# Отчет по НИРС

# Тема:

# «Деревья поиска, экспертные системы и методы Монте-Карло в позиционных играх»

Выполнил студент группы ИУ7-104

Кочуркин Иван Алексеевич

Руководитель: Филиппов Михаил Владимирович

# Введение

В данной работе описываются основные проблемы, препятствующие созданию эффективного ИИ в позиционных играх, таких как «Го» и «Точки». Проведен анализ существующих методов и алгоритмов, используемых для **построения ИИ в играх «Шахматы», «Го» и «Точки»**. Предложены подходы по модификации проанализированных алгоритмов для улучшения качества игры и их применения в других позиционных играх ("Точки").

# Краткое описание игр «Го» и «Точки»

И в «Го» и в «Точках» играют два игрока, один из которых получает чёрные камни (синие точки), другой — белые (красные точки). Игроки последовательно ставят камни (точки) в узлы двухмерного поля.

Цель игры «Го» — отгородить на игровой доске камнями своего цвета большую **территорию**, чем противник.

Цель игры «Точки» — **захватить** как можно больше точек соперника.

Окружением считается любая последовательность камней (точек) соединенных горизонтальными, вертикальными или диагональными связями. При этом в Го стенки поля также учитываются при окружении, а в точках – нет. Также в Го можно захватывать вражеские камни.

Несмотря на схожие правила «Точек» с игрой «Го», стратегия в этих играх существенно различаются. В «Го» нужно захватить большее количество территории и стенки поля также могут являться окружающей территорией, поэтому постановка начальных ходов в игре (фусеки) всегда начинается с краев. В точках нужно окружить как можно большее число точек противника и стенки не являются окружением. Кроме того, для создания окружения в точках допускаются пустые позиции внутри него, в то время как в го - нет. Т.е. окружения в точках обычно имеют более глобальный характер, по сравнению с го. Таким образом, в точках не имеет смысла играть с краев, а имеет смысл создавать сразу большие глобальные контуры, в которые в будущем могут попасть точки противника, т.е. играть глобально, что и подтверждается опытом и наблюдениями (если интересно, то здесь большой архив игр сильных и не очень игроков). Т.е. в точках размер множества допустимых ходов увеличивается по сравнению с размером множества допустимых ходов в го, даже если сравнивать поля одинаковых размеров (хотя в точки играют на большем поле), а значит необходимо разработать методы мощного генерации полезных и отсечения бесполезных ходов.

# Сравнительный анализ позиционных и логических игр

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Шахматы** | **Го** | **Точки** |
| **Цель** | Поставить мат сопернику | Захват территории | Захват точек соперника |
| **Комбинаторная сложность (NP)** | Да | Да | Да |
| **Сложность правил** | Средняя | Низкая | Низкая |
| **Размер поля** | 8\*8 | 9\*9 — 19\*19 | 39\*32 |
| **Фактор ветвления** |  |  |  |
| **Возможное количество партий** |  |  |  |
| **Среднее количество ходов в партии** | 80 | 150 |  |
| **Сложность представления** | Средняя | Высокая | Высокая |
| **Аддитивная природа игры** | Нет | Да | Да |
| **Сложность**  **формализуемости правил** | Средняя | Тяжелая | Тяжелая |
| **Уровень**  **динамичности** | Высокий | Средний | Отсутствует |
| **Ко-борьба** | - | Да | Нет |
| **Уровень исследований ИИ** | Высокий | Высокий | Низкий |

## Выводы и примечания:

1. Из-за слишком большого количества начальных возможных позиций и большого фактора ветвления метод глубокого *альфа-бета поиска*неприменим для позиционных игр типа «Го» и «Точки».
2. Из-за аддитивной природы игры в «Го» и «Точках» (каждый ход прибавляется по камню), количество игровых моментов увеличивается и вариантов становится больше (в отличие от шахмат, где фигуры могут убираться).
3. Проблема представления игрового состояния в «Го» и «Точках» является нетривиальной: нужно соблюсти баланс между количеством обрабатываемой и сохраняемой информации на каждом шаге и скоростью работы.
4. Относительная статичность камней (точек) в «Го» и «Точках», позволяет людям просчитывать длинные цепочки ходов, что очень сложно для машины, в отличие от «Шахмат», «Реверси» и других.
5. На данный момент существует большое количество научных трудов и ПО для «Шахмат» и «Го», однако исследованиями игры «Точки» почти никто не занимался в силу ее малой популярности.

# Минимаксное дерево поиска

## Описание

Традиционный метод поиска, в котором рассматриваются все гипотетически возможные последовательности ходов до определённой глубины, а затем используют оценочную функцию, чтобы оценить ценность хода, с которого начиналась каждая последовательность. Ход, который приводит к наилучшему результату, повторяется на доске и далее такая же процедура проводится для каждого хода компьютерного игрока. В то время как способы, основанные на использовании дерева поиска, давали хорошие результаты применительно к шахматам, они были менее успешны в применении к го.

## Достоинства

1. Простота реализации.

## Недостатки

1. Сложность создания эффективной оценочной функции.
2. Сложность создания эффективного генератора ходов в оптимальной последовательности.
3. Большой коэффициент ветвления.
4. Сложность реализации эффективного параллельного алгоритма, особенно с использованием хеширования.

## Модификации

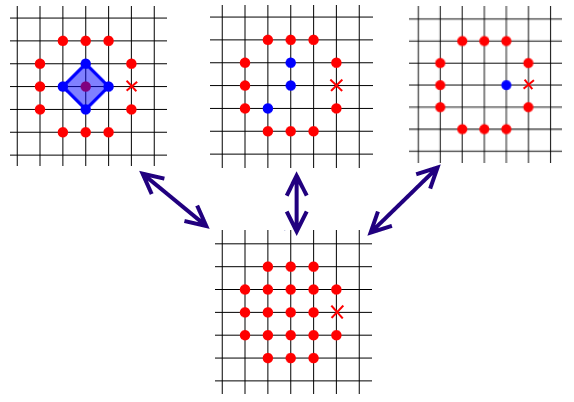
1. Alpha-Beta и его модификации:
   * Alpha-Beta с амортизацией откатов.
   * PVS. Эффективность зависит от генератора ходов.
   * Negascout: просчет первого хода с полным окном, а последующих – с окном (alpha; alpha + 1). Эффективность зависит от генератора ходов.
   * MTD-f. Эффективность сильно зависит от хеш-таблиц.
2. Использование кеширования для минимизации коллизий.
3. Распараллеливание алгоритмов:
   * С взаимоблокировками.
   * Без взаимоблокировок.

## Преимущества кэширования

1. Минимизация коллизий в игровом дереве.
2. Может быть применено не только для хеширования состояний доски, но и для хеширования более высокоуровневых примитивов.

## Разработанный метод хеширования игровых состояний дли игры «Точки»

1. Инициализация **таблицы случайных чисел HashTable** для каждого состояния клетки:  
   Известно, что в точках состояние клетки можно задать с помощью координат x, y, а также цвета точки в этой клетке. Таким образом, если игровое поле имеет размеры 39\*32, то данный массив случайных чисел будет иметь размер 39\*32\*2.
2. Инициализируется **хеш-ключ Key,** описывающий состояние игрового поля в данный момент.
3. После хода игрока
   1. Если окружений не произошло: Key = Key xor HashTable[x][y][color].
   2. Если были окружены точки соперника, то ключ вычисляется следующим образом:



Для синих точек:

Key = Key xor HashTable[x][y][blue].

Key = Key xor HashTable[x][y][red].

Для пустых позиций:

Key = Key xor HashTable[x][y][red].

Данная техника позволяет хешировать разные формально, но одинаковые по сути игровые состояния.

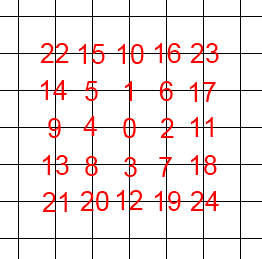
Далее ключ записывается в **таблицу перестановок** для дальнейшего использования в алгоритмах поиска.

# Экспертные системы

Данные методы связаны с использованием формализованных эвристик и правил профессионалов и любителей высокого уровня.

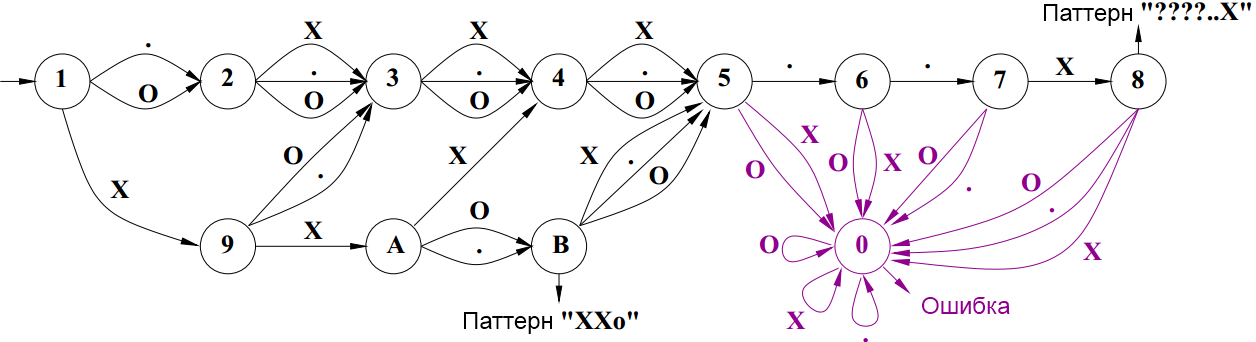
Существуют две методологии реализации таких систем:

* Сравнение с образцом (pattern matching)
* Распознавание образов (pattern recognition)

Реализуется с помощью **Конечных Автоматов**. Т.к. поле у игр «ГО» и «Точки» двухмерное, а КА работает с одномерной цепочкой, то состояния берутся из поля в определенной последовательности, указанной на рисунке слева.

При этом каждая позиция может иметь следующие состояния:

* X – поставлена точка оппонента.
* x – пустое или поставлена точка оппонента.
* O – поставлена дружеская точка.
* o – пустое или поставлена дружеская точка.
* . – пустое.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Состояние | . | O | X | # | Вывод |
| 1 | 2 | 2 | 9 | 0 |  |
| 2 | 3 | 3 | 3 | 0 |  |
| 3 | 4 | 4 | 4 | 0 |  |
| 5 | 6 | 0 | 0 | 0 |  |
| 6 | 7 | 0 | 0 | 0 |  |
| 7 | 0 | 0 | 8 | 0 |  |
| 8 | 0 | 0 | 0 | 0 | Найден паттерн “????..X” |
| 9 | 3 | 3 | A | 0 |  |
| A | B | B | 4 | 0 |  |
| B | 5 | 5 | 5 | 0 | Найден паттерн “XXo” |

## Достоинства

1. Высокая скорость работы
2. Глобальный поиск.

## Недостатки

1. Требуется привлечение экспертов в предметной области для составления базы знаний.
2. Сложность формализации экспертных знаний.
3. Требуется много времени на составление баз знаний.
4. Возможное ослабление программы, т.к. поверхностное ориентирование в ситуации может привести к ошибкам.

## Программы, использующие данный метод:

* Handtalk (Goemate), The Many Faces of Go, Go Intellect и Go++, GnuGO.

# Методы Монте-Карло

## Простой метод

Суть этого метода состоит в том, что сначала на текущей доске выбираются позиции, на которые можно пойти, а затем, начиная последовательно с каждой из них, разыгрывается большое количество случайных партий. Позиция, которая даёт наибольшее соотношение побед к поражениям, выбирается для следующего хода.

### Достоинства

* Простота реализации.
* Почти не требуются знания по предметной области для реализации алгоритма.
* Простая и эффективная реализация параллельного алгоритма.
* Стратегический поиск.

### Недостатки

* Требуются большие вычислительные ресурсы для приемлемой игры.
* Очень слабый тактический поиск.
* Большая энтропия => Большая эффективность только для полей маленького размера (9\*9).

### Программы, использующие данный метод

* The Many Faces of Go v12**,** Leela, MoGo, Crazy Stone, Olga и Gobble

## Метод UCT (Upper confidence bounds applied to trees)

Метод UCT (метод Монте-Карло, примененный к деревьям решений) является естественным расширением к обычному Монте-Карло методу. Метод UCT решает проблему выбора ходов в дереве таким образом, что важные ходы, которые чаще приводят к победе, рассматриваются чаще, чем ходы, которые с большой вероятностью приведут к проигрышу.

### Описание

1. Запускается цикл с заданным количеством итераций (входной параметр).
2. Инициализируется узел *node*, от которого просчитывается случайная партия, результатом которой является либо 0 (поражение), либо 1 (победа).
3. Обновление количества посещений и количество посещений, приведших к выигрышу, для узла *node*. Соответственно *node.Visits* и *node.Wins*.
4. Если список дочерних ходов для *node* пуст и *node.Visits* < *10*, то переход к шагу 2, в противном случае – к шагу 5.
5. Если список дочерних ходов *childs* для *node* пуст, то он создается.Для каждого *child* из списка дочерних ходов *node*  выбирается ход с наибольшим значением *Uct*, которое вычисляется с помощью следующей формулы:



1. Возврат к шагу 2.
2. Результирующий наилучший ход выбирается из всех дочерних узлов *childs* с наибольшим значением *childs.Visits.*

### Достоинства

* Несложная реализация.
* Почти не требуются знания в предметной области для реализации алгоритма.
* Возможность эффективной реализации параллельного алгоритма.
* Возможность достижения баланса между стратегическим и тактическим поиском (параметр UctK), т.е. большее количество случайных партий для ходов, приводящих к победе с большей вероятностью, чем для ходов с меньшей вероятностью.
* Положительный опыт использования модификаций данного алгоритма в играх против профессионалов

### Недостатки

* Требуются большие вычислительные ресурсы для приемлемой игры.
* Большая энтропия => Большая эффективность только для полей маленького размера (9\*9).

# Исследования и выводы

## Коллизии в хеш-таблицы в зависимости от ее размера.

На основе эксперимента было вычислено, что оптимальный размер хеш-таблицы составляет ~220 элементов. И дальнейшее увеличение размера почти не уменьшает количество просчитываемых ходов. Ходы выбирались на расстоянии менее двух позиций до всех поставленных точек.

## Сравнение UCT и метода Alpha-Beta поиска.

Метод UCT работает лучше, чем Alpha-Beta. Ходы выбирались на расстоянии менее двух позиций до всех поставленных точек. При этом основной проблемой UCT является случаи каскадных атак и защит.

# Дальнейший план работ

## Методы минимакса

* Статичная природа игры в «Точках. Просчет точек только одного цвета и нахождение предварительных alpha и beta для последующего использования их в алгоритмах минимакса (Alpha-Beta, Negascout).
* Моделирование ходов «2 атаки 1 защита»: два хода точек одного цвета, приводящих к окружению + один ход точек другого цвета, защищающий группу. Это довольно часто-встречающаяся ситуация.
* Динамическое отсечение траекторий.

## Экспертные системы

* Разработать базу шаблонов для точек (формат, атрибуты).
* Реализация распознавания паттерном с помощью КА.

## Методы Монте-Карло

* Модификация метода UCT: просчет не всей партии, а локального участка, например соединение групп.
* «Умный» выбор ходов (с помощью шаблонов).
* Исследовать метод в различных вариациях с различными параметрами.

## Общее

* Разработка метода для разбиения множества точек на непересекающиеся траектории.
* Реализовать и исследовать описанные алгоритмы в комбинации друг с другом.

# Использованные источники:

1. <http://ru.wikipedia.org/wiki/Компьютерное_го>
2. Modification of UCT with Patterns in Monte-Carlo Go (Sylvain Gelly — Yizao Wang — R´emi Munos — Olivier Teytaud).
3. Статья Павла Торгашева «Игра "Точки". Методы и алгоритмы» <http://pointsgame.narod.ru/>
4. GNU Go Documentation <http://www.gnu.org/s/gnugo/gnugo_toc.html>