# Grundlagen der Künstlichen Intelligenz

Programming Exercise 3: Propositional Logic

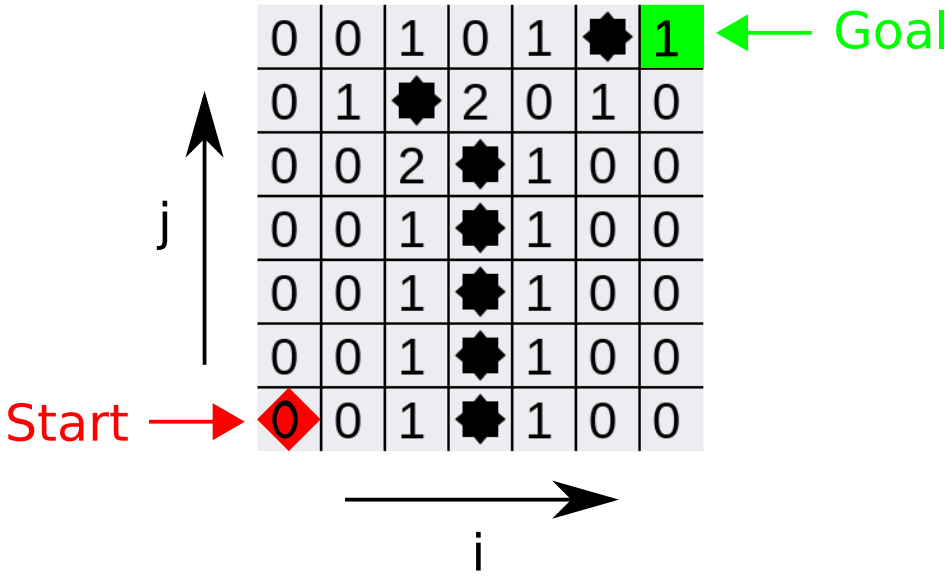
Niklas Kochdumper, Matthias Kübrich and Furkan Mert Algan

(last updated 14th December 2018)

Submission deadline: 10th January 2019, 23:59

## Problem 3: Minesweeper

This exercise is based on the computer game classic *Minesweeper*. Given a square grid with hidden mines, the goal is to navigate an agent from the start to the goal position while avoiding all fields containing mines. Since the agent gets as a perception the number of mines located in the fields that neighbor his current position, this problem can be solved with propositional logic.




### Problem Description

To solve the task with propositional logic, the problem is modeled with the following propositional variables:

- $B_{ij}$: There is a mine in field $ij$
- $Z_{ij}$: There are no mines in fields that neighbor filed $ij$
- $O_{ij}$: There is exactly one mine in a field that neighbors field $ij$
- $TW_{ij}$: There are exactly two mines in fields that neighbor field $ij$
- $TH_{ij}$: There are exactly three mines in fields that neighbor field $ij$
- $F_{ij}$: There are exactly four mines in fields that neighbor field $ij$

Further, we have the following knowledge about the environment:

- The index $ij$ refers to the $(j-1)$-th filed from the bottom in the $(i-1)$-th column (indexing starts with 0).
- The environment is a square grid with 7 fields in each direction.
- Neighboring fields are only the four fields touching at the faces, meaning the neighboring fields for field $ij$ are $(i-1)j$, $(i+1)j$, $i(j-1)$, and $i(j+1)$.

- If the agent stands on field $ij$, he obtains as a perception the number of mines located in neighboring fields (so either $Z_{ij}$, $O_{ij}$, $TW_{ij}$, $TH_{ij}$ or $F_{ij}$) as well as the knowledge if there is a mine on the field or not (so either $B_{ij}$ or $\neg B_{ij}$).
- There is no mine in field 00 ($\neg B_{00}$).
- There is no mine in field 66 ($\neg B_{66}$).
- The start position of the agent is the field 00.
- The goal position is the field 66.

Your task for this programming exercise is to design a knowledge base that describes all information about the environment with the propositional variables from above. If the knowledge base is specified correctly, then the agent will be able to safely navigate through the minefield.

**Programming Framework**

For this programming exercise *Jupyter Notebooks* will be used. The template for the exercise can be found in Moodle. Since you only have to implement one single function, only minor programming skills in Python are necessary to complete this exercise. The following steps are required to correctly set up the environment for the programming exercise:

1. **Installation of Anaconda:** If you do not already have the *Jupyter Notebook* environment installed on your machine, the installation is the first step you have to perform. We recommend to install *Anaconda*, since this will set up the whole environment for you. Instructions on how to install *Anaconda* can be found in the "*AIMAcode Installation Instructions*" file in Moodle.
2. **Download of the AIMA python code:** The template for the programming exercise is based on the code from the *AIMAcode* project. Therefore, you first have to download the code from this project before the template can be used. Instructions on how to obtain the code from this project can be found in the "*AIMAcode Installation Instructions*" file in Moodle.
3. **Download of the template:** Download the .zip-folder with the template from Moodle. To avoid issues with the relative file paths, we recommend to copy all files contained in the .zip-folder into the root-directory of the *AIMAcode* project that you downloaded in the previous step.

After completing the above steps, you are all set up to start with the exercise. The main function of the template is the *Jupyter Notebook* **Exercise.ipynb**, which is also the only file you have to work on. The notebook contains the empty function `generate_knowledge`. Your task is to implement the knowledge base you created into this function. An example on how to add propositional sentences to the knowledge base in such a way that they are compatible with the remaining code of the template is shown in the function `generate_knowledge_example`. If you execute the notebook the agent will start to navigate through the minefield while relying on the knowledge base you specified. You can see a visualization of the minefield as well as the input and the output of the inference algorithm at the bottom of the notebook, which should help you to debug your knowledge base. In addition, there are multiple different minefields available on which you can test your knowledge base. If your knowledge base is correct, then the agent is able to successfully reach the goal position without stepping on a mine.

### Inference Algorithms

The templates contains the two inference algorithms *Forward Chaining* and the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm. We recommend to choose from these two algorithms the one that is better suited for the structure of the knowledge base you defined since this leads to a potentially significant speed-up for inference:

- **Forward Chaining:** The implementation of the *Forward Chaining* algorithm used in the template can only handle propositional sentences with the following structure:
  - $\alpha_1 \wedge \cdots \wedge \alpha_n$
  - $\alpha_1 \wedge \cdots \wedge \alpha_n \Leftrightarrow \beta_1 \wedge \cdots \wedge \beta_m$
  - $\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta_1 \wedge \cdots \wedge \beta_m$,

  where $\alpha_i$ and $\beta_i$ are either a propositional variable $S$ or the negation of a propositional variable $\neg S$.
- **DPLL:** The *DPLL* algorithm is another algorithm for inference in propositional logic which was not part of the lecture. If you are curious to discover how the algorithm works you can check out e.g. `https://en.wikipedia.org/wiki/DPLL_algorithm`. The algorithm is able to handle knowledge bases with arbitrary structure.

### Submission

For the submission, you have to upload in Moodle the **Exercise.ipynb** file containing your implementation of the knowledge base. We will test on several minefields different to the ones that were provided for debugging if the agent safely reaches the goal position with your implementation of the knowledge base. If this is the case, you successfully completed this programming exercise. As a time limit, the test counts as failed if the agent did not reach the goal state after 5 minutes. We will test all submissions only after the deadline expired, which means that you can update your submission until the deadline.