

## Introduction

Octapawn is a variant of well-known, chess like puzzle HexaPawn<sup>1</sup>. OctaPawn is played on a 4 x 4 board with four black pawns versus four white pawns. The pawns can only move forwards, one square at the time (not two). To “take” an opponents pawn, a player must move diagonally. A player win if his/her pawn reaches the far rank first, and lose if no pawn can't be moved when it's his/her turn.

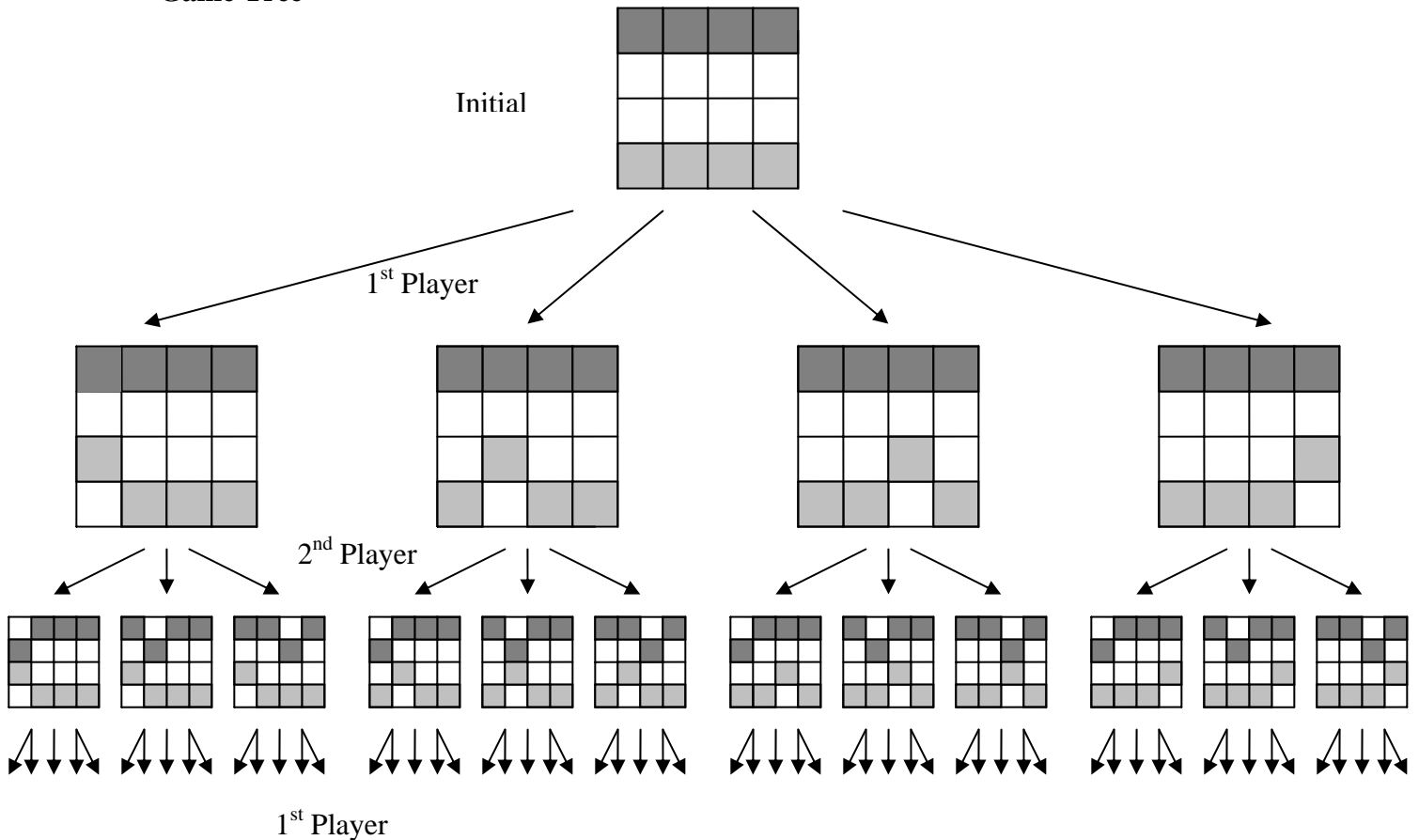
14/11/2006

## Game Rules

The game rules based on the original hexapawn game.

1. Played on 4 x 4 board.
2. Each player has four chess pawns.
3. Each pawn may be moved one square forward but not backward.
4. A pawn may move one square forward if the square ahead of it is empty.
5. A pawn may capture an opposing pawn if that pawn is diagonally ahead of it.
6. A player loses if he/she has no legal moves or the other player reaches the end of the board with a pawn.

## Game Tree



First Alternative Board Representation

30	31	32	33	34	35
24	25	26	27	28	29
18	19	20	21	22	23
12	13	14	15	16	17
6	7	8	9	10	11
0	1	2	3	4	5

1. The pawns for first player are placed from squares 7 to 10, and squares 25 to 28 are for second player pawns.
2. The game board for 4 x 4 dimensions is extended to 6 x 6 dimensions which give the total of 36 numbers of squares.
3. The purpose of the extended squares is as an indicator, to indicate that any pieces will not hit outside the game space.

**Classes****Square**

Each square on the board has unique value. The square may be empty, extended, or filled with pawn type and its color, or perhaps pawn strength. Therefore, a class called "Square" needs to be created. This method creates static square type for empty, extended, first and second pawn.

**Board**

From "Square" class, a "Board" class can be created. The board is made up from "Square" array of size 36. The methods in "Board" are instantiate orderly by first filling all squares as "extended square", and then replace some of squares as "empty squares" and fill pawn "initial position". Perform move method is included, and it acts as updating board state after any move has be made.

**BaseRules**

The following methods have to be implemented.

1. First, a pawn must not hit the extended squares. The valid range are  
 $0 < \text{"sq. number"} \bmod 6 < 6$   
 $6 < \text{"sq. number"} < 29$   
 Square validity checked method is needed.
2. Since the rules state that pawn only can move one square forward, it seems that the number of square difference from "current square" to "square ahead" is 6.

E.g. pawn moves from sq. 8 to sq. 14, and from sq. 25 to sq. 19. Before making any move, the square ahead needs to be checked whether it's empty or not.

3. A pawn may capture an opposing pawn if that pawn is diagonally ahead of it. There are two possibilities, either to diagonally left or right. The number of square difference from "current square" to "diagonally square ahead" are 5 and 7 for diagonally left and right respectively.

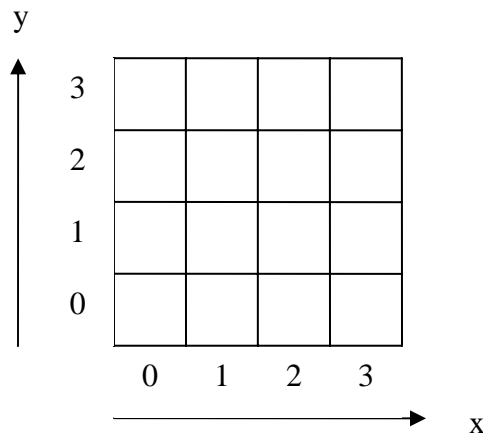
The methods in BaseRules list all possible legal moves of current position from both players. The legal moves will be stored in array (ArrayList or Vector) of "Board".

### Engine/Evaluation

The game tree will be evaluated in this class. This class determines how well the machine plays, by getting the best possible move in short period of time. The types of searching that is going to be used are min-max<sup>2</sup> search and alpha-beta<sup>3</sup> pruning.

28/11/2006

### Alternative to Board Setup– Two Dimensional



Each square on the board can be notated using x and y axis as (x, y) coordinate. The value of x and y must be between 0 and 3.

1. Pawns for first player are put on squares (0, 0), (1, 0), (2, 0), (3, 0) and squares (0, 3), (1, 3), (2, 3), (3, 3).
2. First, a pawn must not hit square with  $x < 0$ ,  $x > 3$ ,  $y < 0$ ,  $y > 3$ .
3. Since the rules state that pawn only can move one square forward, it can be done by adding/subtracting 1 to y current value e.g. first player pawn at sq(0, 0) will move to sq(0, 1) and second player pawn will move from sq(0, 3) to sq(0, 2). Before making any move, the square ahead needs to be checked if it's empty or not.
4. A pawn may capture an opposing pawn if that pawn is diagonally ahead of it. There are two possibilities, either to diagonally left or right. For the first player, let say pawn is at sq(x, y). Then to diagonally left capture will be (x-1, y+1) and diagonally right capture will be (x+1, y+1). And for the second player, let say pawn

is at  $sq(x, y)$ . Then to diagonally left capture will be  $(x-1, y+1)$  and diagonally left capture will be  $(x+1, y+1)$ .

### One Dimensional Array Vs Two Dimensional Array Board Representation

Comparison	One Dimensional Array	Two Dimensional Array
No. of Array	1 (linear)	1 (matrix)
Size of Each Array	At least 16	4 x 4
Are Outer Edges to make sure pawn does not go "off the board" needed?	Yes	No
Pawn Move Formula	Simple	Simple
Pawn Capture Formula	Simple	Simple
Implementation	Easy	Easy
Array Handling	Easy	Easy and convenient

### Project Aim

The project will be implemented using JAVA. It's decided to take two-dimensional board representation. The main goal of this project is focusing on 4 x 4 board game playing level. But later, hopefully player can choose other board dimensions. For smaller board size, min-max searching algorithm is going to be used since the game tree would not grow larger. But when the board size is getting larger, the min-max search would become slow. So alpha-beta pruning has to be used so that selected tree branches are taken, and as a result, it reduces searching time.

And lastly, when the project is done, it would part of open-source software.

## Project Time Table

Month	Date	Progress
November	14/11	Timetable, game rules, and game tree
	21/11	Design First Alternative – One Dimensional Board Representation
	28/11	Design Second Alternative – Two Dimensional Board Representation
December	5/12	List out game classes and methods
	12/12	
	19/11	Preparation for Coming Exam
	26/12	
January	2/1	
	9/1	
	16/1	Exam ended on the 19th
	23/1	First Implementation – Using Two Dimensional Board Representation
	30/1	Insert game engine using Min-Max Algorithm
February	6/2	Insert game engine using Alpha-Beta Pruning
	13/2	Optimize Code and Debug
	20/2	GUI
	27/2	
March	6/3	Writing Report
	13/3	Revise Report
	20/3	
	27/3	

Note: An interim report is due in by Friday December 8th 2006  
Final project deadline - 12 noon on Friday March 30<sup>th</sup> 2007

## References

1. <http://en.wikipedia.org/wiki/Hexapawn>
2. <http://www.seanet.com/~brucemo/topics/minmax.htm>
3. [http://en.wikipedia.org/wiki/Alpha-beta\\_pruning](http://en.wikipedia.org/wiki/Alpha-beta_pruning)