Exercise 4.1
Consider a strongly simplified version of *Blackjack*. In this case, instead of cards a die is used which can take the values 1, 2 and 3 with equal probability. Two players take turns rolling the die and sum up the rolled numbers. Once a player’s total is more than 5, that player loses the game. If a player says *stop*, his opponent may roll one more time, but he doesn’t have to. If both players’ total is less than 6, the player with the higher total wins. If the players are tied, the game ends in a draw.

1. Draw the complete game tree that would be produced by the *expectimax* algorithm if it is *MAX*’ turn, his total at the moment is 3, and *MIN*’s total is 2.

2. Would *MAX* roll again?

Exercise 4.2
Minesweeper, the well-known computer game, is closely related to the Wumpus world. A minesweeper world is a rectangular grid of $N$ squares with $M$ invisible mines scattered among them. Any square may be probed by the agent; instant death follows if a mine is probed. Minesweeper indicates the presence of mines by revealing, in each probed square, the number of mines that are directly or diagonally adjacent. The goal is to have probed every unmined square.

1. Let $X_{i,j}$ be true iff square $[i, j]$ contains a mine. Write down the assertion that there are exactly two mines adjacent to $[1, 1]$ as a sentence involving some logical combination of $X_{i,j}$ propositions.

2. Generalize your assertion from 1) by explaining how to construct a CNF sentence asserting that $k$ of $n$ neighbors contain mines.

3. Give an example of a configuration of probe values that induces long-range dependencies such that the contents of a given unprobed square would give information about the contents of a far-distant square. [*Hint*: consider an $N \times 1$ board]

---

1 Please use the cover sheet from the home page to stitch all sheets together.
Exercise 4.3
Programmieraufgabe
Implement the $\alpha\beta$-pruning algorithm in a programming language of your choice.
As input, the program will be given a tree with integer values in the leaves. It is player MAX’ turn.
Output of the program should be the pruned tree.

IMPORTANT!

1. The program must be executable on junop.informatik.uni-freiburg.de.

2. Das Ergebnis an bbringma@informatik.uni-freiburg.de

3. The input consists of a file called input.tree, the output has to be saved in a file called output.tree. We will not pass any parameters to the program!

4. The input and output data are string-encoded trees. The format is easiest explained by using an example. The following tree

   A
  / \   
 B   C
   / \
  D   E
     / \
    F   G

is encoded as $A(B()C()D(E(F(G()))))$.

   A node label is followed by a pair of brackets which encloses the children of the node. This means that there are always as many pairs of brackets in a tree as there are nodes. A leaf is followed by an empty pair of brackets.

   As node labels for inner nodes and the root node we will use the letter $X$ in the input data. The leaves are labeled with positive and negative integers. In the output tree, the inner nodes and the root node should be labeled with their respective minimax values.

5. Except for the line with the encoded tree there are no additional characters in the input data. The same holds for the first line of the output file. In the second line of the output the names of the group members should be found, making pairing the results with groups easier.

6. Several example files with input data can be found on the webpage with the exercise sheets.
### Alpha-Beta-Search

**Function**  \( \text{Alpha-Beta-Search}(\text{state}) \)  \textbf{returns} an action

**Inputs:**  \( \text{state} \), current state in game

\[
v \leftarrow \text{Max-Value}(\text{state}, -\infty, +\infty)
\]

**Return** the action in \( \text{Successors}(\text{state}) \) with value \( v \)

### Max-Value

**Function**  \( \text{Max-Value}(\text{state}, \alpha, \beta) \)  \textbf{returns} a utility value

**Inputs:**  \( \text{state} \), current state in game

- \( \alpha \), the value of the best alternative for MAX along the path to \( \text{state} \)
- \( \beta \), the value of the best alternative for MIN along the path to \( \text{state} \)

**if**  \( \text{Terminal-Test}(\text{state}) \)  \textbf{then}  \textbf{return}  \( \text{Utility}(\text{state}) \)

\[
v \leftarrow -\infty
\]

**for**  \( \alpha, s \)  \textbf{in}  \( \text{Successors}(\text{state}) \)  \textbf{do}

\[
v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta))
\]

**if**  \( v \geq \beta \)  \textbf{then}  \textbf{return}  \( v \)

\[
\alpha \leftarrow \text{Max}(\alpha, v)
\]

**return**  \( v \)

### Min-Value

**Function**  \( \text{Min-Value}(\text{state}, \alpha, \beta) \)  \textbf{returns} a utility value

**Inputs:**  \( \text{state} \), current state in game

- \( \alpha \), the value of the best alternative for MAX along the path to \( \text{state} \)
- \( \beta \), the value of the best alternative for MIN along the path to \( \text{state} \)

**if**  \( \text{Terminal-Test}(\text{state}) \)  \textbf{then}  \textbf{return}  \( \text{Utility}(\text{state}) \)

\[
v \leftarrow +\infty
\]

**for**  \( \alpha, s \)  \textbf{in}  \( \text{Successors}(\text{state}) \)  \textbf{do}

\[
v \leftarrow \text{Min}(v, \text{Max-Value}(s, \alpha, \beta))
\]

**if**  \( v \leq \alpha \)  \textbf{then}  \textbf{return}  \( v \)

\[
\beta \leftarrow \text{Min}(\beta, v)
\]

**return**  \( v \)