|  |
| --- |
|  |

|  |
| --- |
| **Draughts Game** |
| Project Documentation |
|  |
|  |
|  |
| **Kevin Smith T00048442** |
| **3/2/2012** |
|  |

# Project Description

The aim of this project is to model a draughts game and its rules in a way that may be used with any frontend required. The frontend that will be developed to demonstrate the game model will be a Windows Phone application. The draughts game will be playable against another human player or may be played against an AI opponent. The AI’s decision making process will be accomplished with the use of a min max algorithm.

**Draughts Game Model**

# 

The frontend will be coded on the XNA framework. The project has been developed with a model view controller pattern in mind. During development the focus was on successfully creating an application which provided the necessary functionality required. As the XNA framework and phone platform where both new to me the development process progressed hand in hand with the learning process. And as a result the initial coding approach was to focus on accomplishing functionality with considerations such as performance and code optimisation being postponed till later in the development process.

Model

Controller

View

**XNAGameScreenView**

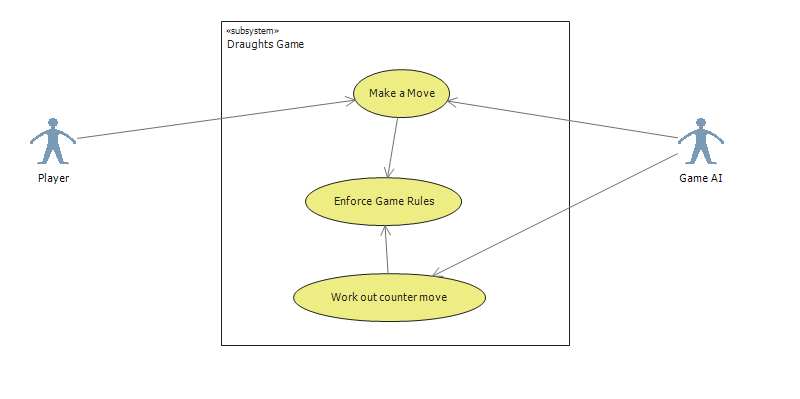
**XNADraughtsGame**

**Draughts Game Model**

The game board is represented by an 8 x 8 2D array of type Draught Piece.

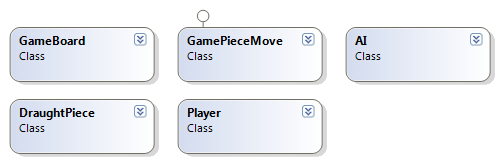
To relate the physical board on the game screen to the model of the board I’m using an 8 x 8 2D array of type Rectangle with each rectangle corresponding to Draught Piece or a null value in the case of an empty square.

# Use Case Diagram



# VOPC

## Draughts Model Classes



## Enums



## Windows Phone XNA Classes



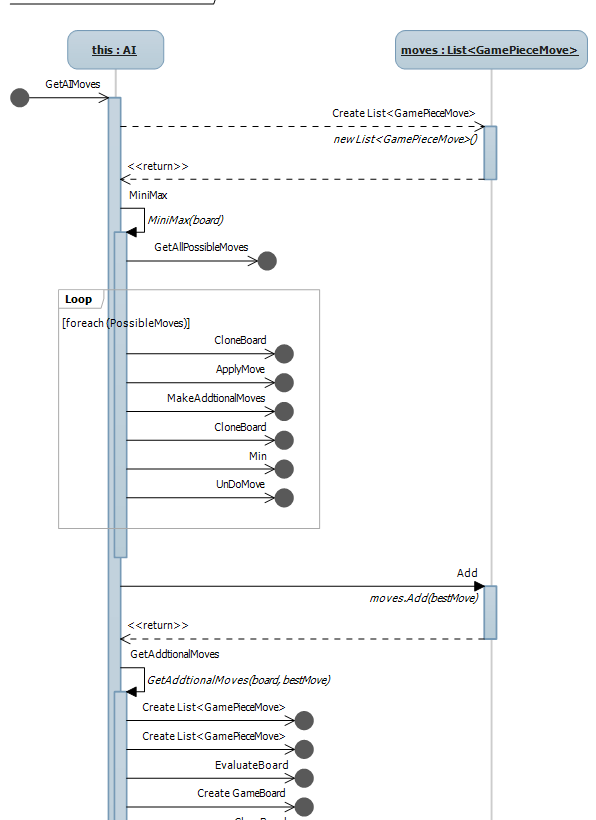
# Flow of Events

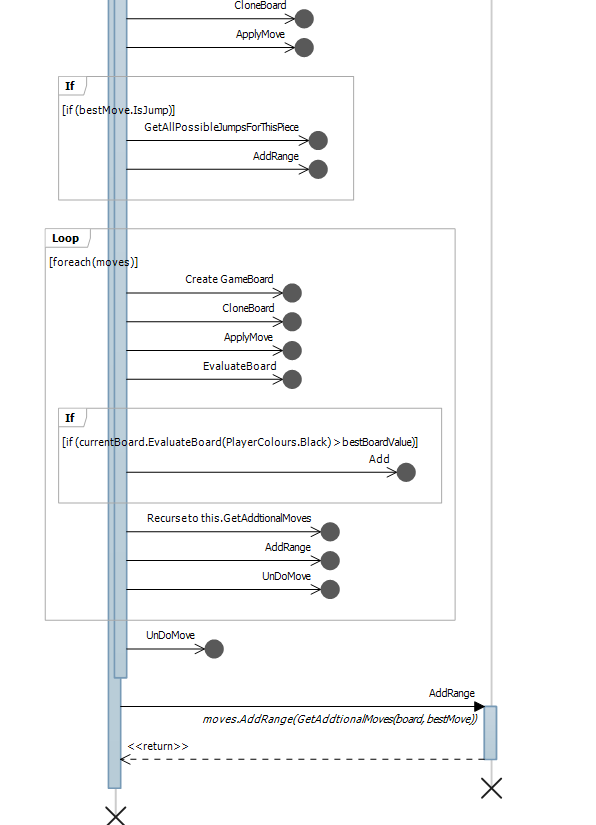
### Game Turn

1. Player selects a draught piece
2. The System determines if this is a movable piece.
3. If piece is movable.
   1. Board squares that may be moved to are highlighted.
4. Players clicks on game board square
5. The System determines that this is a valid move
6. If move invalid
   1. Return piece to original position.
   2. Repeat to step 1.
7. If move valid AI turn begins
   1. System creates list of all possible moves
   2. System creates a tree of possible moves for itself and player to a predefined depth or till game over.
   3. System evaluates paths thru tree and selects best next move.
   4. System makes move.
8. If game over
   1. Announce winner.
9. Else
   1. Return to step 1.

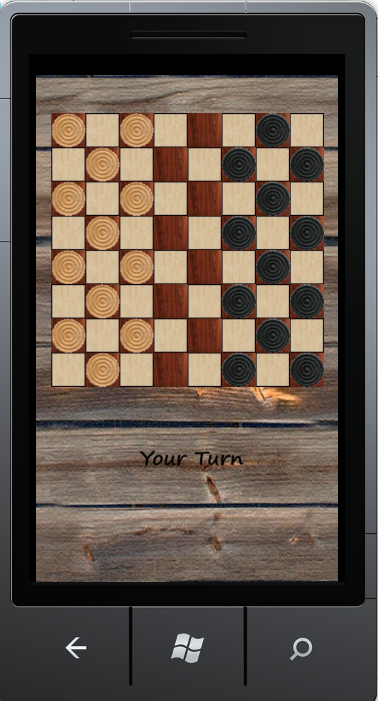
# Sequence Diagram

A sequence diagram showing a single iteration thru the AI move moves process.



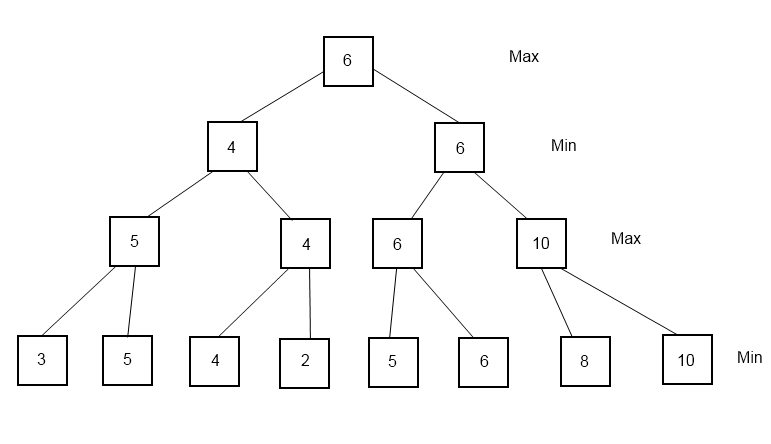


# Screen Prototype



# Min Max Algorithm Details.

As draughts is a two player full information game (i.e. both players are aware of every possible move the other may make) the algorithm that I will use is the min max algorithm which will traverse the game tree.

The AI player is the **max** player and the human player is the **min** player. A search tree is generated, starting with the current position till the game is over or a specified tree depth or a specified amount of time has passed.

A representation of how min max works for a game where 2 moves are available each turn.

* When the leaf nodes are reached the game board is evaluated.
* The max nodes will take the highest value of its children.
* The min nodes will take the lowest values of its children.

### Min Max Pseudo code

(Minmax starts from the point of view of the max player)

**public void Minmax (GameBoard board) {**

Set bestValue = 0;

Get a List of all possible moves for max player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth to Min();

Set currentValue to return value of Min();

if (currentValue > bestValue)

{

Set bestMove = move;

bestValue = currentValue;

}

Undo the move;

}

**}**

**public int Min(Gameboard board, int RequiredDepth){**

if (the game is over() or the depth is reached)

return the evaluated value of the current board;

else

{

Set bestValue = 0;

Get a List of all possible moves for min player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth and current depth + 1 to Min();

Set currentValue to return value of Min();

if (currentValue < bestValue)

{

bestValue = currentValue;

}

Undo the move;

}

return bestValue;

}

**}**

**public int Max(Gameboard board, int RequiredDepth){**

if (the game is over() or the depth is reached)

return the evaluated value of the current board;

else

{

Set bestValue = 0;

Get a List of all possible moves for max player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth current depth + 1 to Max();

Set currentValue to return value of Max();

if (currentValue > bestValue)

{

bestValue = currentValue;

}

Undo the move;

}

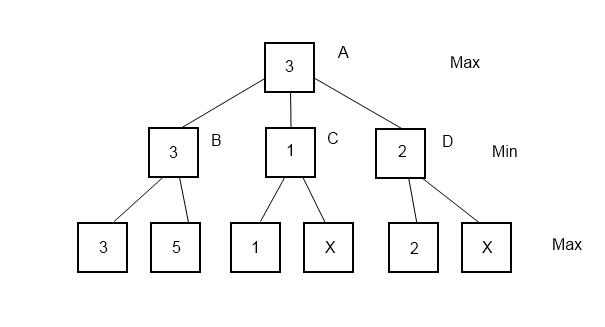
return bestValue;

}

**}**

## Alpha Beta Pruning

If we consider the first turn in draughts where each player has moved once there is already 7 possible moves for each player making a total of 49 possible boards. So we can see how quickly the search tree may become too large to generate (Draughts has nearly 6 million possible board configurations). Two ways of limiting the tree are to specify a limit to the depth of the tree or a limit to the time spent searching traversing the tree. However doing this decreases the effectiveness of the algorithm.

With alpha-beta pruning we use two more parameters alpha and beta. To start with alpha is set to minus infinity, beta to plus infinity.

**Using Alpha Beta Pruning we can eliminate some of the paths thru the tree. In the above example**

* **A which is a max node will take the highest value of its children.**
* **B, C, D min nodes will take the lowest value of their children.**
* **Since B = 3 and C’s first child is 1 there is no need to evaluate the next child of C as it will only be taken if it is less than 1. And A will not take C since B is higher.**
* **The same applies to D.**

### ****Min Max with Alpha Beta Pruning Pseudo code****

(minmax starts from the point of view of the max player)

Set alpha = 0;

Set beta = MaxValue;

**public Gameboard Minmax(GameBoard board, int RequiredDepth){**

Set bestValue = 0;

Get a List of all possible moves for max player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth and alpha and beta to Min();

Set currentValue to return value of Min();

if (currentValue > bestValue)

{

Set bestMove = move;

bestValue = currentValue;

}

Undo the move;

}

**}**

**public int Min(Gameboard board, int RequiredDepth){**

if (the game is over() or the depth is reached)

return the evaluated value of the current board;

else

{

Set bestValue = 0;

Get a List of all possible moves for max player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth, current depth + 1, alpha and beta to Max();

Set currentValue to return value of Max();

if (currentValue < bestValue)

{

SetbestValue = currentValue;

beta = bestValue;

}

if (beta <= alpha)

return bestValue;

Undo the move;

}

return bestValue;

}

**}**

**public int Max(Gameboard board, int RequiredDepth, int alpha, int beta){**

if (the game is over() or the depth is reached)

return the evaluated value of the current board;

else

{

Set bestValue = 0;

Get a List of all possible moves for max player.

foreach (move in the list of PossibleMoves)

{

Make a copy of the current game board;

Apply the move to the copy of the board;

Check for additional jumps

Apply any additional jumps

Pass this copy of the board and required depth, current depth + 1, alpha and beta to Min();

Set currentValue to return value of Min();

if (currentValue > bestValue)

{

SetbestValue = currentValue;

alpha = bestValue;

}

if (beta >= alpha)

return bestValue;

Undo the move;

}

return bestValue;

}

}

# Performance Analysis

The solution containing the draughts model classes and unit test project also contains a console application project that is used to run and produce some metrics with regard to the performance of the MinMax algorithm with and without the alpha pruning component.

**Two Step MinMax**

Calls to Min method 7

Calls to Max method 49

**Four Step MinMax**

Calls to Min method 14

Calls to Max method 30

**Eight Step MinMax**

Calls to Min method 28

Calls to Max method 42

**Until Game Over MinMax**

Calls to Min method 188

Calls to Max method 189

**Two Step MinMax without Alpha Beta Pruning**

Calls to Min method 7

Calls to Max method 49

**Four Step MinMax without Alpha Beta Pruning**

Calls to Min method 386

Calls to Max method 2921

**Eight Step MinMax without Alpha Beta Pruning Halted before completion**

Calls to Min method 37966

Calls to Max method 307971

**Until Game Over MinMax without Alpha Beta Pruning Not Run do to the time and resource constraints.**

These simple metrics display the complexity of the game trees as well as the benefits of the alpha pruning technique. Performance analysis also highlighted an issue where a check to ensure that a move to a position outside of the game board was prevented by catching an out of bounds exception. This caused quite a slow down on performance and so the try catch block was replaced by coding a conditional statement to handle this eventuality.

# Future Improvements

If development where to continue on this project I would consider converting from an XNA framework to a Silverlight & XNA Hybrid which was recently released by Microsoft. This type of project would allow the menu’s to be implemented separately from the main game and so require less state management. Also it would allow easier access to some phone features such as the screen dynamic menu bar.

I would also improve the evaluation method to include other factors and hence increase the apparent intelligence of the AI. Other factors that could be considered are.

* Number of home spaces protected
* Number of opponent home spaces unprotected
* Number of opponent pieces trapped
* Number of player pieces trapped
* Number of available paths to king square
* Proximity to a king square

A leader board could be added to keep track of best results such as

* quickest game
* Most pieces left at end of game.
* Least amount of moves required to win game

This information could be stored locally using the Isolated Storage feature of Windows Phone applications.

Support for WAN games could be added. This would involve the setting up of a server service to handle enable any phones running the client game software to interact with each other.

# Project Deliverables

## **Software**

**DraughtsGameTestAndAnalysis Solution**

**DraughtsCoreClassLibrary Project (Draughts model classes contain extra code for testing and analysis not contained in deployed versions)**

**DraughtsCoreClassesTests Project**

**DraughtsGamePerformance Analysis Project**

**DraughtsGame Solution**

**WPDraughts Project**

**WPDraughts Content**

## **Documentation**

**Draughts Project Documentation.docx**

**Test Plan.docx**

**Test Results.docx**

**Project code.docx**