Project Report – Subtraction Game

Question 1:

We represent the piles as a 1 dimensional array of size 3, with each index representing one of the piles of stones. We do this as the game state is only about the number of stones in each pile, which can be implemented simply. However for the resulting game state of possible moves we are using a 3d array of size [3][1][3], as this basically represents 3 (1 for each possible move) separate 2D arrays that are each 1 row and 3 columns. That 1 row and 3 columns represents each of the new pile sizes for that move.

Question 2:

We can classify a winning game by looking at the resulting gamestate given a certain move. If the next move results in 2 empty piles and the AI went last we know that the AI has won as it was the last player to make a move. If the next possible moves don't result in a winning state we say that the current state isn't a winning state as well. If the next state doesn't have 2 empty piles we recursively call isWinner until it does.

Question 3:

Dynamic programming is a good concept to use because there are many states that the game can be in. In trying to make an unbeatable AI we are going to have to use recursive calls to determine if a game is in a winning state. However, as we have seen with the Fibonacci algorithm, recursive calls waste time in calling already seen numbers. The same principle applies to finding a winning state. So to make our program run faster we store results that we have already used in a separate array and check that to see if we need to make any further calls.

Question 4:

We choose to use a 3d int array, then 2 other 1d Boolean arrays. The 3d array stores the states of moves that have already been called, 1 of the Boolean arrays stores whether or not that state has been seen before and the other then states whether it is a winning game or not. When we call isWinner we then compare the arrays to see if we need to keep calling iswinner or to just return the game and winning state at the current location. They are updated at the same time as to keep the indexes equal.

getMoves Psuedocode:

input: the 3d array created in the main file
output: updated list of game states
if (pile1 >= pile2)
    in 3d array index for move 1 update pile1 to pile1 minus pile2 in the current state
    in 3d array index for move1 update pile2 to pile2 in current state
    in 3d array index for move1 update pile3 to pile3 in current state
if (pile2 >= pile3 && pile 3 is not 0)
    in 3d array index for move2 update pile1 to pile1 minus pile2 in the current state
    in 3d array index for move2 update pile2 to pile2 in current state
    in 3d array index for move2 update pile3 to pile3 in current state
if (pile1 >= pile3 && pile3 is not 0)
    in 3d array index for move 3 update pile1 to pile1 minus pile2 in the current state

in 3d array index for move3 update pile2 to pile2 in current state
in 3d array index for move3 update pile3 to pile3 in current state

isWinner Psuedocode:

input: the current game state
output: whether or not it is a winning game state
create 3d array to hold game states after moves outside function
create Boolean array to determine if game state has been seen or not outside function
create Boolean array to determine if game state is a winner outside function
sort the current state inputted
loop through new game states entered already
compare current state to those game states
if there is a match, check that it has been seen, and then check that it is a win
if it is return true, if its not a winning game state return false
if there is no match add it to the 3d array
call getMoves on the current game state
if current state has 2 piles = 0 and AI went last return true and update memoize data structures
if game is over and user went last return false and update memorize data structures
if game is not over increment turn variable so we will know who went last
create and populate new 1D arrays for each resulting game state of possible moves
if one of these resulting game states is a winning game make then the current state is a winning game so return true and update the memorize data structures

bestMove Psuedocode:

input: current game state
output: resulting game state of the best move
call getMoves to populate the 3d array of possible moves and their game states
create 1d array to hold values of possible game states
populate that array with resulting state from 1st move
test if that is a winning game state or not
if it is return that move, if not populate that array with resulting state from 2nd move
test if that is a winning game state or not
if it is return that move, if not populate that array with resulting state from 3rd move
return that move