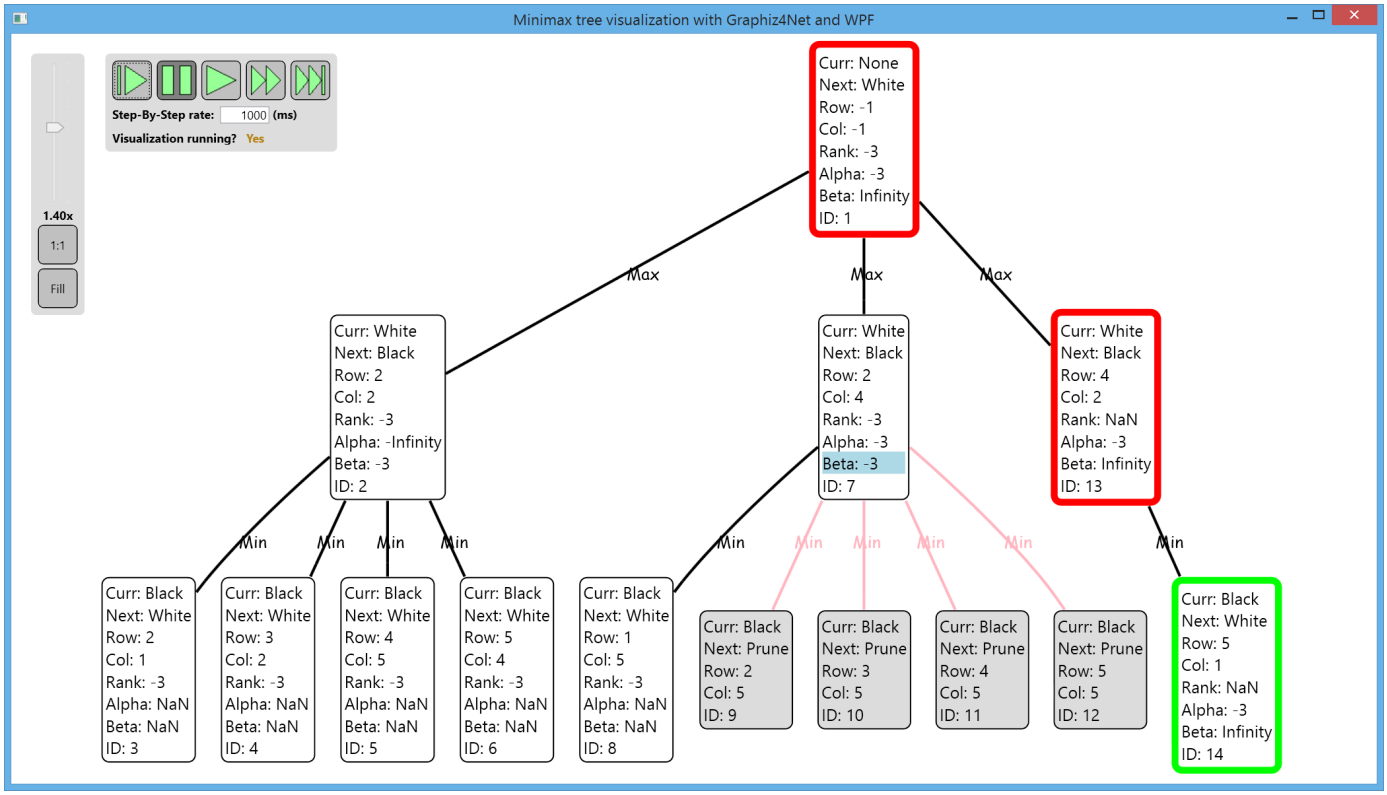
Minimax Visualization

Visualization

A visualization toolkit is provided in the subproject “TreeVisualization”:



The visualization has 5 buttons:

- **Single Step**: Process one step of the visualization and then pause again.
- **Pause**: Pause the visualization.
- **Step-By-Step** (Play): Process steps of the visualization at a regular rate, specified by the “Step-By-Step rate” field.
- **Fast Forward**: Process steps of the visualization at max speed, but still show visuals.
- **Skip To End**: Process steps of the visualization at max speed, but do not show visuals until the end.

Zooming in the window can be controlled with the buttons on the left, or via the mouse (see the zoom button tooltips for details).

There are two ways to use the visualization: to view how the Full Sail AI performs minimax processing (which you can see in the Example_Visualization folder of the project), or to visually debug your own program. In the case of your own program, you will need to reload the TreeVisualization subproject in Visual Studio to use the visualization. You will also need to set the Boolean "visualizationFlag" to true in your code.

Full Sail AI Visualization

To view the Full Sail AI visualization, look at the Example_Visualization folder of the project. For reference, it uses highlighting colors in the following ways:

- **Green**: This is the node currently being processed.
- **Red**: These are previous nodes which are not yet finished processing.
- **Orange**: This is an alpha-beta node which is being updated.
- **Pink/Light Blue alpha/beta**: This is a node which alpha or beta pruned some of its children.
- **Gray with Pink parent link**: This is a node which has been pruned.

Student AI Visualization

To use the visualization toolkit in your AI, you will need to call a few functions regularly in your code. The interface you will use is called TreeVisLib. A rough pseudocode outline of how the TreeVisLib library should be used in regular (not alpha-beta) minimax is as follows:

```
Run()
{
   StartVisualization();
   StartRoot();  // no current player!
   GetBestMove(board, color, depth);
   FinishRoot();
   FinishVisualization();
}

GetBestMove()
{
   for each move:
        StartMove();
        // Make move on board & call GetBestMove()
        // Recursively call GetBestMove(), if necessary
        // Figure out rank of current move
        FinishMove();
}
```

Alpha-beta pruning follows a similar flow, but is more complex:

```
Run()
{
   StartVisualization();
   parentId = StartAlphaBetaRoot();  // no current player!
   GetBestAlphaBetaMove(board, color, depth, alpha, beta, parentId);
   FinishAlphaBetaRoot();
   FinishVisualization();
}

GetBestAlphaBetaMove()
{
   for each move:
        if move is pruned:
             StartAndFinishPrunedMove();
        else:
             nodeId = StartAlphaBetaMove();
             // Make move on board
             if leaf node:
                  UpdateAlphaBetaMove(nodeId);
             else:
                  // Recursively call GetBestAlphaBetaMove()
                  // Figure out rank of current move
                  if updated best move:
                       UpdateAlphaBetaMove(parentId);
                       if alpha or beta pruning is starting:
                            HighlightAlpha/Beta(parentId);
             FinishAlphaBetaMove();
}
```

Any highlighting methods not shown above should be used whenever you feel they are appropriate.

Visualization API

You can find these functions in TreeVisLib.cs in the CoreAI subproject. Only the data structures and functions which are potentially useful for regular minimax, or minimax with alpha-beta pruning, are described here – please see TreeVisLib.cs itself for a description of the other functions.

PS: In **Debug** mode in Visual Studio, after calling any of these functions, you may need to press Play in the visualization window to return control to your code!

Data Structures

```
public enum Node.MinMax : int { Min, Max, None };
```

Tells visualization what level this move is on (i.e. this move's rank will be selected if minimum, or selected if maximum, or is not specified). The parent link will be labeled with this.

```
public enum Node.Player : int { Black, White, Prune, None };
```

Tells visualization what player is being used for this move, or for the next move. Black and White should be used for moves by those players, Prune is used for moves which are alpha-beta pruned, and None should be used if you do not wish to specify any player (for example, you may not wish to figure out the next player, which is an optional field in the visualization).

Regular Minimax Methods

```
public void StartVisualization()
```

Resets TreeVisualization window and all associated variables. Always call this function before each visualization.

```
public void FinishVisualization()
```

Does some bookkeeping, and in fast-forward mode, actually updates the TreeVisualization window. Always call this function last.

```
public int StartRoot(Node.Player nextPlayer)
```

Starts the root node in visualization (i.e. adds in a node for it). Uses dummy values for unknown parts (row, column, and rank). nextPlayer is who will take the next move.

Returns a node ID of the node created, or an error code if unsuccessful.

```
public void FinishRoot(int row, int column, int rank)
```

Finishes the root node in visualization, adding in row, column, and rank of move to be picked. Make sure not to start any more moves after this.

```
public int StartMove(Node.MinMax minMax, Node.Player player, int row, int column, [optional] Node.Player nextPlayer)
```

Starts move in visualization (i.e. adds in a node for it). Uses dummy values for unknown parts (rank). minMax refers to whether this move is at max or min level. player corresponds to move at row, column. nextPlayer is who will take the next move, and is an optional argument.

Returns a node ID of the node created, or an error code if unsuccessful.

```
public void FinishMove(int rank)
```

Finishes move in visualization, adding in rank. Make sure not to start any child moves after this.

Alpha-Beta Minimax Methods

```
public int StartAlphaBetaRoot(Node.Player nextPlayer)
```

Starts the root node in visualization (i.e. adds in a node for it). Uses dummy values for unknown parts (row, column, and rank). nextPlayer is who will take the next move.

Returns a node ID of the node created, or an error code if unsuccessful.

```
public void FinishAlphaBetaRoot(int row, int column, int rank)
```
Finishes the root node in visualization, adding in row, column, and rank of move to be picked. Make sure not to start any more moves after this.

```
public int StartAlphaBetaMove(Node.MinMax minMax, Node.Player player, int row, int column, double alpha, double beta, [optional] Node.Player nextPlayer, [optional] double rank)
```
Starts alpha-beta move in visualization (i.e. adds in a node for it). Uses dummy values for unknown parts (rank). minMax refers to whether this move is at max or min level. player corresponds to move at row, column. nextPlayer is who will take the next move after this one. rank is the rank of this move, which will often not be known but can be specified if it is known.

Returns a node ID of the node created, or an error code if unsuccessful.

```
public void UpdateAlphaBetaMove(int id, int rank, double alpha, double beta)
public void UpdateAlphaBetaMoveRank(int id, int rank)
public void UpdateAlphaBetaMoveAB(int id, double alpha, double beta)
```

Updates the alpha-beta move with id in visualization to current rank and/or alpha and beta values. Call this after each child move finishes.

```
public void FinishAlphaBetaMove()
```
Finishes alpha-beta move in visualization. Make sure not to start any child moves after this.

```
public int StartAndFinishPrunedMove(Node.MinMax minMax, Node.Player player, int row, int column)
```
Starts and finishes pruned alpha-beta move in visualization (i.e. adds in a node for it). minMax refers to whether this move is at max or min level. player corresponds to move at row, column.

Highlighting Methods

```
public void HighlightNode(int nodeId, string color, [optional] int highlightBorderSize)
```
Highlights a node using color and (optionally) highlightBorderSize. Get nodeId from StartMove() or from watching the visualization. "color" can be a named color or hex value.

```
public void UnHighlightNode(int nodeId)
```
Un-highlights a node.

```
public void HighlightAlpha(int nodeId)
public void HighlightBeta(int nodeId)
public void HighlightAlphaBeta(int nodeId)
```
Highlights alpha and/or beta field of a node. Color is specified in MainWindow.xaml.

```
public void UnHighlightAlpha(int nodeId)
public void UnHighlightBeta(int nodeId)
public void UnHighlightAlphaBeta(int nodeId)
```
Un-highlights alpha and/or beta field of a node.

Visualization Troubleshooting

Visualization having issues? Try the following:

- Getting a weird exception in the visualization window about "cannot find the file specified"? Check that your xcopy works on the command line. Alternately, you can do the copy manually by copying all the files from the "Project\TreeVisualization\Graphviz238\ bin\" folder to "Project\TreeVisualization\bin\x64\Debug" folder (or whatever platform\build type you are using). (Yes, this will pollute this directory with lots of files and is an ugly hack. :) The build process also does this, actually. But even if you do a clean/rebuild of the project, those files will remain, fortunately.)
- Too many levels in your visualization tree? Make sure Finish*() (any function that starts with Finish) is called when you are finished with a move. Those calls move back up a level in the tree. If they aren't called properly, levels will keep being added to the tree when they shouldn't.
- Nodes appearing too high up in the tree? This is the opposite of the previous problem. Make sure Finish*() isn't called too early, or the program will go back up the tree early.
- Program hanging after setting visualizationFlag to "true"? Make sure to reload the TreeVisualization project.
- Program going slow after you close the visualization window? The visualization functions can be slow if you close the visualization window, because they sometimes throw exceptions after the window has been closed. It's better to only call the visualization functions based on the visualizationFlag, and control visualization via that.
- Having trouble graphing pruned nodes? Check that you're actually considering all of your moves in your for() loop, even pruned moves. Even though you're not really processing them, you need to look at all of them in order to display their information in the visualization. Make sure you're not exiting the loop early.

Some things that may look like issues but aren't:

- Getting an error that says "Object reference not set to instance of object"? This is normal, actually. Quit the program normally once and the error will go away. (The program always saves settings, like window size and difficulty, upon quitting. The first time the program runs, though, there are no settings, and this error is thrown, though the application still works fine. Also, the program only saves the settings if you quit it normally, so if you don't quit it normally, the settings never get saved.)
- Calling a TreeVisLib function and then Debug stops working in Visual Studio? Click one of the play buttons in the visualization window, and control will return to Visual Studio.
- Root node has -1 for row and column? This is normal also (until you finish). The top node is special, because it corresponds to a move (the eventual move selected, which you return from Run()) that we cannot know until we are finished processing, unlike every other move. Each other node corresponds to a board where a new move has been made, but for the root node, no actual new move has been made (it corresponds to the original Board passed into Run()).