NEUCHESS: An Implementation of Chinese Chess Computer Game

Xiaofei Nan

1. Introduce of Chinese chess and NEUCHESS

Chinese chess
- Large popularity
- Similar to chess
- Zero-sum game with complete information
- Uniqueness includes the movement of cannon piece, a rule prohibiting the kings from facing each other directly and so on

Contrast of two chess games
- Chess's arms: King/Queen/Rook/Bishop/Knight/Pawn/
- Chinese chess's arms: King/Rook/Horse/Cannon/Elephant/Guard/Pawn

NEUCHESS
- An implementation of Chinese computer game
- Implemented by software
- Named after my college in China
- NEUCHESS has won the championship of the Chinese chess in the 11th World Computer Olympiad.
- Beat several grandmasters of Chinese chess. The latest achievement is drawing with the No.1 grandmaster of Chinese chess, Yinchuan Xu, Aug. 2006, Beijing.
2. Why do this
History of Computer Board Game

Why research on board game?
- The precursors of artificial intelligence once indicated seriously that if one can master the essence of playing chess, he can also master the core of mankind intelligence behavior.
- The important principles which exist in playing chess, maybe also exist in any activities that need mankind intelligence.
- Computer games are a very challenge project in artificial intelligence.

With the achieved success of computer games in Othello, Checker, and Chess, scholars all over the world have paid their attentions to Chinese chess, Shogi, and Go.

So Chinese scholars will face more challenges.

Deep Blue: Computer Games in Chess
- Chess computer games have a long history
- May 1997, Deep-Blue vs. Garry Kasparov in a 6-game match
- Final score was 3.5 : 2.5

Dr. Feng-Hsiung Hsu
Garry Kasparov

Computer game history
- 2003, Kasparov draws a four-game match against X3D Fritz.
- 2005, Hydra defeats Michael Adams 5.5-0.5.
- 2005, a team of computers (Hydra, Deep Junior and Fritz), wins 8.5-3.5 against a rather strong human team formed by Veselin Topalov, Ruslan Pomaracov and Sergey Karjakin, who had an average ELO rating of 2681.
- 2006, the undisputed world champion, Vladimir Kramnik, is defeated 4-2 by Deep Fritz.

Other Computer Game
- Go
- Connect6
- Poker
3. Outline and Abstract

### Game process and system model

#### System state equation

\[
S_{n+1} = S_n \cdot q_{n+1} \\
S_F = S_0 \cdot q_1 \cdot q_2 \cdot \ldots \cdot q_F = S_0 \cdot \mathcal{Q} \\
\mathcal{Q} = \{ q_1, q_2, \ldots, q_F \} \quad \text{(Chess manual)} \\
\mathcal{Q}_{\text{red}} = \{ q_1, q_2, \ldots \} \quad \text{(Red)} \\
\mathcal{Q}_{\text{black}} = \{ q_1, q_2, \ldots \} \quad \text{(Black)}
\]

#### 4-depth game tree of Red's turn

![Game Tree Diagram]

### Contrast of some board games

<table>
<thead>
<tr>
<th>Game</th>
<th>Board Size</th>
<th>State Space Complexity</th>
<th>Game Tree Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td>3*3</td>
<td>4.3</td>
<td>5.6</td>
</tr>
<tr>
<td>Shogi</td>
<td>8*8</td>
<td>28</td>
<td>58</td>
</tr>
<tr>
<td>Chess</td>
<td>8*8</td>
<td>50</td>
<td>123</td>
</tr>
<tr>
<td>Chinese chess</td>
<td>10*9</td>
<td>52</td>
<td>150</td>
</tr>
<tr>
<td>Othello</td>
<td>19*19</td>
<td>172</td>
<td>400</td>
</tr>
</tbody>
</table>

The numbers given by the power of 10

### The thinking process playing chess achieved by computer

- Current board and state
- Extending game tree (State development)
- All feasible following boards and states
- Move generator
- All feasible move operators

\[
\{q_{i+1}\} \Rightarrow q_{i+1,j}, \ j = 1\ldots k \\
\{S_{i+1}\} \Rightarrow S_{i+1,j}, \ j = 1\ldots k
\]

### C-chess game tree is conceivable huge

- If each step has an average of 45 possible moves, each game has an average of 90 steps
- \(45^{10} \approx 10^{10}\) states should be considered from the initial state to the end of the game.
- Reputedly, the astronomical number is bigger than the number of atoms on the Earth.
- Until the destruction of the Earth, the first step’s move can not be calculated although using the fastest computer in the world.
Features of the problem

- A decision-making system of two players
- A countermeasure system
- A knowledge system
- Impossible to express and solve the problem in mathematic methods, since the state and operator can’t be expressed analytically.
- The possible way to get the solution is to use searching method, just like mankind’s thinking.
- Many key technologies have to be broken through.

4. The Key Technologies

Board and piece representation
Move generation
Evaluation function
Searching game tree
Opening book and Endgame database

Board and Piece Representation

- The state at step n can be expressed by a set of matrices as follows
  \[ S_n = \{ S^B_n, P^B_n, P^M_n, B_n, \ldots \} \]
  - \( S^B \) — Chessboard state matrix
  - \( P^B \) — Chessboard position matrix
  - \( P^M \) — Chessman position matrix
  - \( B \) — Bit-board matrix

Board Array

Initial State of chessboard

```
Board[BoardIndex[90]] =
(2, 0, 0, 7, 0, 0, -7, 0, 0, -2,
3, 0, 4, 0, 0, 0, 0, -4, 0, -3,
6, 0, 6, 0, 0, 0, -7, 0, 0, -6,
5, 0, 0, 0, 0, 0, 0, 0, 0, -5,
1, 0, 0, 7, 0, 0, -7, 0, 0, -1,
5, 0, 0, 0, 0, 0, 0, 0, 0, -5,
6, 0, 0, 7, 0, 0, -7, 0, 0, -6,
3, 0, 4, 0, 0, 0, 0, -4, 0, -3,
2, 0, 0, 7, 0, 0, -7, 0, 0, -2 )
```
Piece-coded of C-chess

Initial position of chessmen

ChessmanIndex[33]=
{0, 9, 89, 19, 79, 17, 77, 39, 19, 79, 29, 69, 6,
26, 46, 66, 86, 40, 0, 80, 10, 70, 12, 72, 30, 50, 20, 60,
3, 23, 43, 63, 83}

Bit Board Expression

\[
B = [b_{i,j}]_{i=0}^{9} \quad b_{i,j} = \begin{cases} 1 & s_{i,j} \neq 0 \\ 0 & s_{i,j} = 0 \end{cases}
\]

\[
B = \begin{bmatrix}
B_v^1 & \cdots & B_v^9 \\
B_h^1 & \cdots & B_h^9
\end{bmatrix}
\]

\(B_v^i\) (Vertical bit-vector)
\(B_h^i\) (Horizon bit-vector)

Corresponding value of bit-vector

Advantages and Disadvantages

- Accelerating regional judgments
- Easily to updated
- Most Significant Bit (MSB)
**FEN data structure**
- Used to store the information of a chessboard
- Not used in searching
- Basically a string
- Can be translated from string to board, and vice versa.

**Hash-number and Hash-transform**

<table>
<thead>
<tr>
<th>Inc.</th>
<th>King</th>
<th>Rook</th>
<th>Horse</th>
<th>Cannon</th>
<th>Elephant</th>
<th>Pawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>h1</td>
<td>h2</td>
<td>h3</td>
<td>h4</td>
<td>h5</td>
<td>h6</td>
</tr>
<tr>
<td>1</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
<td>a6</td>
</tr>
</tbody>
</table>

\[
P^M \xrightarrow{HashTransform} H
\]

Hash-number is a 64-bit random number

\[
h_k = \text{Random64}(k_M, P^M), \quad k = 1, 2, ..., 32
\]

**Chessboard Hash-number**

Get every piece’s hash-number

\[
h_k = \text{Random64}(k_M, P^M), \quad k = 1, 2, ..., 32
\]

The chessboard Hash-number by Exclusive OR

\[
H = \bigoplus_{k=1}^{32} h_k \quad \bigoplus \text{ is Exclusive OR operator}
\]

\(H\) can be used as the index of the current chessboard

*Note: There is no the inverse Hash-transform*

**Move generation**

- Move expression
- Scanning board method
- Matching template method
- Prefabricated table method

**Move expression**

- “马八进七，炮五平二”——Just a relative move expression, since it is not a complete information on move
- The complete information on move includes 4 parts
  - \(q = \{\text{from} \to \text{moved} \to \text{killed}\}\)
  - \text{from} —— The position started from;
  - \text{moved} —— The chessman moved;
  - \text{to} —— The position arrived to;
  - \text{killed} —— The chessman captured.
- The moves generated must be legal, complete and ordering.
Scanning board method

- Legal and effective area to move
- Chessboard: \( A = \{(i,j) | 0 \leq i \leq 10, 1 \leq j \leq 9\} \)
- Red side field: \( A_r = \{(i,j) | 1 \leq i \leq 10, 1 \leq j \leq 9\} \)
- Black side field: \( A_b = \{(i,j) | i \leq 5, 1 \leq j \leq 9\} \)
- Red palace: \( A_{r_p} = \{(i,j) | 1 \leq i \leq 10, 4 \leq j \leq 6\} \)
- Black palace: \( A_{b_p} = \{(i,j) | 1 \leq i \leq 3, 4 \leq j \leq 6\} \)

Where: \( A \)-area, \( R \)-red, \( B \)-black, \( P \)-palace

The problems of scanning board

- This method is very intuitive, but it is not practical.
- During the process of move generation it should scan the effective areas, restrict conditions and arrival position continuously, which will take more time.
- It can not generate the capture move and non-capture move separately.

Chessman moved from \((i,j)\)

Matching template method

- It is easy to find the position to move
- Stop if it is own piece, capture the opponent piece.
- It is suitable to Horse and Elephant

Prefabricated table

- Moves: capture move and non-capture move
- Prefabricated table is a way to accelerate move generation.
- Used in rook, cannon, horse and elephant
- Pre-generate all possible moves ahead of time, put them into memory, fetch them when needed
- For rook and cannon: Made up by 8 matrixes

\( \begin{array}{c|c|c|c|c} \text{Chessman moved from} & \text{Valid \textit{to}} & \text{Effective \textit{area}} & \text{Condition \textit{subject to}} & \text{End state} \\
\hline\hline \text{Rook} & (i,i) & (i,i) & i \leq 10 & \text{x} \\
\hline \text{Horse} & (i,i) & (i-1,i) & j \leq 9 & \text{x} \\
\hline \text{Horse} & (i,i) & (i+1,i) & j \leq 9 & \text{x} \\
\hline \text{Horse} & (i,i) & (i,i+1) & j \leq 9 & \text{x} \\
\end{array} \)

Stop if it is own piece, capture the opponent piece

\( \begin{array}{c|c|c|c} \text{Horse template} \\
\hline \text{Horse} & \text{x} & \text{x} & \text{x} \\
\hline \text{Horse} & \text{x} & \text{X} & \text{X} \\
\hline \text{Horse} & \text{x} & \text{x} & \text{x} \\
\end{array} \)

- \( \text{Hor}_\text{rook}_\text{attack}[90][512] \)
- \( \text{Ver}_\text{rook}_\text{attack}[90][1024] \)
- \( \text{Hor}_\text{rook}_\text{unattack}[90][512] \)
- \( \text{Ver}_\text{rook}_\text{unattack}[90][1024] \)
- \( \text{Hor}_\text{cannon}_\text{attack}[90][512] \)
- \( \text{Ver}_\text{cannon}_\text{attack}[90][1024] \)
- \( \text{Hor}_\text{cannon}_\text{unattack}[90][512] \)
- \( \text{Ver}_\text{cannon}_\text{unattack}[90][1024] \)
Prefabricated table method

Example for Cannon moves table
Starting position is \((i, 6)\)

- Row bit-vector of piece distribution \([101000010]\)
- Non capture arrival row bit-vector \([000110100]\)
- Capture arrival row bit-vector \([100000000]\)

Folded Board

- Assist with prefabricated table for horse and elephant
- Horse_leg_table[from]
- Horse_table[from][Horse_leg_table[from] & ALLPIECE]
- 90*20*96 too much, need simplification
- Re-index the board by “folding the board”
  - Horse_table[from][Folded(Horse_leg_table[from] & ALLPIECE)]

Code of folding process

```
inline int BitBoard::Folded()
{
    unsigned __int32 Temp32;
    unsigned __int16 Temp16;
    Temp32 = m_data[0] ^ m_data[1] ^ m_data[2];
    Temp16 = ((unsigned __int16 *) &Temp32)[0] ^ ((unsigned __int16 *) &Temp32)[1];
    return (int) (((unsigned __int8 *) &Temp16)[0] ^ ((unsigned __int8 *) &Temp16)[1]);
}
```

Compression: 96 bit board \rightarrow 8 bit board

If we choose another index way?

Evaluation function

- Chess evaluation is to mark the chess state.
- One can hardly checkmate the opponent in some steps.
- Then how to judge the state at this situation, the feasible method is to quantify the chess state, judge it by scores.
- Abundant knowledge of C-chess is necessary
- So the design of evaluation function becomes the most humanistic part in the computer game.
The evaluation of the piece value

\(-e_1(m)\)

- Rook - 600,
- Cannon - 285,
- Elephant - 120,
- Horse - 270,
- Pawn - 20
- King should be infinity relatively
  for example 6000

The evaluation of piece position

\(-e_2(m,i,j) (m=rook)\)

\[\begin{array}{cccccccccccc}
14 & 14 & 12 & 18 & 16 & 18 & 12 & 14 & 14 \\
16 & 20 & 18 & 24 & 26 & 24 & 18 & 20 & 16 \\
12 & 12 & 18 & 18 & 12 & 12 & 12 & 12 & 12 \\
12 & 18 & 16 & 22 & 22 & 22 & 16 & 18 & 12 \\
12 & 14 & 12 & 18 & 18 & 12 & 14 & 12 & 12 \\
12 & 16 & 14 & 20 & 20 & 20 & 14 & 16 & 12 \\
6 & 10 & 8 & 14 & 14 & 14 & 8 & 10 & 6 \\
4 & 8 & 6 & 14 & 12 & 14 & 6 & 8 & 4 \\
8 & 4 & 8 & 16 & 16 & 8 & 16 & 8 & 4 \\
-2 & 10 & 6 & 14 & 12 & 14 & 6 & 10 & -2 \\
\end{array}\]

The evaluation of piece position

\(-e_2(m,i,j) (m=pawn)\)

\[\begin{array}{cccccccccc}
0 & 3 & 6 & 9 & 12 & 9 & 6 & 3 & 0 \\
18 & 36 & 36 & 80 & 120 & 80 & 56 & 36 & 18 \\
14 & 26 & 42 & 60 & 80 & 60 & 42 & 26 & 14 \\
10 & 20 & 30 & 40 & 40 & 30 & 20 & 10 & 10 \\
6 & 12 & 18 & 20 & 18 & 18 & 12 & 6 & 6 \\
2 & 0 & 8 & 0 & 8 & 0 & 8 & 0 & 2 \\
0 & 0 & -2 & 0 & 4 & 0 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]

Other factors for evaluation

- \(e_3\) — The evaluation of piece agility
  relating to the number of possible moves of a piece;
  \(e_3\) = the number * a score for every piece
  pawn 15, guard 1, elephant 1, rook 6, horse 12, cannon 6,
  king 0
- \(e_4\) — The evaluation of king safety ……
  give a score for each potential dangerous state (possible
  check or checkmate)
- \(e_5\) — The evaluation of piece cooperation
  e.g. A piece is under the protection of another one:
  Two cannons have more power than just doubling one

Formula of evaluation function

\(e_1, e_2, e_3, e_4, \ldots\)

\[E^R = \sum_{k=1}^{N} E^R_{k} = \sum_{l=1}^{T} \sum_{k=1}^{N} E^R_{k,l}\]

\[E^B = \sum_{k=1}^{N} E^B_{k} = \sum_{l=1}^{T} \sum_{k=1}^{N} E^B_{k,l}\]

\[E = E^R - E^B\]

Searching game tree

The goal is to find
the principal continuation
and the best move
There are many search methods

• Breadth-first search
• Depth-first search
• Iterative search
• cut-off and pruning
• ……

The founder of computer game


• Proposed the Max-Min search method
• The red side always takes the maximum of the sub-node’s values
• The black side always takes the minimum of the sub-node’s values

MaxMin search (1)

The principal continuation and the best move

MaxMin search (2)

The principal continuation and the best move

Pseudo code of MaxMin

```c
int MinMax(int depth)
{
    if (SideToMove() == WHITE) { //
        return Max(depth);
    } else {
        return Min(depth);
    }
}

int Max(int depth)
{
    int best = -INFINITY;
    if (depth <= 0) {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = Min(depth - 1);
        UnmakeMove();
        if (val > best) { //
            best = val;
        }
    }
    return best;
}

int Min(int depth)
{
    int best = INFINITY;
    if (depth <= 0) {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = Max(depth - 1);
        UnmakeMove();
        if (val < best) { //
            best = val;
        }
    }
    return best;
}
```

NegaMax algorithm

• Max-Min algorithm undoubtedly brings mass of trouble into algorithm realization.
• Knuth and Moore proposed the significant NegaMax algorithm as follows.

\[
F(v) = \max \left\{ -F(v_1), -F(v_2), \cdots -F(v_n) \right\}
\]

If the leaf nodes are red side: RedValue-BlackValue
If the leaf nodes are black side: BlackValue-RedValue
Very useful for programming implementation and search speed improving.
Alpha-Beta Search
• A pruning searching strategy without risk
• Alpha: the best one I already have.
  if value <= Alpha, no use
• Beta: the worst one for the opponent (the upper bound that the opponent could bear/accept)
  if value >= Beta, no use, the opponent could avoid this.
Find Alpha < value < Beta

\[ \text{Pseudo Code of NegaMax} \]

\[
\begin{array}{l}
\text{int NegaMax(int depth)}
\text{\{ int best = -INFINITY; if (depth <= 0) return Evaluate(); }
\text{ \} }
\text{GenerateLegalMoves(); while (MovesLeft()) }
\text{ \{ MakeNextMove(); val = -NegaMax(depth - 1); if (val > best) }
\text{ \{ best = val; \} \} return best; } \\
\end{array}
\]

\[ \alpha \text{ cut-off (1)} \]
Depth-first search
The principal continuation and the best move

[Diagram showing a cut-off effect]

\[ \alpha \text{ cut-off (2)} \]
The principal continuation and the best move

[Diagram showing another cut-off effect]

Contrast of cut-off effects
• Cut-off effect is quite different
• Closed relation with the best move emerging
• The key is to start from the best move
Efficiency analysis of Alpha-Beta Search

- Improved by Knuth and Moore in 1975
- The efficiency is related to the order of the moves.
- Best: Minimum tree
  \[ N_B = 2B^{D+1} - 1 \]
  D is even
  \[ N_B = 2B^{D+1/2} - 1 \]
  D is odd
- Worst: Maximum tree
  \[ N_D = B^D \]
- N_B is the number of searched nodes, D is searching depth, B is the Branching Factor

Aspiration Window

- [Alpha Beta] a window
- More narrow, more cutoff
- guess a X. Let Alpha=X-window, Beta=X+window, window is the width of the Window
- If value (X-window, X+window), OK
- If value=X+window, (value, INFINITY)
- If value<=X-window, (-INFINITY, value)

PVS (Principal Variation Search)/Negascout

- Three kinds of nodes:
  - (a) Alpha node: all the children less than Alpha
  - (b) Beta node: at least one child great than Beta
  - (c) PV node: at least one child great than Alpha, but no one greater than Beta
- Cutoff risk of the three nodes:
  - (1) Beta node: no risk, cut at once
  - (2) Alpha node and PV node: risk, because probably find another better node.

Pseudo Code of AlphaBeta

```c
int AlphaBeta(int depth, int alpha, int beta) {
    if (depth == 0) {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (val >= beta) {// beta cutoff
            return beta;
        }
        if (val > alpha) {
            alpha = val;
        }
    }
    return alpha;
}
```

Pseudo code of PVS

```c
int AlphaBeta(int depth, int alpha, int beta) {
    BOOL fFoundPv = FALSE;
    if (depth == 0) {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        if (fFoundPv) {
            val = -AlphaBeta(depth - 1, -alpha - 1, -alpha);
        } else {
            val = -AlphaBeta(depth - 1, -beta, -alpha);
        }
        UnmakeMove();
        if (val > alpha) {
            alpha = val;
        } else {
            return alpha;
        }
    }
    return alpha;
}
```
Iterative Deepening

- How many plies do we need? Same time, more plies will be searched in end period of a match than in the middle period.
- Iterative deepening ensures full use of time
- The result of depth-N search will be the first node searched in depth-N+1

Complexity Analysis

- \( N \) is the number of searched nodes, \( D \) is searching depth, \( B \) is the Branching Factor (45)
- \( N = B^0 + 2B^1 + 3B^2 + \ldots + BD \)
- Time complexity is \( O(B^D) \)

Pseudo Code of Iterative Deepening

```c
for (depth = 1; ; depth++) {
    val = AlphaBeta(depth, Alpha, Beta);
    if (TimedOut()) {
        break;
    }
}
```

Null Move

- NullMove Chrilly Donninger 1993
- “If I do nothing here, can the opponent do anything?”
- In regular case, you are asking about the best way to hurt the opponent. Now, asking if the opponent can hurt you.
- Like a fighter expressing heavy confidence in his ability, by giving the opponent a free shot. If the opponent can’t knock the fighter down with a free shot, chances are he’s going to lose the fight if the fighter goes to the trouble of hitting him.
- You give the opponent a free shot at you, and if you are still good, that you exceed Beta.

Pseudo code of Null Move

```c
#define R 1

int AlphaBeta(int depth, int alpha, int beta) {
    if (depth == 0) {
        return Evaluate();
    }
    MakeNullMove();
    int val = -AlphaBeta(depth - 1 - R, -alpha, -beta + 1);
    UnmakeNullMove();
    if (val >= beta) {
        return beta;
    }
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        if (val < beta) { // If the opponent can’t hurt you, you are good.
            val = -AlphaBeta(depth - 1, -beta, -alpha);
            UnmakeMove();
            if (val >= beta) { // If the opponent can’t hurt you, you are good.
                return beta;
            }
        }
        if (val > alpha) {
            alpha = val;
        }
    }
    return alpha;
}
```

Null move can’t be used in:

- When the king is in check
- In the end game
- Can not use two null move continuously
Search algorithm is hot research

- Many excellent search algorithms emerged based on move ordering and finding the pseudo best move.
- Such as Transposition table, Killer heuristic, Method of analogies, Deep α-β cut-off, Fail-Soft Alpha-Beta, Aspiration Search, Minimal Window Search, Principal Variable Search, Negascout, Tolerance search and so on.
- Iterative deepening search is a very popular search algorithm.

Opening book

- Chess playing process includes 3 phases
  - Opening phase
  - Middle phase
  - Endgame phase
- The task of opening phase is to move the major pieces quickly and occupy the good position.
- In order to avoid the strategic mistake the game programs mostly use the opening book.
- It is selects the very mature opening moves from the many many chess manuals.

Endgame database

- The major chessmen removed and just a few pieces left on chessboard, it is the endgame.
- There are a lot of chess manuals to deal with the different endgames.
- It is difficult to put the manuals into the game program.
- So game program is in inferior position under endgame phase.
- It is a challenge job to set up the endgame database in game program.

4. The Challenges in Computer Games

- Shogi and Go have a long way to go
- Development of Computer Game Theory
- Application of Computer Game Theory, such as commercial negotiation, the disposal of all kinds of abrupt incident (terror, disaster), the diplomacy of the nation
- Principle of Computer Game Theory is the important part of War Simulation Theory

- The game problem is very pervasive.
- As long as some conflict of interest exists between the two parties in the “chessboard”, game becomes a method which the conflict expresses and gets solution.
- Game and countermeasure will become a kind of hot problems on intelligent system research.
The end

Thank you