# CSCE625: Introduction to Artificial Intelligence
# Programming Assignment 2:

# Search for Games

*Instructor: Dr. Dylan Shell*
*TA: Brittany Duncan*

*NAME: Tingshan YU*
*UIN: 522006225*
*E-Mail: tyu@tamu.edu*

## ***The definition of my game tree:***

In this programming assignment, we need to deal with a simple game played between two people. A pile of n matches is placed between player 1 and player 0, each player takes turns to remove 1, 2 or 3 matches each step from the pile. The player who takes the last match will lose the game. This is a zero-sum game.

First of all, let me give a definition of my game tree:

***State:***

The state in my program is defined to be the remaining matches in the pile at current round of game. States are nodes in game trees generated. The initial value is n.

***Player:***

The player variable in my program will take two values. 1 stands for the player 1 and 0 stands for the opposite player. The two players will take turns to play the game.

***Action:***

The actions are defined to be the edges in the game tree generated by the two tree algorithms—the Minimax Algorithm and the Alpha-Beta Pruning Algorithm. Each action will takes values from the set {1, 2, 3}, which stands for the number of matches a certain player take from the remaining pile. Each action will cause the state move from one node from another.

***Terminal-Test:***

When the state value reaches 1 at any node, the tree stops to generate more nodes from this node. That is to say, the game terminates.

***Utility function:***

Since this is a zero-sum game, I define that the utility function can take two values, 1 and -1, which add up to zero. 1 stands for that the MAX player (player1) wins the game, and -1 stands for player1's loss of the game. -1 stands for that the MIN player (player0) wins the game and 1 stands for player0's loss.

After clearly defined the game tree, it is generated using the Minimax Algorithm and the Alpha-Beta Pruning Algorithm.

## ***The Minimax Algorithm:***

The Minimax algorithm is used to generate game tree first. It computes the minimax decision from the current state. It uses a simple recursive way to generate the game tree. This algorithm performs a complete depth-first exploration of the game tree.

In my program, the Minimax Algorithm is used to calculate the utilities of every node in my game tree. First of all, we check whether a node is a leaf node (the state value is 1). If the node is a leave node and the current player is the MAX player, then the utility is -1. If the MIN player is to play at the leave node, then the utility is 1.

For each internal node in the game tree, if the current player is MAX player, then its utility is the maximum value of its child-nodes. If the current player is the MIN player, then its utility is the minimum of its child-nodes. I trace back to calculate all nodes' utilities in the game tree.

The pseudo code is as bellow:

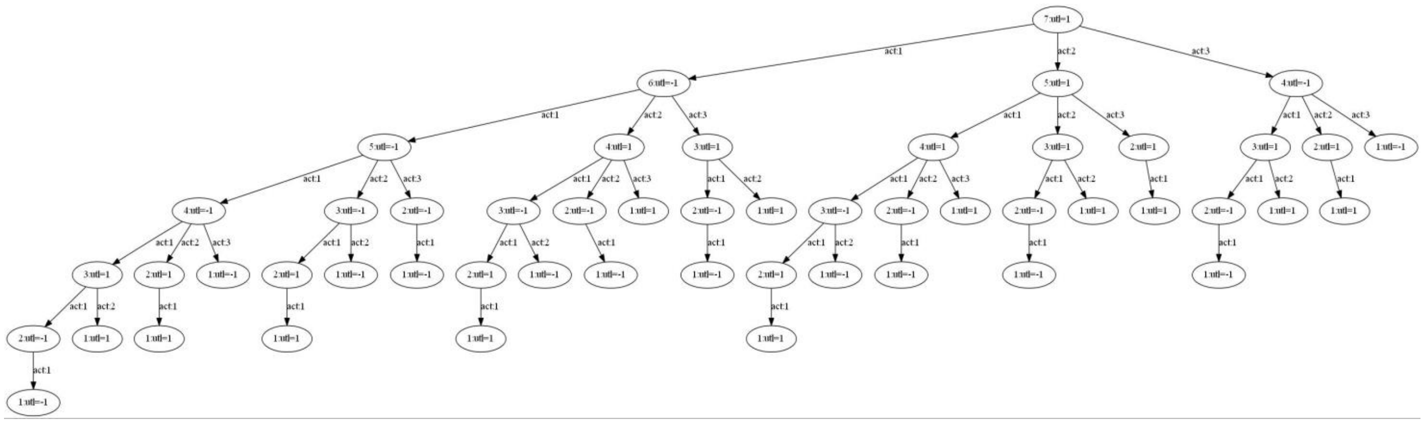
```
function MINIMAX-DECISION(state) returns an action
   return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$
   for each a in ACTIONS(state) do
      $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
   return v
```

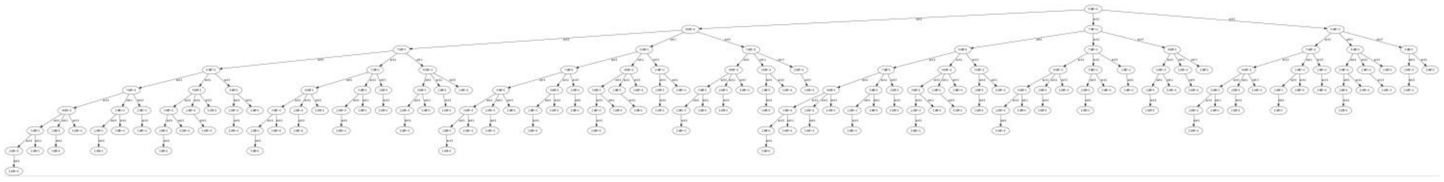
```
function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$
   for each a in ACTIONS(state) do
      $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
   return v
```

After run my program, the dot files of game trees are generated for n = 7, 9, 12, then using the GraphViz, game trees are visualized as below:

For n = 7:



For n = 9:



For n = 12:

You can easily see that the nodes in the game tree will increase fairy fast. Indeed, the amount of nodes increases exponentially with respect to n. Even though my program can generate game trees for n = 15, 21, it will be hard for human eyes to recognize them in a one-page image. Therefore, I only generate image for the game trees when n = 7, 9, 12. As showed above, we can hardly recognize the nodes in the game tree when n = 12.

## ***<u>The Alpha-Beta Pruning Algorithm:</u>***

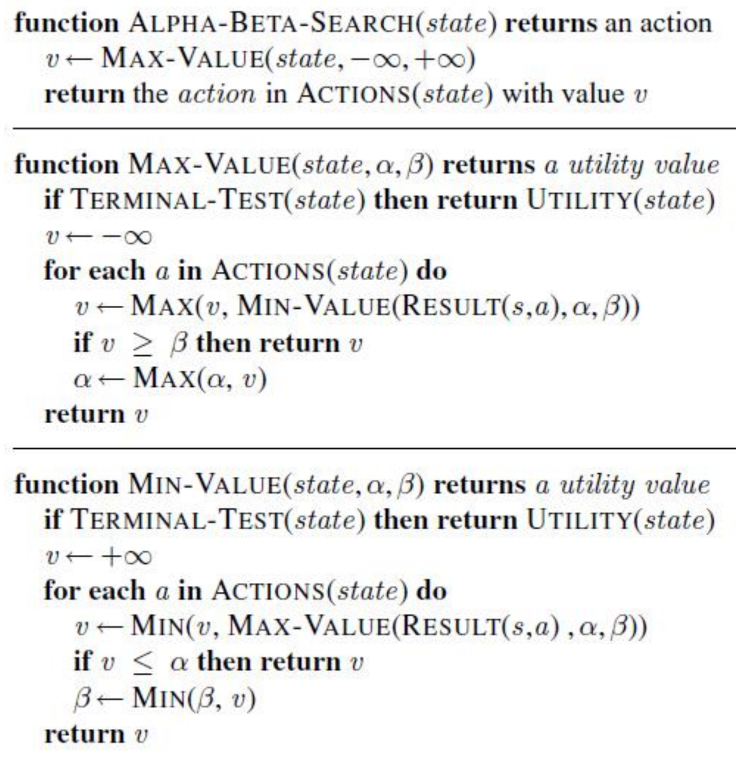
Secondly, I use the Alpha-Beta Pruning Algorithm to generate the game trees.

In the Minimax Algorithm, the amount of states in the game tree is exponential to the tree depth. Even though we cannot avoid this by using Alpha-Beta Pruning, we can decrease it by half.

In Alpha-Beta Pruning, the same move is returned as Minimax, but prunes away branches that cannot possibly influence the final decision.

In my program, Alpha-Beta Pruning is used like this: for any internal node, if the current player is MAX player, then I check its child-nodes from left to right. Once I find one child has the utility +1, then I pruning all the sub-trees rooted at these child-nodes. The reason is that the child-nodes on the right of it cannot have utility larger than +1, +1 is already the maximum. So we do not need to check the nodes on the right of it. If the current player is the MIN player, I check whether there is a node with the utility -1 and prunes the sub-trees rooted on its right.

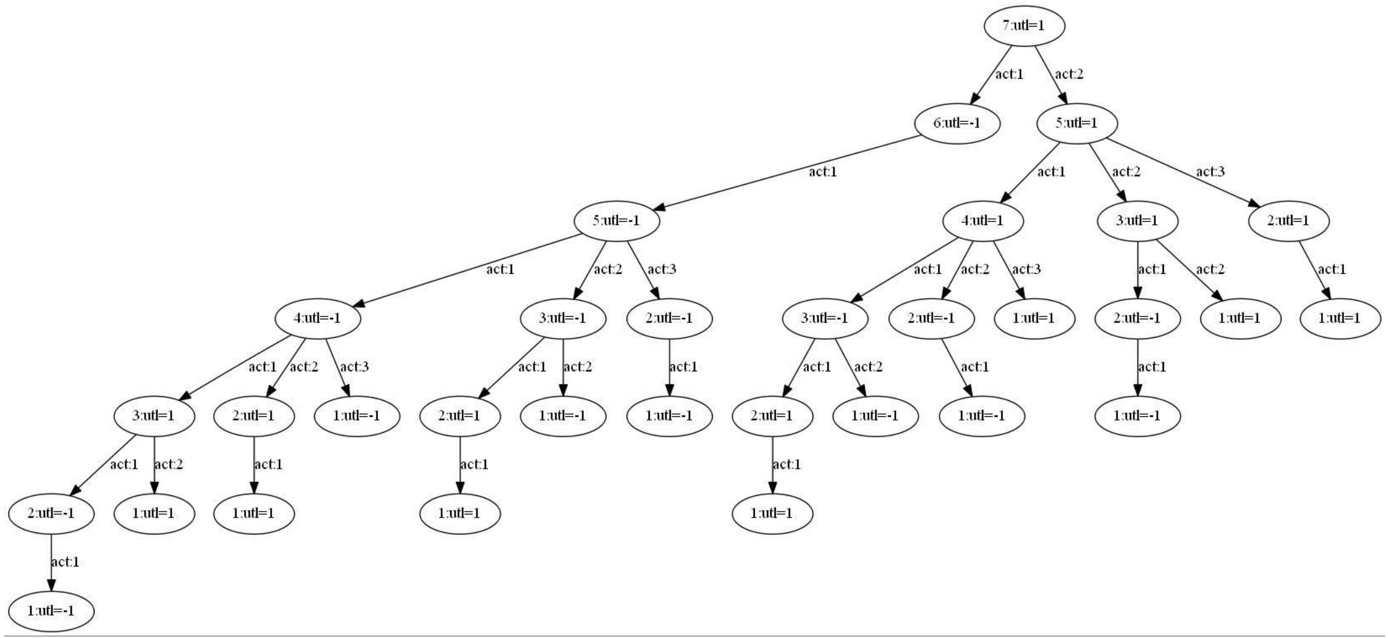
The pseudo code is as below:



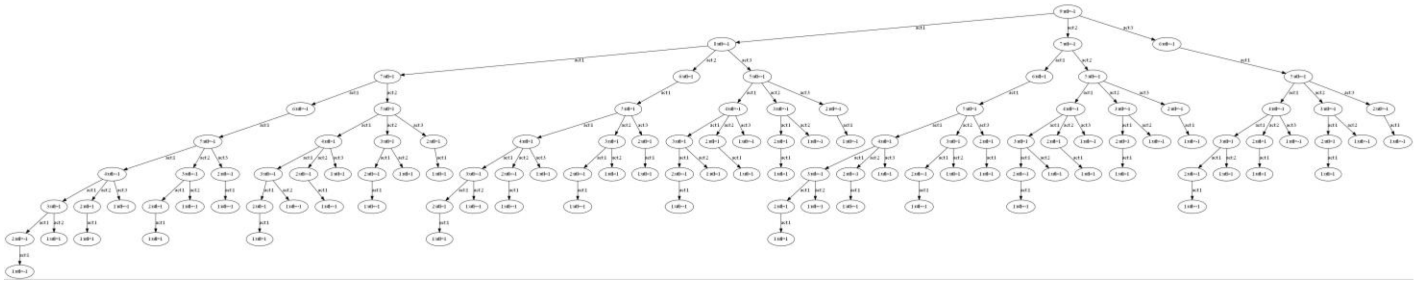
```
function ALPHA-BETA-SEARCH($state$) returns an action
    $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
    return the $action$ in ACTIONS($state$) with value $v$
--------------------------------------------------------------
function MAX-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s,a$), $\alpha, \beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$
--------------------------------------------------------------
function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta$, $v$)
    return $v$
```

After running my program, game trees pruned can be generated. Examples are showed below for n = 7, 9, 12.

For n = 7:



For n = 9:



For n = 12:

## ***<u>My analysis of the games by addressing the questions in 2(a)–2(c)</u>***

### 2 (a)
### what are the relative sizes of the trees?

Suppose that n matches are given as the initial state.

For the Minimax Algorithm and Alpha-Beta Pruning, the amount of nodes with respect to n is showed in the following form:

| **n** | **Amount of nodes by Minimax** | **Amount of nodes by Alpha-Beta Pruning** |
|---|---|---|
| **5** | **15** | **15** |
| **7** | **52** | **32** |
| **9** | **177** | **113** |
| **12** | **1104** | **456** |
| **15** | **6872** | **1600** |

From the above form, we can get that the amount of node grow exponentially with regard to the amount of given matches.

And when we compare the amount of nodes in game trees generated by Alpha-Beta Pruning to Minimax, we can easily find that Alpha-Beta generate far more less nodes than Minimax, especially when n grows larger. In fact, it will be far less than half of Minimax when n is very large. Although it is still exponent, is has been improved by Alpha-Beta Pruning.

For the Minimax, let f(n)be the amount of nodes in game tree of  n matches at the start. Then this formula holds:

$f(n) = f(n-1) + f(n-2) + f(n-3) +1$:
$f(1) = 1$,
$f(2) = 2$,
$f(3)= 4$.

## 2 (b)
## Does opting to go first or second play a role in the outcome of the game? What is the value of the game?

Sure, opting to go first or second fairy matters.

Take the game trees generated as examples. No matter how the opposite player act, the player plays first nearly always can find a path to win the game if the player chooses the optimal action at each step.

The value of the game tree is to search an optimal path to win the game. Its aim is to make optimal decision no matter what the given condition is. Maybe it can be utilized as a part to develop intelligent systems to help people make decisions.

## 2(c)
## How well will the optimal player fair in a play-off against a random player?

When n is large (in fact, 6 is enough), an optimal player can always win a random player. The analysis is as below:

Suppose that the optimal player plays first:
For n = 1, loss;
For n = 2, take one, then win;
For n = 3, take two, then win;
For n = 4, take three, then win;

For n = 5, the utility is -1, that means, if the random player takes the optimal action, the optimal player will loss. But when the random player takes other actions, the optimal player still will win;

Therefore, the optimal player should avoid making decisions from 5 matches. So the optimal player should force the random player to remove from 5 matches.

As each time, each player can remove at most 3 matches. So we can find that if the optimal player can always force the random player to remove from (4*k + 1) matches, it will surely win the game when k decreases to 1 (that is to say, the random player must remove from 5).

How to always leave (4*k + 1) matches to the random player? It is quite easy. When the random player removes j matches, the optimal player just take (4 - j) matches. Then the amount of matches can decreases by 4 periodically.

In sum:

When n=1, the optimal player is certain to loss,

When n = 5 is the initial state for the optimal player to remove, the random player has a probability to win.

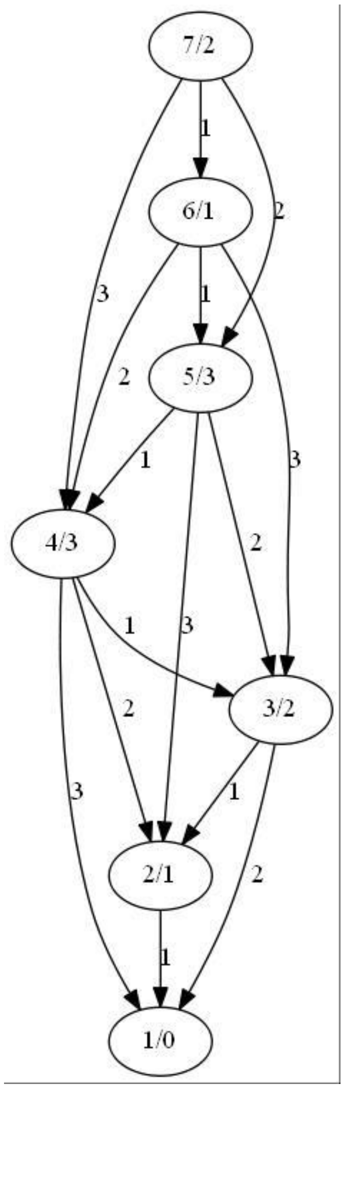
In other case, the optimal player will surely win the game.

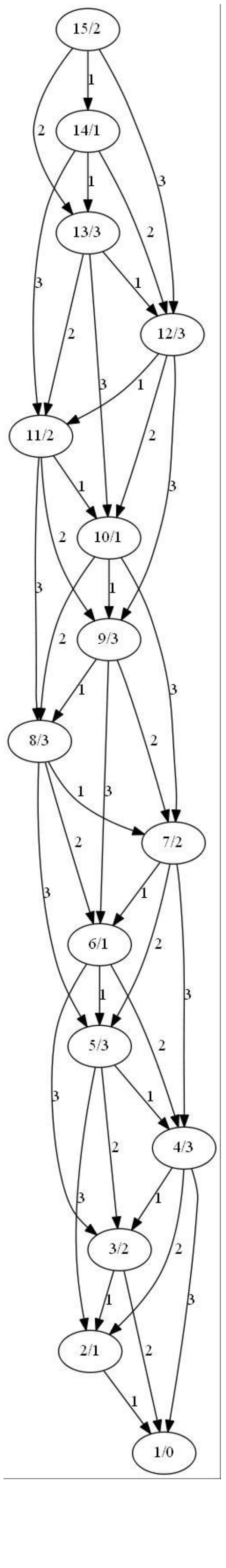
## ***<u>My Moore machines represented in some graphical way:</u>***

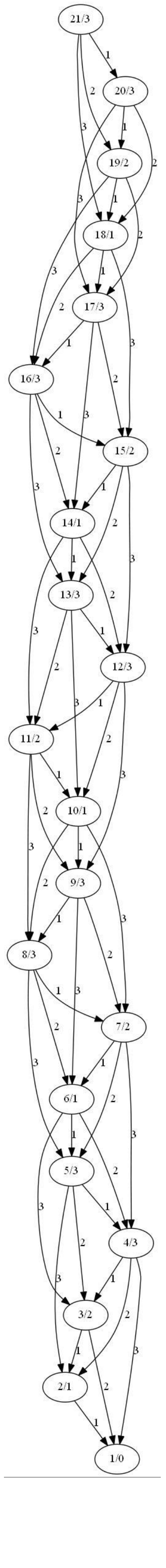
As clearly stated in the question:
The input alphabet is {1, 2, 3} representing the number of matches taken by the opponent.
The output alphabet is {1, 2, 3} representing the number of matches I choose to take, i.e. the optimal action that I should take for the current state.
The state is the amount of remaining matches at the current node.

Running my program, dot files of the Moore Machine for n = 7, 15, 21 is generated, and then I use GraphViz to visualize them as below:

From left to right in the next page, there are Moore Machine for n = 7, 15, 21.


7/2
6/1
5/3
4/3
3/2
2/1
1/0


15/2
14/1
13/3
12/3
11/2
10/1
9/3
8/3
7/2
6/1
5/3
4/3
3/2
2/1
1/0


21/3
20/3
19/2
18/1
17/3
16/3
15/2
14/1
13/3
12/3
11/2
10/1
9/3
8/3
7/2
6/1
5/3
4/3
3/2
2/1
1/0