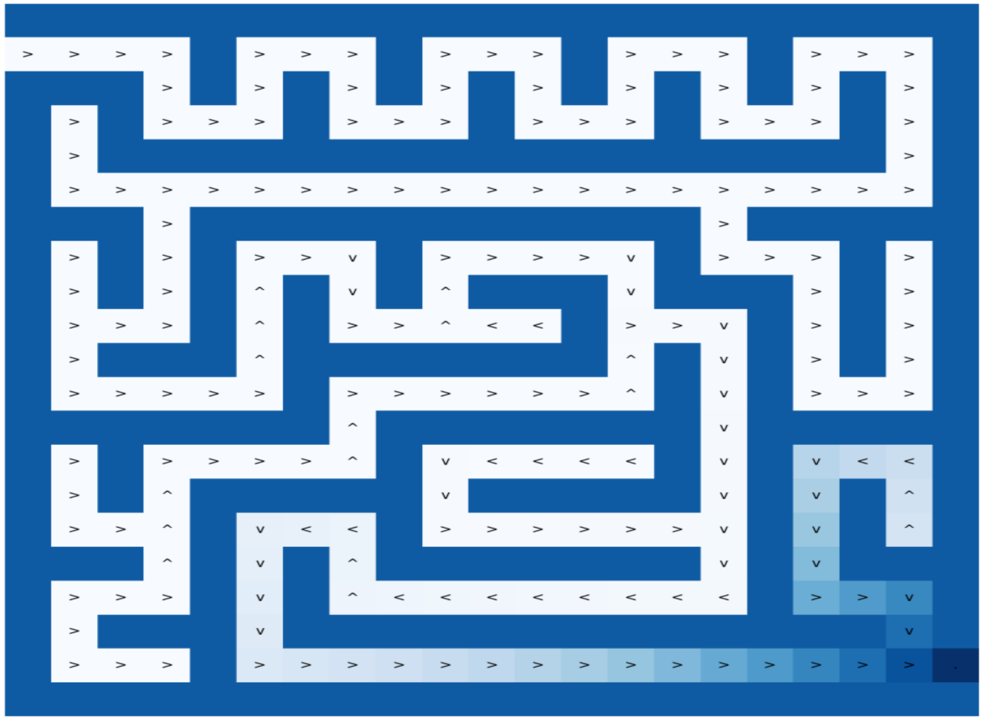
Parallel programming

HW4 assignment



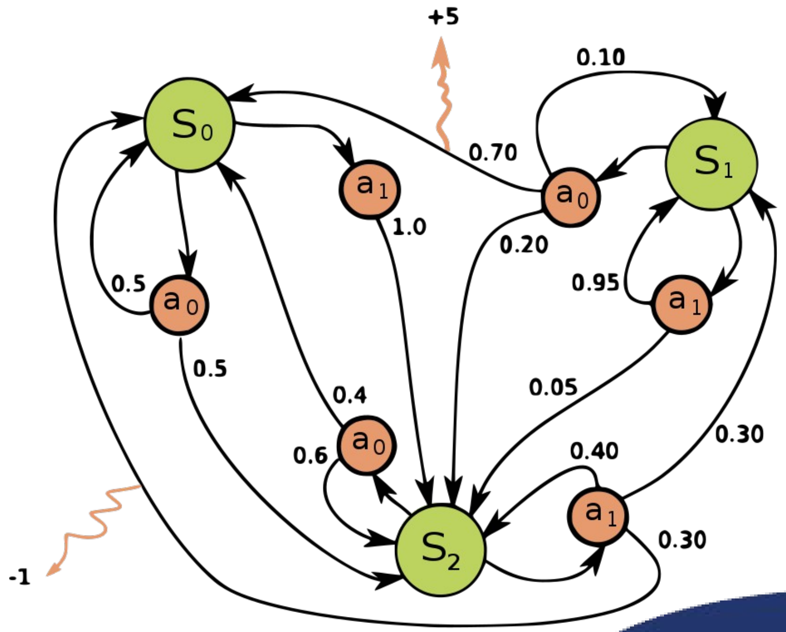
Markov Decision Process (MDP)

- Discrete-time stochastic control process
- Finite sets of states and actions
- At each time step the process starts with some state
- Decision is made among the actions available in the state
- The process randomly moves into a new state

Formal definition of MDP

- Markov decision process is a 4-tuple
  - is a set of states called the **state space**
  - is a set of actions called the **action space** (alternatively)
  - is the reward received after transitioning from state to
  - is the probability of the fact that taking the action in state at time step will lead to state at time step

- Stochastic environment
  - There is a nonzero probability, that action *a* will lead to desired state



Policy definition

- Given some state, the policy returns an action to perform in this state

- ***Optimal policy*** is the policy which maximizes ***the long-term reward***

- ***Our goal*** is to find the optimal policy

Policy Iteration

- ***Policy Iteration*** is an iterative algorithm based on dynamic programming
- It requires to store two arrays:
    - Array of values **V** which contains real values
    - Policy array **π** which contains actions
- At the end of the algorithm, **π** contains the solution and **V** contains the discounted sum of the rewards to be earned
- We are talking about policies instead of actions due to ***stochastic*** behavior of the environment
- ***Three steps*** of the Policy Iteration algorithm:
    - Initialize random policy and actions for every state
    - Policy Evaluation
    - Policy Improvement

Policy evaluation

- Get an action for every state in the policy and evaluate the value function using Bellman's equation:

$$V(s) = \max[a \in A] \{R(s, a) + \gamma * \Sigma[p(s' \mid s, a) * V(s')]\}$$

- ***p(s' | s, a)*** - transition probability from ***s*** to ***s'*** by action ***a***
- ***R(s, a)*** - reward from the current state
- ***V(s)*** (resp. ***V(s')***) - values of state ***s*** (resp. ***s'***)
- ***γ*** is the discount factor in range [0, 1)

Policy improvement

- Get the best action from value function for every state:

$$\pi'(s) = \text{argmax}[a \in A] \{\Sigma[p(s' \mid s, a) * V(s')]\}$$

- ***π'(s)*** is the new policy (optimal action) for state ***s***
- If the optimal action is better than the present policy action, then **replace** the current action by the best action

Policy iteration summary

- Iterate through the policy iteration and policy improvement steps
- If the policy did not change throughout an iteration, then we can consider that the algorithm has ***converged***

Your state space

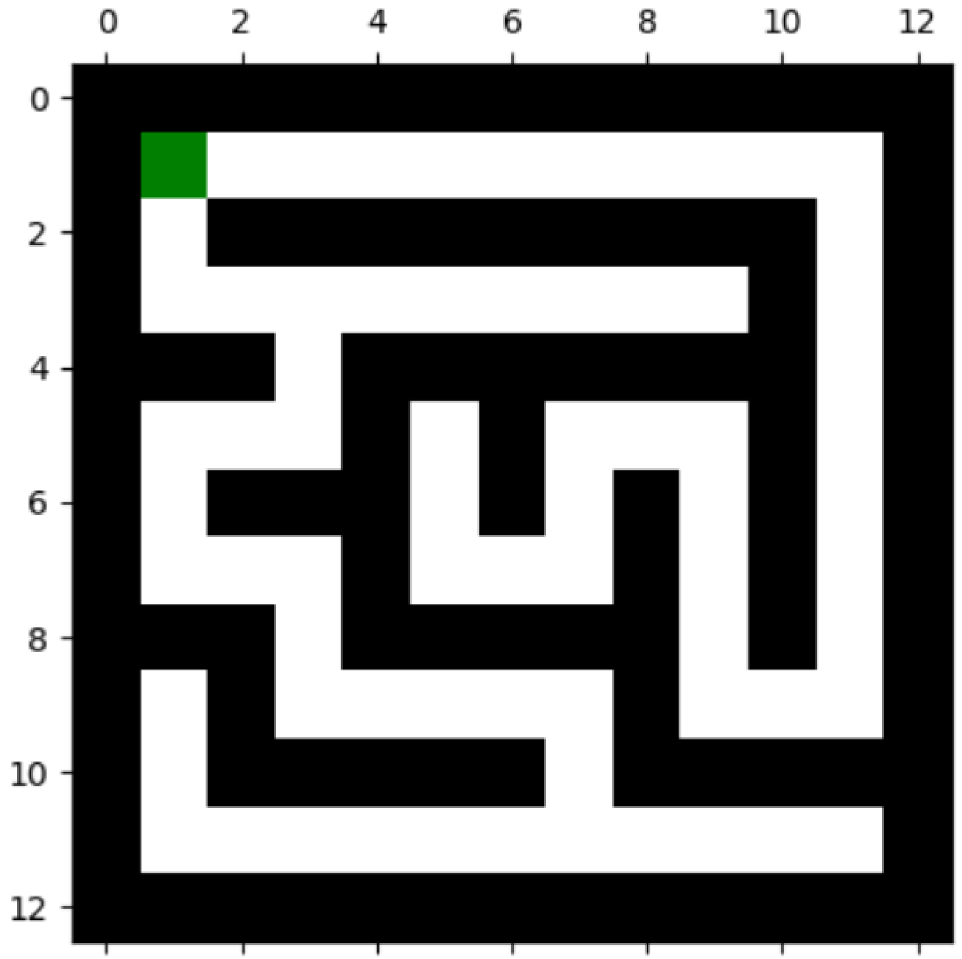
- 2D maze with walls and desired state
- Goal is to find optimal policy that will lead to desired state
- Given an agent (vehicle) with actions
  - Go right
  - Go left
  - Go Up
  - Go Down
- Each action has 80% success rate
  - At 80% vehicle will go to desired direction
  - At 10% vehicle will move to $+90^o$ direction
  - At 10% vehicle will move to -90^o direction
- Only accessible states are other fields of maze, walls are inaccessible

Inputs and outputs

- Input is .txt file where
  - In first line there are 2 integers **w** and **h** representing width and height
  - On the rest **h** lines there are exactly **w** integers of values {0,1,2}, where
    - 0 represents accesible state (field)
    - 1 represents unaccesible state (wall)
    - 2 represents desired state
- Output is .txt file with **h** lines of **w** integers where
  - Each value representing optimal policy at given state
    - 5 is policy for unaccessible states (walls) or final states
    - 0 is „Go Up“
    - 1 is „Go Right
    - 2 is „Go Down“
    - 3 is „Go Left“

Input Example

```
13 13
0 1 1 1 1 1 1 1 1 1 1 1 1
1 2 0 0 0 0 0 0 0 0 0 0 1
1 0 1 1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 0 0 0 1 0 1
1 1 1 0 1 1 1 1 1 1 1 0 1
1 0 0 0 1 0 1 0 0 0 1 0 1
1 0 1 1 1 0 1 0 1 0 1 0 1
1 0 0 0 1 0 0 0 1 0 1 0 1
1 1 1 0 1 1 1 1 0 1 0 1
1 0 1 0 0 0 0 0 1 0 0 0 1
1 0 1 1 1 1 1 0 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1
```



Output Example

```
5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 3 3 3 3 3 3 3 3 3 3 5
5 0 5 5 5 5 5 5 5 5 5 0 5
5 3 3 3 3 3 3 3 3 3 5 0 5
5 5 5 0 5 5 5 5 5 5 5 0 5
5 1 1 1 5 2 5 1 1 2 5 0 5
5 0 5 5 5 2 5 0 5 2 5 0 5
5 0 3 3 5 1 1 0 5 2 5 0 5
5 5 5 0 5 5 5 5 5 2 5 0 5
5 2 5 3 3 3 3 3 5 1 1 0 5
5 2 5 5 5 5 5 0 5 5 5 5 5
5 1 1 1 1 1 1 0 3 3 3 3 5
5 5 5 5 5 5 5 5 5 5 5 5 5
```