baseline.cpp`
  - `test_my_player_vs_human.cpp`
- Алгоритм реализован в соответствии с интерфейсами `core/game.hpp` и совместим с игровым движком.

Полный исходный код приведён в Приложении А.

4 Результаты работы программы

В этом разделе представлены результаты тестирования ИИ-игрока `MyPlayer` в различных сценариях: против простого и сложного бота, против самого себя, а также в игре против человека. Тесты проводились с помощью встроенных утилит проекта из папки `tests/`.

4.1 Против простого BaselineEasy (500 игр X, затем 500 игр O)

Против простого базового игрока MyPlayer показал уверенные результаты. В роли первого игрока (X) он одержал 443 побед из 500 возможных, что составляет 88.6%. В роли второго игрока (O) результат составил 390 побед из 500 игр, то есть 78%. Среднее время, затрачиваемое на ход в обеих ролях, варьировалось от 3.2 до 3.6 миллисекунд, что соответствует требованиям по производительности.
Приложении B1.

4.2 Против усиленного BaselineHarder (500 игр X, затем 500 игр O)

Играя против усиленного базового бота, MyPlayer столкнулся с более серьёзным сопротивлением. В роли X удалось выиграть 198 игр из 500 (39.6%), а в роли O — 146 из 500 (29.2%). Несмотря на снижение доли побед, среднее время ответа оставалось в пределах нормы — около 3.5–3.7 миллисекунд на ход.
Приложении B2.

4.3 Сводный анализ

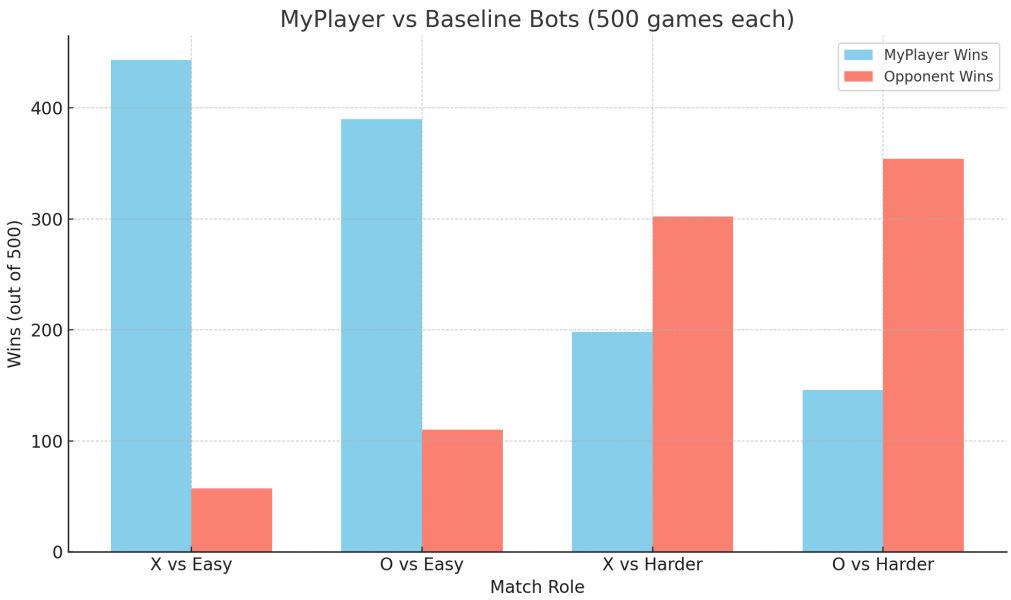

Рис. 1. Сравнительная диаграмма побед против BaselineEasy и BaselineHarder

Как видно из результатов, MyPlayer демонстрирует высокую эффективность против слабого игрока и удовлетворительную стратегию против более сильного. Особенно хорошо он проявляет себя в роли первого игрока, где вероятность победы значительно выше. Среднее время отклика стабильно укладывается в пределы 4 миллисекунд, что подтверждает оптимизацию алгоритма под жёсткие временные ограничения.

4.4 Против MyPlayer (self-play)

При тестировании игры самого MyPlayer против самого себя, результаты распределились почти поровну: один экземпляр одержал 231 победу, а другой — 263. Зафиксировано также 6 ничейных результатов. Среднее время на ход в этом режиме составило около 3.5 миллисекунд, что подтверждает стабильность и эффективность алгоритма при симметричной игре.
Приложении В3.

4.5 Против человека (human vs bot)

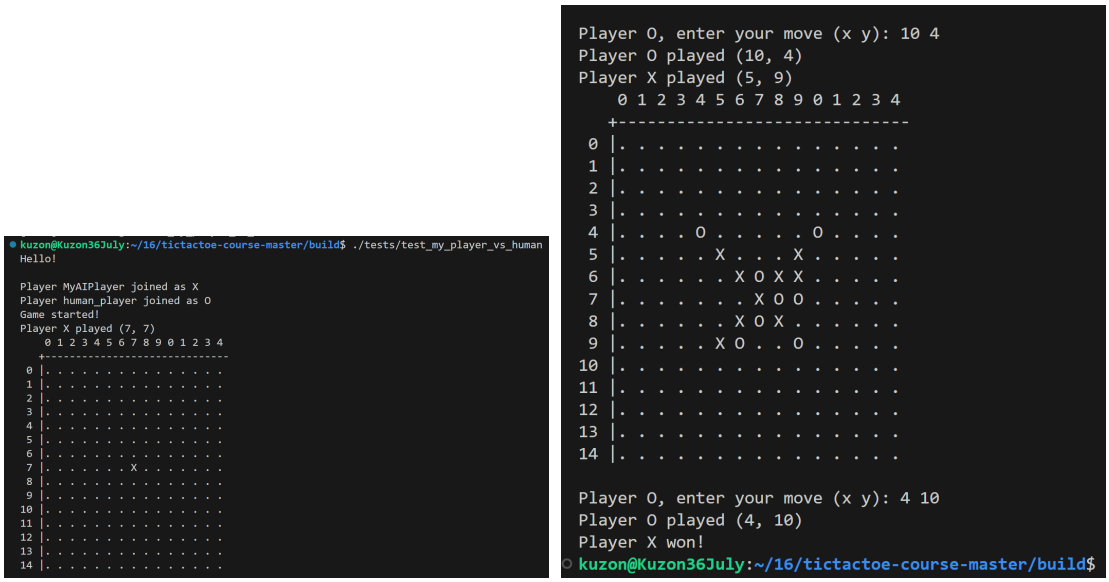

Рис. 2. Интерактивная игра против человека: MyPlayer побеждает

В ходе тестирования с участием человека ИИ-игрок `MyPlayer` продемонстрировал корректную работу программы и одержал победу, что подтверждает его применимость в реальных условиях. В целом, `MyPlayer` успешно побеждает `BaselineEasy` в более чем 80% игр. Против `BaselineHarder` бот всё ещё уступает, особенно во второй роли (O), однако показывает стабильный и конкурентоспособный результат. Производительность остаётся высокой — среднее время на ход стабильно ниже 6 мс, что значительно лучше установленного лимита в 100 мс. ИИ демонстрирует адекватную логику и может быть дополнительно усилен стратегиями, особенно в защите при игре вторым.

Заключение

В рамках данного проекта был успешно разработан и реализован конкурентоспособный бот для игры в «крестики-нолики» на поле размером 15×15 с правилом «пять в ряд». Целью было создание эффективного и интеллектуального агента, который корректно соблюдает правила игры, анализирует состояние поля и принимает стратегические решения, укладываясь при этом в лимит времени менее 100 миллисекунд на ход.

Достижения и реализация

В рамках проекта был разработан и реализован класс `MyPlayer` на языке C++. В него были внедрены ключевые алгоритмы, включая распознавание угроз (выигрышных позиций, открытых четвёрок, двойных угроз), эвристическую оценку позиции, а также алгоритм минимакса с отсечениями альфа-бета на глубине 1. Для повышения непредсказуемости реализован отбор и случайное перемешивание кандидатов на ход. Общий объём кода составляет около 380 строк, размещённых в файлах `my_player.cpp` и `my_player.hpp`.

Результаты тестирования

Сценарий	**Победы**	**Поражения**	**Ничьи**	**Время (мс)**
vs BaselineEasy (X/O)	443 / 390	57 / 110	0	∼3.2–3.6
vs BaselineHarder (X/O)	198 / 146	302 / 354	0	∼3.5–3.7
vs MyPlayer (self-play)	∼50/50	∼50/50	6	∼3.5
vs Human player	Победа	–	–	–

Таблица 1. Итоги тестирования игрока MyPlayer

Примечание: все тесты проводились с помощью официальных утилит из папки `tests/`. Среднее время хода во всех сценариях стабильно меньше 100 мс.

Полученные знания

В рамках реализации данного проекта были получены важные практические навыки. В частности, было освоено применение алгоритма минимакса с отсечением альфа-бета, разработка эвристических функций для оценки позиций на больших игровых полях, обработка различий между стратегиями первого и второго игрока. Кроме того, значительно углубились знания и навыки программирования на языке C++, архитектуры модульных систем и отладки программ.

Затраты времени и статистика

Общая продолжительность работы над проектом составила около четырёх недель. Из них около двух недель было уделено планированию и проектированию [1, 2, 3, 4, 5], около пяти-шести дней — написанию и отладке основного кода, и ещё три-четыре дня — тестированию и оптимизации. В рамках проекта было реализовано более 15 функций, написано примерно 380 строк кода, который полностью интегрирован в общую структуру проекта на языке C++.

Ограничения и направления для улучшения

Несмотря на достигнутые успехи, в текущей реализации существуют определённые ограничения. Игра за второго игрока (O) пока остаётся слабее по сравнению с первым игроком (X). Алгоритм минимакса с глубиной 1 не позволяет распознавать угрозы на отдалённой перспективе. Также логика обнаружения двойных угроз (вилок) реализована вручную, что ограничивает гибкость стратегии.

Возможные улучшения

Будущие доработки могут включать углубление поиска, например, переход к минимаксу с глубиной 2 или использование итеративного углубления. Также можно интегрировать библиотеки дебютов, особенно для второго игрока. Дополнительные возможности включают использование метода Монте-Карло (MCTS) для большей вариативности стратегий, применение генетических алгоритмов и обучение с подкреплением (Reinforcement Learning), что может значительно усилить игровой интеллект.

Список использованной литературы

[1] Павловская, Т. А. C/C++ Программирование на языке высокого уровня. / Т. А. Павловская. — Санкт-Петербург : Питер, 2004. — 464 с.

[2] Дейтел, Х. М. Как программировать на C++. / Х. М. Дейтел, П. Дж. Дейтел ; пер. с англ. — Москва : Бином, 2004. — 1454 с.

[3] Павловская, Т. А., Щупак, Ю. А. C++. Объектно-ориентированное программирование. Практикум. — Санкт-Петербург : Питер, 2005. — 352 с.

[4] Лекции преподавателя, доступные по ссылке: http://13633-oop.mooo.com/.

[5] GeeksforGeeks. Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning). — URL: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/ (дата обращения: 2 июня 2025 г.).

Приложение А.
Полный исходный код класса MyPlayer

Приложение А1. Заголовочный файл (my_player.hpp)

```
#pragma once

#include "core/game.hpp"

namespace ttt::my_player {

  using game::Event;
  using game::IPlayer;
  using game::Point;
  using game::Sign;
  using game::State;

  // AI player for Tic Tac Toe 15X15 (mine)
  class MyPlayer : public IPlayer {

    Sign m_sign = Sign::NONE;            // my sign (X or O)
    const char *m_name;                  // name of the player

    private:
    // utility functions
    static Sign get_enemy(Sign s);      // enemy sign
    static bool is_valid_empty(const State &state, int x, int y); // check if cell is empty
    static bool is_in_bounds(int x, int y, int cols, int rows);   // check if coordinates are in bounds
    static bool is_winning_move(State state, int x, int y, Sign sign);  // check for winning move
    static bool is_manual_winning_opportunity(const State &state, int x, int y, Sign sign); // check for manual win

    static int evaluate_board(const State &state, Sign self, Sign enemy); // board scoring function
    static int minimax(State state, int depth, int alpha, int beta, bool maximizing, Sign self, Sign enemy); // minimax algorithm

    bool is_open_four_threat(const State& state, int x, int y, Sign sign);   // detect _OOOO_ threat
    bool is_critical_threat(const State& state, int x, int y, Sign sign);    // detect fork situations

    public:
    // interface methods
    MyPlayer(const char *name) : m_sign(Sign::NONE), m_name(name) {}
    void set_sign(Sign sign) override;              // set player sign
    Point make_move(const State &game) override;   // main decision function
    const char *get_name() const override;         // return player name
  };
```

```
40
41  }; // namespace ttt::my_player
```

Листинг 1. Заголовочный файл my_player.hpp

Приложение А2. Реализация (my_player.cpp)

```
1
2   #include "my_player.hpp"
3   #include <cstdlib>
4   #include <iostream>
5   using namespace std;
6
7   namespace ttt::my_player {
8
9     using game::State;
10    using game::Sign;
11    using game::Point;
12
13    void MyPlayer::set_sign(Sign sign) {
14      m_sign = sign;//set my sign (X or O)
15    }
16
17    const char *MyPlayer::get_name() const {
18      return m_name;//name of the player
19    }
20
21    Sign MyPlayer::get_enemy(Sign s) {//select enemy sign
22      return s == Sign::X ? Sign::O : Sign::X;
23      //if X is my sign, O is enemy and vice versa
24    }
25
26    bool MyPlayer::is_valid_empty(const State &state, int x, int y) {
27      return state.get_value(x, y) == Sign::NONE;//returns true if the
    cell is empty
28    }
29
30    bool MyPlayer::is_in_bounds(int x, int y, int cols, int rows) {
31      return x >= 0 && x < cols && y >= 0 && y < rows;//returns true if
    the cell is in bounds
32    }
33
34    bool MyPlayer::is_winning_move(State state, int x, int y, Sign sign)
    {//returns true if the move is winning
35      if (!is_valid_empty(state, x, y)) return false;
36      if (state.process_move(sign, x, y) == game::MoveResult::ERROR)
    return false;//invalid move
37      return state.get_status() == game::Status::ENDED && state.
    get_winner() == sign;
38    }
39
40    bool MyPlayer::is_manual_winning_opportunity(const State &state, int
    x, int y, Sign sign) {//returns true if the move is winning
41      if (!is_valid_empty(state, x, y)) return false;
```

```
42      const int dirs[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};//->,top,\,/
43      const int win_len = 5;
44
45      for (int d = 0; d < 4; ++d) {//forward
46        int count = 1;//simulate
47
48        for (int i = 1; i < win_len; ++i) {
49          int nx = x + dirs[d][0]*i;
50          int ny = y + dirs[d][1]*i;
51          if (!is_in_bounds(nx, ny, state.get_opts().cols, state.
    get_opts().rows)) break;
52          if (state.get_value(nx, ny) == sign) count++;
53          else break;
54        }
55
56        for (int i = 1; i < win_len; ++i) {
57          int nx = x - dirs[d][0]*i;
58          int ny = y - dirs[d][1]*i;
59          if (!is_in_bounds(nx, ny, state.get_opts().cols, state.
    get_opts().rows)) break;
60          if (state.get_value(nx, ny) == sign) count++;//if the cell
    contains the same sign
61          else break;
62        }
63        if (count >= win_len) return true;
64      }
65      return false;
66    }
67
68
69    //to understad situatioin
70    //higher score = better for my bot
71    //lower score = better for enemy
72    //using scores
73    int MyPlayer::evaluate_board(const State &state, Sign self, Sign
    enemy) {
74      static const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
75      int score = 0;
76
77      const int rows = state.get_opts().rows;
78      const int cols = state.get_opts().cols;
79
80      for (int y = 0; y < rows; ++y) {//full board
81        for (int x = 0; x < cols; ++x) {
82          Sign s = state.get_value(x, y);//get sign at position(x,y)
83          if (s == Sign::NONE) continue;//skip empty cells
84
85          for (int d = 0; d < 4; ++d) {//5 cells ->top,\,/
86            int count = 0, open_ends = 0;//same sign and empty cells
87
88            for (int i = 0; i < 5; ++i) {
89              int nx = x + dirs[d][0] * i;
90              int ny = y + dirs[d][1] * i;
91
92              if (!is_in_bounds(nx, ny, cols, rows)) break;//stop if out
    of bounds
```

```
93
94              Sign val = state.get_value(nx, ny);
95              if (val == s) count++;
96              else if (val == Sign::NONE) {
97                open_ends++; break;
98              }
99
100             else break;//enemy sign : stop
101           }
102
103           for (int i = 1; i < 5; ++i) {//4 cells
104             int nx = x - dirs[d][0] * i;
105             int ny = y - dirs[d][1] * i;
106
107             if (!is_in_bounds(nx, ny, cols, rows)) break;
108             Sign val = state.get_value(nx, ny);
109
110             if (val == s) count++;
111             else if (val == Sign::NONE) {
112               open_ends++; break;
113             }
114
115             else break;
116           }
117
118           if (count >= 5) return s == self ? 1000000 : -1000000;//max
    score win
119
120           //heuristic scoring
121           int base = 0;
122           if (count == 4) base = open_ends == 2 ? 3500 : 1500;
123           else if (count == 3) base = open_ends == 2 ? 400 : 120;
124           else if (count == 2) base = open_ends == 2 ? 60 : 20;
125
126           else base = count;
127
128           int center_bonus = ((x > cols/3 && x < 2*cols/3) && (y >
    rows/3 && y < 2*rows/3)) ? 5 : 0;
129           if (s == self) score += base + center_bonus;//adding
    setuation under my sign
130           else score -= base * 2;//subtracting enemy good situation
131
132         }
133       }
134     }
135     return score;
136   }
137
138
139   //recursively evaluates board outcomes using Minimax with alpha-beta
    pruning
140   //maximizing = true -> my turn
141   //maximizing = false -> enemy turn
142   //alpha = best score for maximizing player
143   //beta = best score for minimizing player
144   //returns best score
```

```
145     //depth = 0 -> leaf node normally
146     //if not working then depth++
147     //depth = 1 -> one move ahead
148     int MyPlayer::minimax(State state, int depth, int alpha, int beta,
    bool maximizing, Sign self, Sign enemy) {
149       if (depth == 0  state.get_status() == game::Status::ENDED) {
150         return evaluate_board(state, self, enemy);
151       }
152 
153       int best = maximizing ? -1000000 : 1000000;//ini base score : very
     low for maximizing and very high for minimizing
154       const int rows = state.get_opts().rows;
155       const int cols = state.get_opts().cols;
156 
157       for (int y = 0; y < rows; ++y) {
158         for (int x = 0; x < cols; ++x) {
159           if (!is_valid_empty(state, x, y)) continue;
160 
161           State next = state;//simulate move a new game state
162 
163           if (next.process_move(maximizing ? self : enemy, x, y) == game
    ::MoveResult::ERROR) continue;//applying move for either max-ing or
    min-ing player
164 
165           int score = minimax(next, depth - 1, alpha, beta, !maximizing,
     self, enemy);//recursivly evaluate the next state
166 
167           if (maximizing) {//if maximizing : choose the higher score
168             if (score > best) best = score;
169             if (score > alpha) alpha = score;//update alpha
170           }
171           else {//if minimizing : choose the lower score
172             if (score < best) best = score;
173             if (score < beta) beta = score;//update beta
174           }
175 
176           if (beta <= alpha) return best;//alpha-beta pruning: cut off
    branches
177         }
178       }
179 
180       return best;
181     }
182 
183     //decision function
184     Point MyPlayer::make_move(const State &state) {
185 
186       const int rows = state.get_opts().rows;
187       const int cols = state.get_opts().cols;
188 
189       Sign enemy = get_enemy(m_sign);//enemy
190 
191 
192       if (state.get_move_no() == 0) {//center try first
193         int cx = cols / 2, cy = rows / 2;
194         if (state.get_value(cx, cy) == Sign::NONE) return {cx, cy};
```

```
195         for (int y = 0; y < rows; ++y)
196         for (int x = 0; x < cols; ++x)
197         if (is_valid_empty(state, x, y))
198         return {x, y};
199       }
200 
201       for (int y = 0; y < rows; ++y) {//checking board position
202         for (int x = 0; x < cols; ++x) {
203           if (!is_valid_empty(state, x, y)) continue;
204 
205           if (is_winning_move(state, x, y, m_sign)) return {x, y};//win
206           if (is_winning_move(state, x, y, enemy)) return {x, y};//block
     win
207           if (is_manual_winning_opportunity(state, x, y, m_sign)) return
     {x, y};
208           if (is_manual_winning_opportunity(state, x, y, enemy)) return
    {x, y};
209           if (is_open_four_threat(state, x, y, enemy)) return {x, y};//
    block _OOOO_
210           if (is_critical_threat(state, x, y, enemy)) return {x, y}; //
    block fork
211           if (is_open_four_threat(state, x, y, m_sign)) return {x, y};//
    create _OOOO_
212           if (is_critical_threat(state, x, y, m_sign)) return {x, y};//
    create fork
213         }
214       }
215 
216 
217       struct ScoredCell { int x, y, score; };// evaluate and sort top
    candidates by board score
218       ScoredCell candidates[225];
219       int count = 0;
220 
221       for (int y = 0; y < rows; ++y) {//generating all possible moves
    with heuristic scores
222         for (int x = 0; x < cols; ++x) {
223           if (is_valid_empty(state, x, y)) {
224             State next = state;
225             if (next.process_move(m_sign, x, y) == game::MoveResult::
    ERROR) continue;
226             int score = evaluate_board(next, m_sign, enemy);
227             candidates[count++] = {x, y, score};//storing move and it's
    score
228           }
229         }
230       }
231 
232       //shuffling , partial sorting(same score or not)
233       for (int i = count - 1; i > 0; --i) {
234         int j = std::rand() % (i + 1);
235         ScoredCell tmp = candidates[i];//swp
236         candidates[i] = candidates[j];
237         candidates[j] = tmp;
238       }
239 
```

```
240       //sort candidates based on the score
241       int limit = (count < 4) ? count : 4;
242       for (int i = 0; i < limit; ++i) {
243         int max_idx = i;
244         for (int j = i + 1; j < count; ++j)
245         if (candidates[j].score > candidates[max_idx].score)
246         max_idx = j;
247         ScoredCell tmp = candidates[i];
248         candidates[i] = candidates[max_idx];
249         candidates[max_idx] = tmp;
250       }
251
252       int best_score = -1000000;
253       Point best_move = {0, 0};
254
255       // cout << "top 5 possible moves \n";
256       // for (int i = 0; i < limit; ++i) {
257         //     cout << "move: (" << candidates[i].x << ", " <<
    candidates[i].y <<")"<< endl;
258         // }
259
260       for (int i = 0; i < limit; ++i) {
261         int x = candidates[i].x, y = candidates[i].y;
262         State next = state;
263         if (next.process_move(m_sign, x, y) == game::MoveResult::ERROR)
    continue;
264
265         int depth = 1; //keeping depth limited
266         int score = minimax(next, depth, -1000000, 1000000, false,
    m_sign, enemy);
267         if (score > best_score) {
268           best_score = score;
269           best_move = {x, y};
270         }
271       }
272
273       //fallback move if no good candidates found
274       if (limit == 0) {
275         for (int y = 0; y < rows; ++y)
276         for (int x = 0; x < cols; ++x)
277         if (is_valid_empty(state, x, y))
278         return {x, y};
279       }
280
281       //step 9: print best Move
282       // cout << "best move: (" << best_move.x << ", " << best_move.y <<
     ") "<< endl;
283
284
285       return best_move;
286     }
287
288     bool MyPlayer::is_open_four_threat(const State &state, int x, int y,
    Sign sign) {
289       const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
290
```

```
291       const int cols = state.get_opts().cols;
292       const int rows = state.get_opts().rows;
293
294       for (int d = 0; d < 4; ++d) {//checking all 4 directions
295         for (int i = -4; i <= 0; ++i) {
296           int count = 0, empty_count = 0;
297
298           for (int j = 0; j < 5; ++j) {
299             int nx = x + dirs[d][0] * (i + j);
300             int ny = y + dirs[d][1] * (i + j);
301
302             if (!is_in_bounds(nx, ny, cols, rows)) break;
303
304             Sign val = state.get_value(nx, ny);
305
306             if (nx == x && ny == y) val = sign; // simulate placing here
307             if (val == sign) count++;//counting empty spot
308             else if (val == Sign::NONE) empty_count++;
309             else break;
310           }
311           //_xxxx_
312           if (count == 4 && empty_count >= 2) return true;
313         }
314       }
315       return false;
316     }
317
318
319     bool MyPlayer::is_critical_threat(const State& state, int x, int y,
    Sign sign) {//returns true if the move is critical threat
320       const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};//->,top
    ,\,/
321
322       const int cols = state.get_opts().cols;
323       const int rows = state.get_opts().rows;
324
325       int threat_count = 0;
326
327       // forward threat
328       for (int d = 0; d < 4; ++d) {//checking all 4 directrions
329         int count = 1;
330         int open_ends = 0;
331
332         for (int i = 1; i < 5; ++i) {//direction
333           int nx = x + dirs[d][0]*i;
334           int ny = y + dirs[d][1]*i;
335
336           if (!is_in_bounds(nx, ny, cols, rows)) break;//stop if out of
    bounds
337           Sign val = state.get_value(nx, ny);//get sign at position(x,y)
338           if (val == sign) count++;
339           else if (val == Sign::NONE) { open_ends++; break; }//check if
    the cell is empty
340           else break;
341         }
342
```

```
343
344         //backward threat
345         for (int i = 1; i < 5; ++i) {
346           int nx = x - dirs[d][0]*i;
347           int ny = y - dirs[d][1]*i;
348
349           if (!is_in_bounds(nx, ny, cols, rows)) break;//stop if out of
    bounds
350           Sign val = state.get_value(nx, ny);//get sign at position(x,y)
351           if (val == sign) count++;
352           else if (val == Sign::NONE) { open_ends++; break; }
353           else break;
354         }
355         //_xxxx_
356         if (count == 4 && open_ends >= 1) threat_count++;//critical
    threat
357       }
358
359       return threat_count >= 2;//double threat
360     }
361
362
363   } // namespace ttt::my_player
364
```

Листинг 2. my_player.cpp

Приложение В.
Результаты тестирования

Приложение В1. Против простого BaselineEasy (500 игр X, затем 500 игр O)

```
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$ ./tests/test_stats_vs_baseline
Testing 500 times:- MyPlayer(as 1st X then 2n as O) vs baselineEasy player
MyPlayer wins: 443
BaselineEasy wins: 57
draws:  0
errors: 0

MyPlayer play time:
 - move time (ms): 3.22701
 - event time (ms): 0.00054587

BaselineEasy play time:
 - move time (ms): 0.00385298
 - event time (ms): 0.0031431

game process average time: 1.62276 (ms)
BaselineEasy wins: 110
MyPlayer wins: 390
draws:  0
errors: 0

BaselineEasy play time:
 - move time (ms): 0.00435906
 - event time (ms): 0.00316388

MyPlayer play time:
 - move time (ms): 3.62371
 - event time (ms): 0.000544824

game process average time: 1.82132 (ms)
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$
```

Рис. 3. Результаты MyPlayer против BaselineEasy

Приложение В2. Против усиленного BaselineHarder (500 игр X, затем 500 игр O)

```
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$ ./tests/test_stats_vs_baseline
Testing 500 times:- MyPlayer(1st as X then 2nd as O) vs baselineHarder player
MyPlayer wins: 198
BaselineHarder wins: 302
draws:  0
errors: 0

MyPlayer play time:
 - move time (ms): 3.68455
 - event time (ms): 0.000542271

BaselineHarder play time:
 - move time (ms): 0.00509322
 - event time (ms): 0.00273346

game process average time: 1.85146 (ms)
BaselineHarder wins: 354
MyPlayer wins: 146
draws:  0
errors: 0

BaselineHarder play time:
 - move time (ms): 0.00513062
 - event time (ms): 0.0028442

MyPlayer play time:
 - move time (ms): 3.51148
 - event time (ms): 0.000536341

game process average time: 1.76511 (ms)
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$
```

Рис. 4. Результаты MyPlayer против BaselineHarder

Приложение В3. Против MyPlayer (self-play)

```
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$ ./tests/test_stats
Testing 500 times: MyPlayer vs MyPlayer
MyPlayer wins: 231
MyPlayer wins: 263
draws:  6
errors: 0

MyPlayer play time:
 - move time (ms): 3.64476
 - event time (ms): 0.000563431

MyPlayer play time:
 - move time (ms): 3.37268
 - event time (ms): 0.000544543

game process average time: 3.51311 (ms)
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$
```

Рис. 5. MyPlayer против MyPlayer (self-play)

Приложение В4. Тестирование алгоритма

```
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$ make test
Running tests...
Test project /home/kuzon/16/tictactoe-course-master/build
    Start 1: smoke_test_player
1/4 Test #1: smoke_test_player ...............   Passed    0.09 sec
    Start 2: test_player_stats
2/4 Test #2: test_player_stats ...............   Passed   28.58 sec
    Start 3: test_againts_baseline
3/4 Test #3: test_againts_baseline ...........   Passed    0.09 sec
    Start 4: test_stats_vs_baseline
4/4 Test #4: test_stats_vs_baseline ..........   Passed  125.79 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) = 154.56 sec
kuzon@Kuzon36July:~/16/tictactoe-course-master/build$
```

Рис. 6. Пример игрового состояния во время тестирования