Artificial Intelligence 1
Lab 1

Name1 (student number 1) & Name2 (student number 2)
Group name

day-month-year

Programming

Program description

The task was to implement an effective solution for the game of nim. This is the game, where the players can pick 1,2, or 3 matchsticks from a set, and whoever picks the last one loses. The solution is based on the minimax algorithm, that calculates the final outcomes for all possible moves, and chooses the optimal one.

Problem analysis

A version of minimax algorithm for the problem was given, but it was using separate min and max functions, and double function calls with one returning the value of the minimax, and the other returning the move. Our task was to implement a negamax function for the problem, that returns both. The task was also to implement a transposition table, to make the calculations lot more effective.

Program design

To solve the problem of different return types, we introduced a new struct called Choice, that contains both the value and the move. We defined our negaMax function to return this type. The negaMax is based on ..., but it makes the difference between MAX and MIN turns with the turn value, which is 1 (MAX) or -1 (MIN). Whenever a calculation takes place, the correct variables are multiplied by the turn variable (f.r.:), and the return value also depends on the turn variable. Whenever the negaMax is called recursively, the turn value is multiplied by -1, to suit the calcuations for the player in the next turn. With this recursive function we could also avoid to use two different functions in different depths, like ... in the original solution, because every step can be calcuated with a simple recursive call. We also implemented a simple user input to choose

between the classic (original) and the negaMax vesion, to make the comparison easier. The classic solution was not change at all.
Before implementing the transposition table, we also included two minor modifications for the negamax algorithm, that are really simple, but make the program a lot more effective than the original solution. The first was to evaluate the moves in decreasing order: this way, if there are branches with the same value, the biggest step is chosen, resulting in a shorter sequence and faster solution time. The second was to stop looking for other branches if an optimal value (e.g.1 for MAX) has already been found. This is possible, since there are no difference between branch ends, just the value 1 and -1.
The transposition table was designed to store the values for MAX and MIN for the states that have already been evaluated. The implementation is a matrix of Choice structs, that has one dimension for the states (sticks left), and a dimension for the player. (MAX or MIN). In the beginning of every negaMax call, the table is checked, if a value exists for the state and player. If not, the choice is calculated and stored. At the end the function retrieves the value from the table (no matter it is newly stored or not). The table has 101 states, because the state numbering starts from 1, and not from 0, as it is in an array, so a state of 100 sticks needs a [100] index.

Program evaluation

When using the classic version, or the negamax version before the efficiency enhancements were implemented, the program could produce the optimal output, but it was rather slow. It started to show for states above 30, for states around 40 or more it was too slow for us to wait for an output. It was also interesting to see how the exponential growth of branches shows in this task: for the state of 35 for example, the first step takes a long time, the second takes a lot shorter, but still noticable, and the rest is calculated within a second.
After implementing the two tricks, but before the transposition table the efficiency improved significantly: the program was able to calculate the output up to 80 sticks in a couple of seconds. The output showed a different sequence then in the classic method, because of the order changing, but that is not a problem, since there are multiple optimal paths for winning. The transposition table gave an other significant improvement for the program: after the implementation the program was able to produce the correct output for even the maximal 100 matchsticks within a second.

Program output

Program files

Main.c

```
1 Your code here
```

SomeFile.c

```
Some other code here
```