Pentago Documentation

David Wu and Jun Kang Chin

2006.12.06

Contents

1 Game Introduction

Pentago is a two player non-loopy partizan game. The standard board is a 6 by 6 checkers board that is divided into 4 quadrants of size 3 by 3. Each player will place his or her piece on the board and choose a quadrant to rotate 45° clockwise or counter-clockwise. The goal of the game is to be the first player to connect 4 in a row. A tie occurs when the board is filled and neither player gets 4 pieces in a row.

2 Design Decisions

Pentago is completely modular, so all functions should readily adapt to changes made in the ROWSIZE and NthInARow variable values. The board is implemented using a 1D-array starting at 0, where each index in the array modulo ROWSIZE returns the column number, and the floor of the index divided by ROWSIZE returns the row number. As Pentago is a dart board game, Generic Hash is used to maintain board information. Blank pieces are represented using spaces, player 1 is an x, and player 2 is an o.

3 Hash Function

Pentago uses Generic Hash. For more information, see Generic Hash API.

4 Move Semantics

A move is represented as a 3-digit signed number such that the hundred's place is the column, the ten's place is the row, the one's place is the board to rotate, and the sign of the number determines whether the board is rotated clockwise or counter-clockwise. In this implementation, a positive number refers to a clockwise rotation, whereas a negative number represents a counter-clockwise rotation.

Mathematically speaking,

- $0 \leq$ hundred's place $<$ ROWSIZE
- $0 \leq$ ten's place $<$ ROWSIZE
- $0 \leq$ board to rotate ≤ 4 (because there are only 4 subboards)

ex) -451 would represent a move where a piece is placed at (4 + (5 * ROWSIZE)) on the board and subboard 1 is rotated counter-clockwise.

5 Functions

5.1 Main Function Implementations

5.1.1 `InitializeGame()`

`void InitializeGame()`
Calls InitializeBoard and initializes Generic Hash.

5.1.2 `*GenerateMoves()`

`MOVELIST *GenerateMoves(POSITION position)`
Every empty spot is a potential move with 4 different subboard rotation possibilities and 2 different rotation directions. This function simply goes through the board and creates moves at blank spaces.

5.1.3 `DoMove()`

`POSITION DoMove(POSITION position, MOVE move)`
The move is parsed and a piece is placed on the board. Then, rotateBoard is called and the subboard is rotated.

5.1.4 `rotateBoard()`

`void rotateBoard(char* currBOARD, int boardToRotate, BOOLEAN CW)`

- currBOARD is the board
- boardToRotate is the subboard to rotate
- CW is true when the subboard should be rotated clockwise and false in the counter-clockwise direction.

`rotateBoard` finds the corner indexes of the subboard provided in the argument and iterates through by rotating the outer ring and then stepping into more and more internal rings. numOfSteps represents the number of iterations needed to complete this task.

5.1.5 `Primitive()`

`VALUE Primitive(POSITION position)`
This function is very primitive, as it checks in all directions at each index to see whether or not a game has ended. It does so by first seeing if the index NthInARow - 1 spots away is on the board, and then if it is, the function steps across each piece to determine the outcome of the board.

note: Currently, something is wrong with gStandardGame return values. It could be the way it is being used.

5.1.6 `StepAndCheck()`

```
int StepAndCheck(int location, int horizontalStep, int verticalStep,
char * currBOARD)
```

- location is the current index in the board array
- horizontalStep either -1, 0, or 1 depending on the displacement needed to check the next piece
- verticalStep is similar to horizontalStep.
- currBOARD is the board.

This function is called by Primitives to interatively check to see if there are NthInARow pieces of the same type in a row.

5.1.7 `canGoLeft(), canGoRight(), canGoUp(), canGoDown()`

```
BOOLEAN canGoLeft(int location)
BOOLEAN canGoRight(int location)
BOOLEAN canGoUp(int location)
BOOLEAN canGoDown(int location)
```

These functions are called by Primitives to see if the spot location -1 places away is still on the board.

5.2 Other Functions

5.2.1 `InitializeBoard()`

```
void InitializeBoard(char* board)
```

This function takes in a board and inserts a space character (' ') at each index.

5.2.2 `convertText()`

```
MOVE convertText1(STRING input)
MOVE convertText2(STRINg input)
```

These functions convert stringed inputs into moves. These functions are straightforward string manipulations.

5.2.3 `getBoardToRotate()`

```
int getBoardToRotate(MOVE move)
```

Takes in a MOVE and returns the subboard to rotate.

5.2.4 `getColumn()`

```
int getColumn(MOVE move)
```

Takes in a MOVE and returns the column where the piece should be placed.

5.2.5 `getRow()`

`getRow(Move move)`
Takes in a MOVE and returns the row where the piece should be placed.

5.2.6 `isRotateClockwise()`

`BOOLEAN isRotateClockwise(MOVE move)`
Takes in a MOVE and returns TRUE or FALSE depending on the positivity of the MOVE.

6 Print Formats

The printing format can be changed by changing the PrintFormat value in the beginning of the code.

`PrintFormat1()` allows easier visibility of diagonals, whereas `PrintFormat2()` allows easy subboard locations.

Each of these functions calls printBoardBoardRow (1 or 2), and these functions print out the subboards to the right of the game boards.

6.1 `PrintFormat1()`

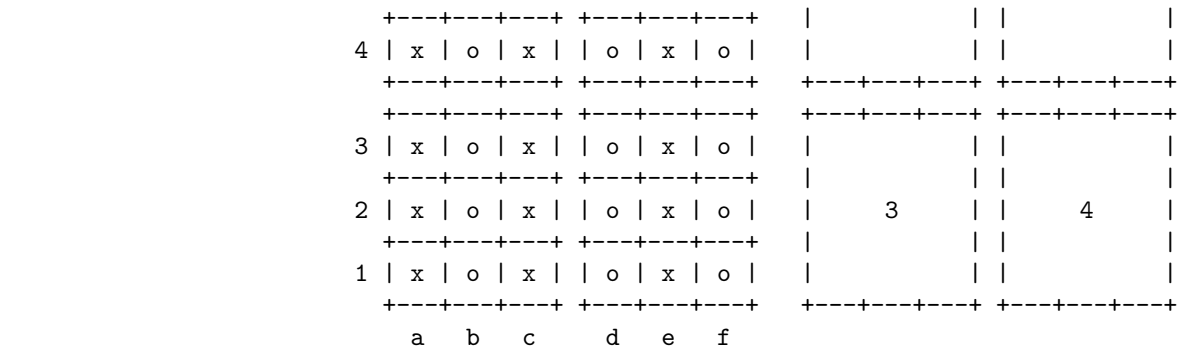
`PrintFormat1()`draws the board in this fashion:

```
   +---+---+---+---+---+---+   +---+---+---+---+---+---+
 6 | x   o   x | o   x   o |   |           |           |
   |           |           |   |           |           |
 5 | x   o   x | o   x   o |   |     1     |     2     |
   |           |           |   |           |           |
 4 | x   o   x | o   x   o |   |           |           |
   +---+---+---+---+---+---+   +---+---+---+---+---+---+
 3 | x   o   x | o   x   o |   |           |           |
   |           |           |   |           |           |
 2 | x   o   x | o   x   o |   |     3     |     4     |
   |           |           |   |           |           |
 1 | x   o   x | o   x   o |   |           |           |
   +---+---+---+---+---+---+   +---+---+---+---+---+---+
     a   b   c   d   e   f
```

6.2 `PrintFormat2()`

`PrintFormat2()` draws the board in this fashion:

```
   +---+---+---+ +---+---+---+   +---+---+---+ +---+---+---+
 6 | x | o | x | | o | x | o |   |           | |           |
   +---+---+---+ +---+---+---+   |           | |           |
 5 | x | o | x | | o | x | o |   |     1     | |     2     |
```



7 Move Formats

`moveFormat1` and `moveFormat2`

The move format can be changed by changing the moveFormat value in the beginning of the code. moveFormat1 allows user input where the subboard rotation direction is represented using cc or cw. For example, a32cw.
moveFormat2 represents the subboard using +/-, where + is counter-clockwise and - is clockwise. For example, b12+