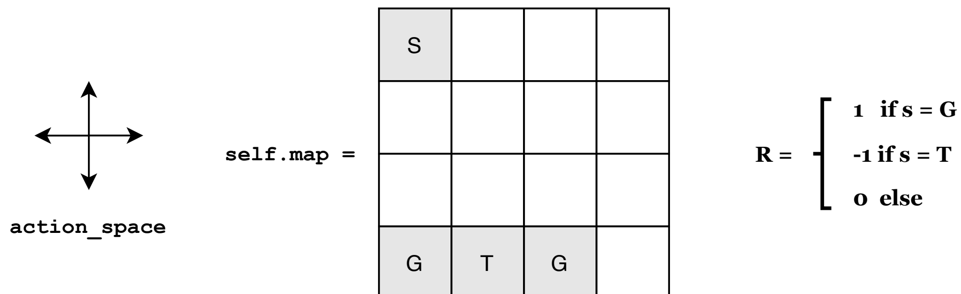
Exercise 3
OpenAI Gym

1 Gym Gridworld

The OpenAI Gym interface is a common standard for environments in reinforcement learning. It defines five methods that your environment has to implement. Adhering to this interface makes your environment compatible with a large number of existing RL algorithms. Documentation can be found here:

- https://github.com/openai/gym
- https://gym.openai.com/docs

In this exercise you will implement an environment yourself, adhering to the OpenAI Gym interface. The environment that you should implement is a simple grid world:



The agent starts in the cell marked **S**. At each time step the agent can step up, right, down, or left. If the agent reaches the goal (**G**), he receives a reward of 1, and the episode ends. If he falls into a trap (**T**), the episode ends with a reward of -1. Actions that would result in the agent leaving the board have no effect.

Programming Tasks:

1. `__init__()`. Each `gym` environment has to define its action and state space via two members, `action_space` and `observation_space`. Define the appropriate spaces in the class initializer. Also, add a member for the current environment state and assign the start state to it.

2. `reset()`. To reset the environment you can call the `reset()` method. This method should reset the environment state to a possible start state, and return the observation for this state

3. `step(action)`. To simulate one time step, the method `step(action)` is called with the action of the agent. This method should simulate the environment dynamics and the agent's actions in one time step.

   `step(action)` should simulate one step in the environment and return a tuple `(observation, reward, done, info)` that describes the transition. `observation` is the information the agent perceives about the new state. `reward` is the reward that the agent is assigned in the new state. `done` is `True` if the next state is a terminal state, `False` otherwise. `info` is a list or dictionary that contains additional (debug) information about the environment – for this implementation, you can just return an empty dictionary ({}).

4. `render()`. This is a helper method that visualizes the current state in a human- or agent-readable way. In our case, this method should print an ascii-representation of the current map and agent.

5. `close()`. This method serves to free up any resources (file handles, sockets, ...) that your environment needs to operate. In our case, the provided implementation (`pass`) does not need to be adapted.

We encourage you to test your implementation to find potential errors and problems before continuing to the second exercise.

2 Temporal Difference Learning

TD-Learning is a technique that allows us to solve MDPs without access to the state transitions P. Your task is to implement a TD-Learning-Agent that solves the environment you implemented in Exercise 1. The file `td_agent.py` contains the skeleton code for this exercise. We supply testing code that already deploys a random policy, and visualizes the value function.

Programming Tasks

1. `learn(n_timesteps)`. This method currently implements `n_timesteps` of environment interaction, by selecting a random action and deploying it in the environment. Implement TD-Learning and update `self.V` at every time step. Run the script and examine the V-estimates.

2. **Policy** `action(s)`. Currently, this method implements a random policy. Manually implement a better policy (e.g., by mapping states to actions with a dictionary) and observe how the TD-Estimate of V changes visually.