Programming Pentago with Matlab

Gustavo C. PINTON and Marcelo M. F. DE CARVALHO

CentraleSupélec

2015

Contents

Introduction

The Objective

- To implement an artificial intelligence able to play effectively the board game Pentago.
- To use Matlab to achieve so
  - Most are developed in C++... BUT Matlab has a pedagogical value.
    - Previously used at Supélec's EL.
    - Previously used by our advisor to implement an AI

Introduction

What is Pentago?

- Two-player game introduced by Tomas Flodén in 2005
- How do we play it?
  - We play it using a 6x6 board divided into four 3x3 sub-boards (or quadrants).
  - Each player assumes a color (a white player, a black player). Taking turns, each one places a marble of his color onto an unoccupied space on the board, and then rotate one of the sub-boards by 90 degrees, either clockwise or anti-clockwise.
  - A player wins by getting five of their marbles in a vertical, horizontal or diagonal row.
  - If all 36 spaces on the board are occupied without a row of five being formed then the game is a draw.

Introduction

Why Pentago?

- Two player, deterministic, perfect knowledge, zero sum game.
  - Interesting for computer science:
    - AI development and theoretical complexity analysis.
  - Great popularity worldwide:
    - Pentago: smaller number of state and some symmetries.
    - Less complex.
- Pentago has been strongly solved with a Cray supercomputer at NERSC.
  - How far can we go using Matlab and a regular laptop?
  - Would it be suitable for Human x Machine and Machine x Machine battles?

Introduction

Work Planning

- First, Pentago structure (i.e. player vs player enabled).
  - Graphical interface.
  - Allow piece placing in an empty space and turn structure.
  - Introduce rotation.
  - Introduction of a «game over» clause (win or tie).
- Second, artificial intelligence (i.e. player vs AI enabled)
  - Random play AI.
  - Research about possible solving algorithms.
- Finally, AI optimizing (i.e. AI vs AI enabled)
  - Implementation of efficient solving algorithms.
  - Further tuning for optimal results.

Graphical Interface - GUI

Graphical components

- Graphical components provided by MATLAB:
  - **figure(. . .)**: window.
  - **uicontrol(. . .)**: two buttons, *Start Game* et *Reset Game*, and textarea.
  - **axes(. . .)**: area .
  - **rectangle(. . .)**: each one of the 36 circles.
  - **image(. . .)**: 8 arrows.
  - **line(. . .)**: two dotted red lines.

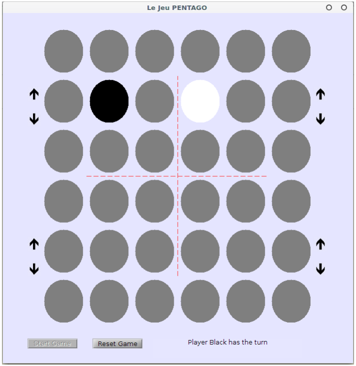

Figure 1: The game's interface.

Graphical Interface - GUI

Positioning of the components

- Parameters of function **figure**:
  - **Visible**: determines whether the window will be visible or not.
  - **Position**: an array containing the coordinates x and y of the screen, width and height of the window, respectively.
  - **Name**: the string that will be displayed in the figure window.
  - **Number Title**: this property set to *off* means that the string *Figure No* will not be displayed in the figure window.
  - **Resize**: set to *off* means that the window cannot be resized.
  - **Color**: sets the background color of the figure. The array contains three values, which determine the color according to the (**R**ed, **G**reen, **B**lue) scale.
- Example:

```
plateau = figure ('Visible', 'on',
        'Position', [1500, 1000, width, height],
        'Name','Le Jeu PENTAGO',
        'NumberTitle','off', 'Resize', 'off',
        'Color', [0.9, 0.9, 1], 'MenuBar', 'none');
```

Graphical Interface - GUI

Positioning of the components

- Parameters of function **uicontrol**:
  - **Callback**: It specifies a function handler, which will be activated when the button is pressed.

```
buttonStart = uicontrol ('Position', [50 30 100 20],
     'String', 'Start Game','Callback',
    @start_pressed);
```

- Parameters of the function **image**:
  - **ButtonDownFcn**: equivalent to the property **Callback** detailed previously.
  - Function **imread(...)**: reads an image from specified file.

```
rotate_up = imread('arrow_alt_up.jpeg');
textQ1R   = image(-30,400,rotate_up, 'ButtonDownFcn'
    ,{@turn_pressed, 1, 'L'});
```

Graphical Interface - GUI

Positioning of the components

- Parameters of the function **axes**:
  - **Color**: specify the color of the axes back planes. Setting it to *none* means that the axes is transparent and the figure color shows through
  - **Visible**: component's visibility. Setting it to *off* prevents axis lines, tick marks, and labels from being displayed.

```
1  axes1 = axes([...],'
       Color', 'none', '
       Visible', 'off');
```

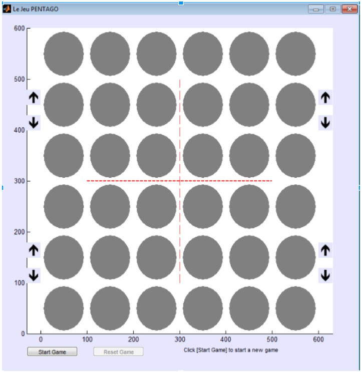

Figure 2: *Axis1* with *Color* and *Visible* different from *none* and *off*

Graphical Interface - GUI

Evaluating a board

- Les variables utilisées:
  - **value**: the result that will be given to that board.
  - Each element of **hasBeenDetected** is an integer whose representation in binary determines if that element has already been detected taking place in at least one line/column or diagonal.
  - **playerScores**: an array that stores the scores of all combination the player's got so far in each one of the possible configurations.
  - **opponentScores**: is the same that *playerScores* but for the opponent.
  - **foundUltraCondition** is true if we detect a line, column or diagonal completely filled with 5 pieces.

Graphical Interface - GUI

Evaluating a board

- Example, checking the lines for a row:

```
while (offset_index >= 1) && (bitand(hasBeenDetected(x,y), 2) == 0) && ...
        (foundUltraCondition == 0)

    if (y + offset_index) <= 6
        t = state_matrix (x , y:(y + offset_index));

        if all(t == t(1))

            if offset_index == 4
                foundUltraCondition = 1;
            end

            scores(2, index_player) = scores(2, index_player) + 10^(3*(offset_index+1));

            hasBeenDetected (x , y:(y + offset_index)) = ...
                hasBeenDetected (x , y:(y + offset_index)) + 2;

        elseif offset_index == 4

            if (sum (t == player) == 4) && (sum (t == opponent) == 1)
                value = value + 10^14;
            elseif (sum (t == player) == 1) && (sum (t == opponent) == 4)
                value = value - 10^14;
            end

        end
    end
    offset_index = offset_index - 1;
end
```

PENTAGO's algorithm

The choice of the algorithm

- Computational resources constraints: processing time and memory.
  - Search algorithms with acceptable complexity instead of full search.
    - Ex.: Alpha-beta, Proof-number, Threat-space, Retrograde.
    - Most employed and our choice: Alpha-beta search
- The Alpha-beta search:
  - Based on the minimax algorithm.
  - Two players evaluate a function, one of them tries to maximize and to the other to minimize it.
    - The value that will be achieve is: $\max\{\min\{\max\{\min\ldots\max\{\min\{eval\}\}\}\}\}$
  - Pruning:
    - If a branch cannot change the result, we skip its whole subtree.

PENTAGO's algorithm

Algorithm and complexity

- Algorithm description:
  1. Consider a node n somewhere in the tree, such that one can move to that node.
  2. If there is a better choice m either at the parent of the node n or at any choice point further up, n will never be reached.
  3. Once we have enough information about n to reach this conclusion, we can prune it.
- Complexity, by the arrange of the nodes (branching factor b, search depth d):
  - Pessimal: $\mathcal{O}(bd)$ - the same as the simple minimax.
  - Optimal: $\mathcal{O}(\frac{bd}{2}) = \mathcal{O}(\sqrt{bd})$.
  - Random: $\mathcal{O}(b\frac{3d}{4})$.

PENTAGO's algorithm

The pruning. . . Alpha? Beta?

- Alpha-Beta pruning gets its name from the following parameters:
  1. α = the value of the best choice we have found so far at any choice point along the path for MAX.
  2. β = the value of the best choice we have found so far at any choice point along the path for MIN.
- Alpha-Beta search:
  - Updates the values of α and β as it goes along.
  - Prunes the remaining branches at a node
    - As soon as the value of the current node is known to be worse than the current values of α and β.
    - Therefore, it does not affect the solution.

PENTAGO's algorithm

Demonstration - Video

- Minimax - Alpha Beta Pruning (Artificial Intelligence) by Ice Blended.
  Available at: `https://youtu.be/SROIGH1P2No?t=104`. Start watching at 1:45

PENTAGO's algorithm

Coding (1)

- The function will return an array. This array will contain:
  - **eval_value**: the evaluation of the board.
  - **move**: the place (convention [line column]) where the next piece will be played
  - **quadrant**: The quadrant where the rotation is applied after the piece be placed.
  - **direction**: The direction of the rotation.
- Example:

```
function [eval_value move quadrant direction] =
    alpha_beta_search (state_matrix, emptySlots, depth,
    alpha, beta, isMaximizing, player)
```

PENTAGO's algorithm

Coding (2) - Variables and Recursion

- Variables used in the search:
  - **alpha, beta**: Parameters of the Alpha-Beta search, said previously.
  - **depth**: The depth left by the search. Starts at 2 goes to 0.
  - **mustStop**: allows us to do the pruning. Do so by breaking the while loops that are used to explore our tree, thus skipping the subtree.
- Recursive function:
  - Base case: maximum depth reached or no possible plays.

```
if depth == 0 || emptySlots == 0
    [eval_value] = evaluate_board(state_matrix,
        player);
    move = []; quadrant = 0; direction = 0;
```

  - Else: explore subtree.

```
else
    dir = ['L' 'R'];  mustStop = 0;
    if (isMaximizing == 1) eval_ = -inf;
    else eval_ = +inf;
    end
```

PENTAGO's algorithm

Coding (3) - A cascade of loops

- A cascade of while loops to emule a tree structure. The exploration stops when we arrive to:
  - Natural limits of the 6x6 board.
  - Pruned subtrees.
- In our case:

```
l = 1;
while (l <= 6) && (mustStop == 0)
    c = 1;
    while (c <= 6) && (mustStop == 0)
        if state_matrix (l,c) == 0
            quad_index = 1;
            while (quad_index <=4) && (mustStop == 0)
                dir_index = 1;
                while (dir_index <= 2) && (mustStop == 0)
                    if isMaximizing == 1
                        state_matrix (l,c) = player;
                    else
                        if player == 'B'
                            state_matrix (l,c) = 'W';
                        else
                            state_matrix (l,c) = 'B';
                        end
                    end
                    state_matrix = rotate_quadrant (state_matrix, quad_index,
                         dir (dir_index) );
```

PENTAGO's algorithm

Coding (4) - Alpha-beta search coded

- Implementation of the alpha-beta search algorithm.
  - Started as a pseudo code.

```
state_game_ = verify_victory(state_matrix);
if isMaximizing == 1
    if state_game_ == player
        eval_MIN = +10^20;
        mustStop = 1;
    else
        [eval_MIN move_MIN quadrant_MIN
              direction_MIN] = ...
            alpha_beta_search(state_matrix,
                 emptySlots - 1, depth - 1,
                 alpha, beta, 0, player);
    end
    if eval_ < eval_MIN
        eval_ = eval_MIN;
        eval_value = eval_;
        move = [l c];
        quadrant = quad_index;
        direction = dir (dir_index);

    end
    if eval_ >= beta
        mustStop = 1;
    end
    alpha = max(alpha, eval_)
```

```
elseif isMaximizing == 0
    if (state_game_ ~= player) && (state_game_
        ~= 0) && (state_game_ ~= 'D')
        eval_MAX = -10^20;
        mustStop = 1;
    else
        [eval_MAX move_MAX quadrant_MAX
              direction_MAX] = ...
            alpha_beta_search(state_matrix,
                 emptySlots - 1, depth - 1,
                 alpha, beta, 1, player);
    end
    if eval_ > eval_MAX
        eval_ = eval_MAX;
        eval_value = eval_;
        move = [l c];
        quadrant = quad_index;
        direction = dir (dir_index);
        end
    if alpha >= eval_
        mustStop = 1;
    end
    beta  = min(beta, eval_);
```

PENTAGO's algorithm

Coding (5) - End of an iteration

- Restore the state of our board(state_matrix) to how it was before doing the play that we are analysing
  - Achieved by doing the reverse of all modifications that we have applied since we ran the alpha beta search on the board.
- Update loop parameters and start another iteration.

```
        if dir (dir_index) == 'L'
                state_matrix = rotate_quadrant(
                    state_matrix, quad_index, 'R');
        else
                state_matrix = rotate_quadrant(
                    state_matrix, quad_index, 'L');
        end

        state_matrix (l,c) = 0;
        dir_index = dir_index + 1;
end
quad_index = quad_index + 1;
end
```

Battle of AIs - A statistical point of view

Simulation

- Player **Black** represents your artificial intelligence.
- Player **White** represents ours.
- 500 matches simulated:
  - Each AI has the first move in exactly half of them, that is, 250 matches.
  - 193 (90 of them when it started playing) were won by player **White**.
  - 223 (116 of them when it started playing) were won by player **Black**.
  - 84 (44 when **Black** started) resulted in draws.

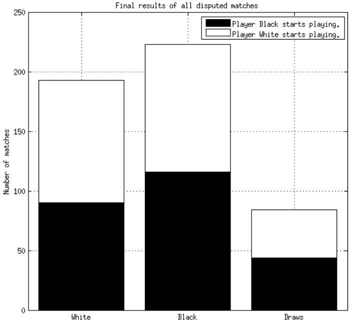

Figure 3: Number of victories of each player.

Battle of AIs - A statistical point of view

Some probabilities

- Based on the previous results, we can observe that:
  - For player **Black**:
    $\mathrm{P}(\mathbf{B}) = \frac{223}{500} = 0.446$
    $\mathrm{P}(\mathbf{B} \mid \mathbf{B}_{started}) = \frac{116}{250} = 0.464$
    $\mathrm{P}(\mathbf{B} \mid \mathbf{W}_{started}) = \frac{107}{250} = 0.428$
  - For player **White**:
    $\mathrm{P}(\mathbf{W}) = \frac{193}{500} = 0.386$
    $\mathrm{P}(\mathbf{W} \mid \mathbf{B}_{started}) = \frac{90}{250} = 0.360$
    $\mathrm{P}(\mathbf{W} \mid \mathbf{W}_{started}) = \frac{103}{250} = 0.412$
  - In case of **match nul**:
    $\mathrm{P}(\mathbf{M}_{draw}) = \frac{84}{500} = 0.168$
    $\mathrm{P}(\mathbf{M}_{draw} \mid \mathbf{B}_{started}) = \frac{44}{250} = 0.176$
    $\mathrm{P}(\mathbf{M}_{draw} \mid \mathbf{W}_{started}) = \frac{40}{250} = 0.160$

Battle of AIs - A statistical point of view

Decision time for each player

- Player **Black** takes, in average, **2.05** seconds to decide.

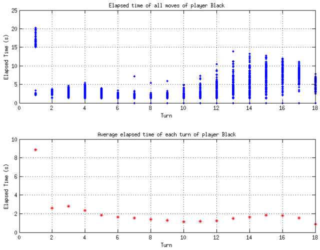

Figure 4: Required time to Player **Black** take a move.

- Player **White** takes, in average, **9.5** seconds to decide.

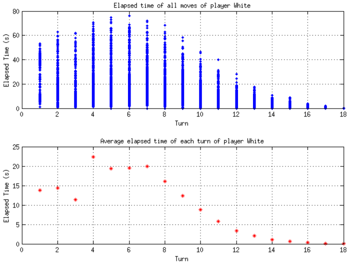

Figure 5: Required time to Player **White** take a move.

Battle of AIs - A statistical point of view

Comparison of algorithms

- The difference between these two times is explained by how which algorithm decides the move and evaluates a board.
  - Our algorithm needs to verify each position of the matrix at least twice: once to evaluate if that **position belongs to a row, column or diagonal**, and another to conclude if that **move is profitable**.
- The **first eight moves** of player **White** take almost **all the time of decision**.
  That's already expected: at the beginning of the match, there are more positions to be verified, contrarily to the end, in which few available positions remain.
- Player **Black**'s moves takes roughly constant time, around 2 seconds, to be decided. The only move that does not follow that property is the first one.

Conclusion

- Obtained results:
  - Graphical interface: Easy to use, functional and enjoyable.
  - AI:
    - Human vs Human, AI vs AI and mixed matches enabled.
    - Good performance: Comparable with the one of our advisor.
- Room for improvement (some of them may be beyond a 2 month project!):
  - Programming Language: C++ instead of Matlab.
  - Algorithm: Transition tables, deeper search.
  - Hardware: GPU processing, supercomputers.
- Aftermatch
  - The next generation of students may take from where we left and pursuit one of the lines of improvement.
  - Or simply have our algorithm as a reference, as we had Mr. Bourgois'.

Bibliography (1)

- Pentago, (2015). Pentago. [online] Available at: http://en.wikipedia.org/wiki/Pentago [Accessed 14 Apr. 2015].
- Pentago is a first player win, (2015). [online] Available at: https://perfect-pentago.net/details.html#intro [Accessed 14 Apr. 2015].
- On solving Pentago, (2015). [online] Available at: http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2011/Buescher_Niklas.pdf [Accessed 14 Apr. 2015].
- Standford University, (2015). [online] Available at: http://web.stanford.edu/~msirota/soco/alphabeta.html [Accessed 14 Apr. 2015].
- Élagage alpha-beta, (2015). [online] Available at: http://fr.wikipedia.org/wiki/%C3%89lagage_alpha-beta [Accessed 14 Apr. 2015].

Bibliography (2)

- Chalmers - GÖTEBORGS UNIVERSITET, (2015). [online] Available at: `http://www.cse.chalmers.se/edu/year/2014/course/TIN172/example-exam-solutions.html` [Accessed 14 Apr. 2015].
- McCarthy, 2006. McCarthy, John (LaTeX2HTML 27 November 2006). "Human Level AI Is Harder Than It Seemed in 1955". `http://www-formal.stanford.edu/jmc/slides/wrong/wrong-sli/wrong-sli.html` [Acessed 20 Dec. 2006]
- Russell, Stuart J., 2003. Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2 `http://aima.cs.berkeley.edu/`
- Nersc, (2015). [online] Available at: `https://www.nersc.gov/users/computational-systems/edison` [Accessed 19 May 2015].
- NVIDIA, (2015). [online] Available at: `https://developer.nvidia.com/how-to-cuda-c-cpp` [Accessed 19 May 2015].