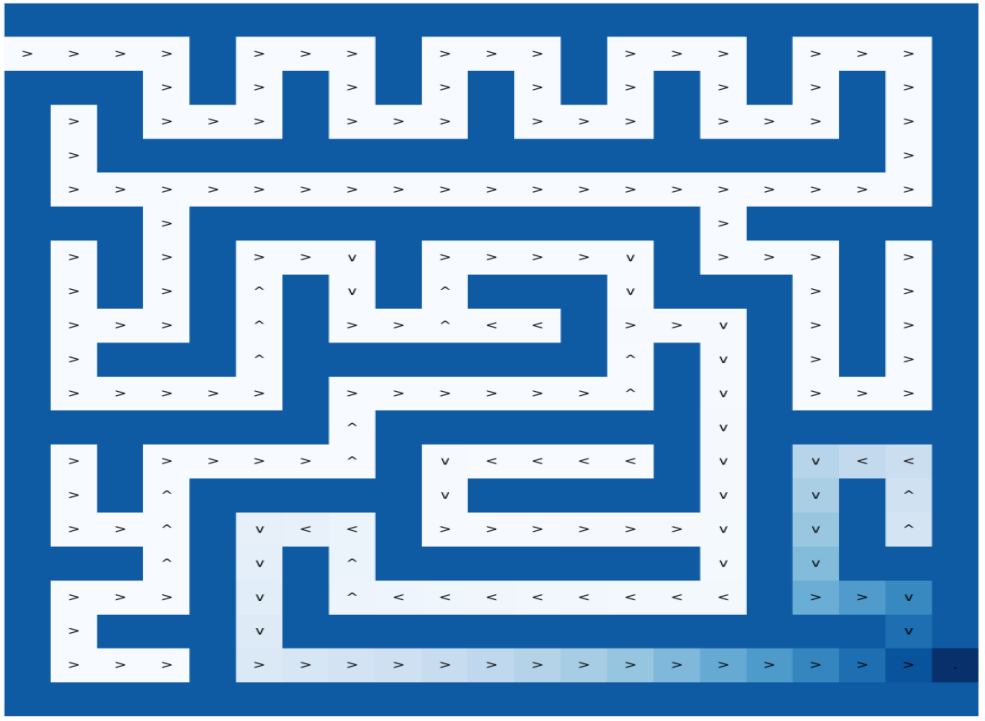
Parallel programming

Homework 3
Value iteration on GPU



FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Motivation

Student Theodor, after a nice evening party, is trying to find his way home. The next day, he has an important exam that he needs to pass, so he would like to get home as soon as possible. However, because he is in a good mood, his coordination is not the best. This is not the only complication; the city is full of open bars, and Theodor is not able to resist having a beer if he enters one of them.

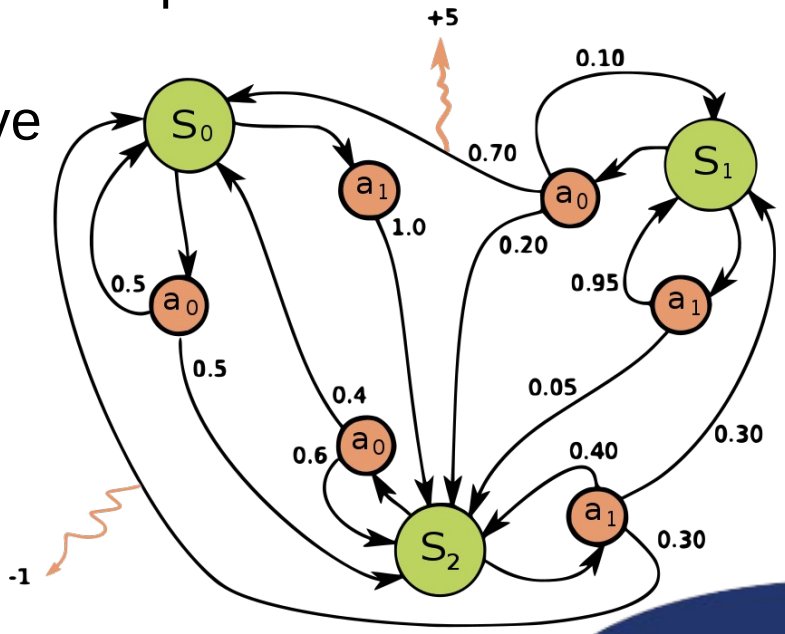
Help Theodor find his way to his bed.

Markov Decision Process (MDP)

- Optimal control in discrete-time stochastic system
- Set of states and actions
  - Finite set of states **S**
  - Finite set of actions **A**
- At each time step, the process is in some state s
- Decision maker may choose any action ***a*** that is available in state ***s***
- The process randomly with some propability moves into a new state ***s′***

Formal definition of MDP

- Markov decision process is a 5-tuple (**S**, **A**, **P**, **R**, **γ**)
  - **S** is a set of states called the **state space**
  - **A** is a set of actions called the **action space**
  - **R(*s*, *a*, *s'*)** is the reward received after transitioning from state ***s*** to ***s'*** due to action ***a***
  - **P(*s*, *a*, *s'*)** is the probability of the fact that taking the action ***a*** in state ***s*** at time step *t* will lead to state ***s'*** at time step *t+1*
  - ***γ*** is discount factor representing the importance of future rewards relative to immediate rewards
- The goal is to find an optimal policy ***π****
  - Given some state, **the policy** returns an action to perform in this state
    - ***π***: **S** → **A**
  - **Optimal policy** is the policy which maximizes **the long-term reward**



Value iteration

- Value iteration is an iterative algorithm based on Dynamic Programming
- Uses the Bellman optimality equation, we can compute an expected reward for each state V(**s**)
- 1. Initialization: $V^0(s) = 0$
- 2. Iterative update for non-terminal state

$$V^{n+1}(s) = max_{a \in A}\{\sum_{s' \in S} P(s,a,s') * [R(s,a,s') + \gamma * V^n(s')]\}$$

- 3. Convergence check
  - Stop iterate when the difference between consecutive iterations is under given epsilon

$$\max(V^{n-1} - V^n) \leq epsilon$$

- 4. Retrieve policy

Continuation of the story

For this problem, assume that Student Theodor moves in a grid (maze) world where there are tiles (cells) representing empty space, walls, pubs, and student's home(s). Because his movement abilities are limited, when he wants to move in a certain direction, there is an 80% probability that he moves in the selected direction, a 10% probability that he moves to the left of the intended direction, and a 10% probability that he moves to the right of the intended direction. When Theodor tries to move in a direction where there is a wall, the action is valid, but the wall stops him. So we can interpret as he did not change his position.

The agent is motivated by the dream of getting home. There is a reward for reaching his home (goal reward). Additionally, there is a negative reward for each action that results in him being in a pub (delay reward). If he tries to leave a pub and ends up on a pub tile again, the delay is encountered again. For any other tile, which is not a pub or a goal, there is a small negative penalty as a step reward.

Problem environment

- 2D grid world
  - Terminal states – walls, goal(s)
  - Non-terminal states – empty and delay tiles
- Agent can move in 4 directions (left, right, up, down)
- Desired move has 80% success rate
  - At 80% agent will go to desired direction
  - At 10% agent will move to $+90^{o}$ direction
  - At 10% agent will move to -90^{o} direction
- Walls are inaccessible; trying to move into a wall = staying in the original tile
- Rewards are distinguished for stepping on empty tile, delay tile, and goal tile

Your task

- Implement python code to compute value matrix using value iteration algorithm
- Your uploaded solution must contain `main.py` file
- Use GPU through Numba library

- Initialize the value matrix with 0

Input example

```
7 7 -1 -10 1000 0.999 1e-6
#######
#     #
###D# #
# #   #
#   ###
#    G#
#######
```

```
Layout meaning:
# - wall
D - delay tile
G - goal tile
  - empty tile
```

- Parameters on first line: width height step_reward delay_reward goal_reward gamma epsilon

Output example

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 970.8413155853887 973.3073336224651 975.7764370408019 978.0538663932286 980.5289087988102 0.0
0.0 0.0 0.0 986.3435536093713 0.0 983.3164283866547 0.0
0.0 986.6353570708447 0.0 991.0812207186736 988.5929912499611 985.7980555516679 0.0
0.0 989.1211370474564 991.6684121885596 993.8835922711005 0.0 0.0 0.0
0.0 991.1398383891352 993.8835922711005 996.6553391211104 999.5001249687577 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
Policy
########
#>>>>V#
###V#V#
#V#V<<#
#>>V###
#>>>>.#
########
```