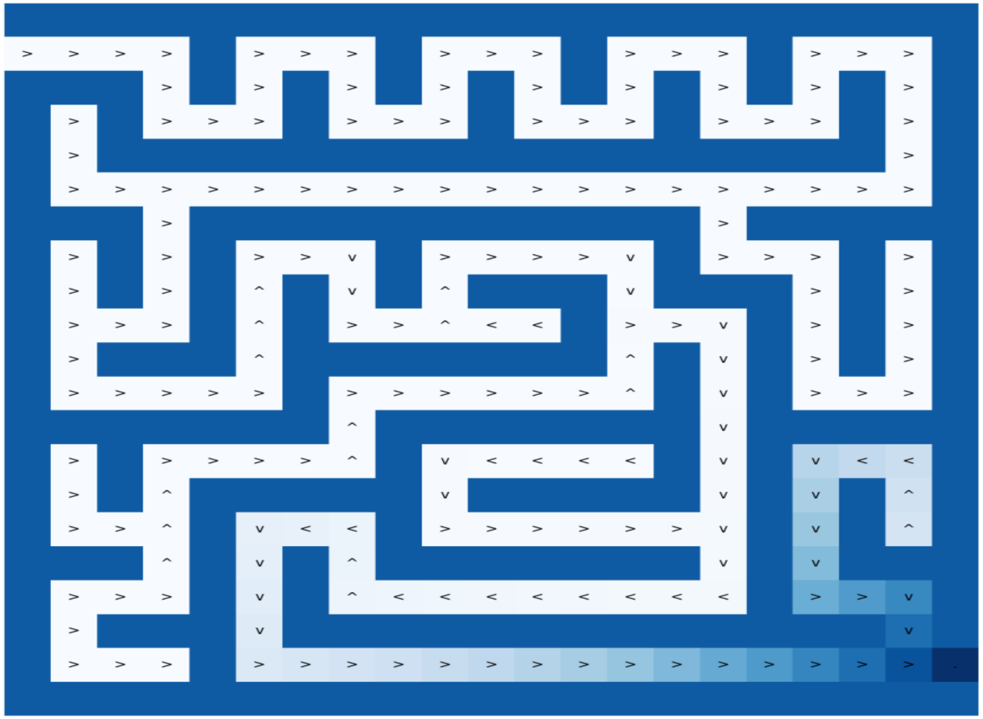
Parallel programming

HW4 assignment

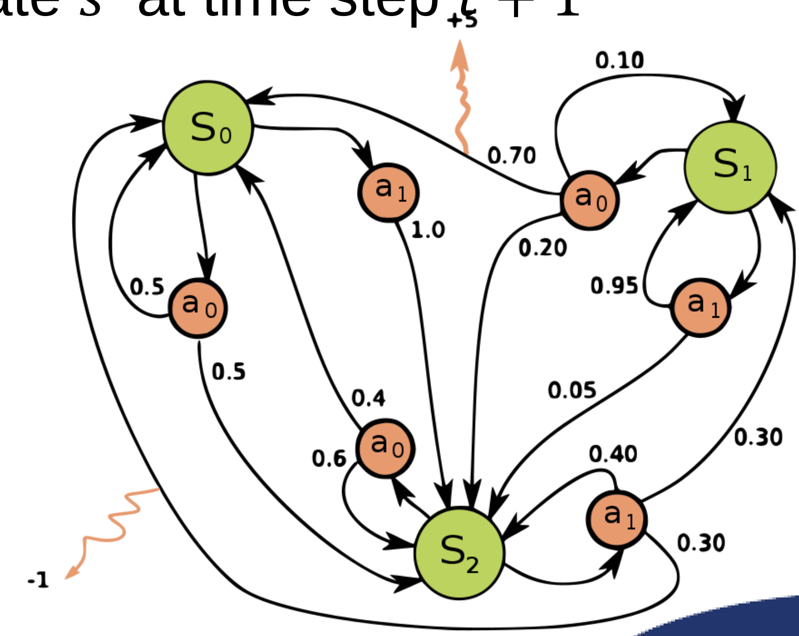


Markov Decision Process (MDP)

- Discrete-time stochastic control process.
- 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 $\boldsymbol{s}$
- Decision maker may choose any action $\boldsymbol{a}$ that is available in state $\boldsymbol{s}$
- The process randomly moves into a new state $\boldsymbol{s}'$

Formal definition of MDP

- Markov decision process is a 4-tuple (S, A, R_a, P_a)
  - $\boldsymbol{S}$ is a set of states called the **state space**
  - $\boldsymbol{A}$ is a set of actions called the **action space** (alternatively A_s)
  - $\boldsymbol{R_a(s, s')}$ is the reward received after transitioning from state s to s' due to action a
  - $\boldsymbol{P_a(s, s')}$ 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 $t + 1$
    - $P(s_{t+1} = s' | s_t = s, a_t = a)$



- 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
  - Reward is based on the chance that policy leads to desired state
- Our goal is to **find that optimal policy**.
- $\pi^*(s) = \arg\max_\pi V^\pi(s)$

Value Iteration

- Value iteration is an iterative algorithm based on Dynamic Programming.
- 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** will contain the solution and **V** will contain the discounted sum of the rewards to be earned.
- We are talking about policies instead of actions because of the **stochastic** behaviour of the environment
- Three steps of value iteration
  1. Initialize state values
  2. Improve values
  3. Extract policy from values

Value Iteration

- Formally: We have a reward function which gives us the rewards for transitioning from one state to another, the state value of all terminal states is zero.
- Simplified: There is no reward function, and the value of the terminal states corresponds to the reward obtained for reaching them
- The simplified version assumes that a reward is obtained only in terminal states and depends only on them
- This task satisfies that assumption

Step 1 - Formally

- Initialize values, arbitrarily for non-terminal states and zero in terminal states

Step 2 - Formally

- Update the value for every non-terminal state using Bellman's equation:
  - $V(s) = \max_{a \in A} \left(\sum_{s'} P_a(s, s') \cdot [\, R_a(s, s') + \gamma \cdot V(s')] \right)$
  - $P_a(s, s')$ is the transition probability from state s to state s' by action a
  - $R_a(s, s')$ is the reward for transitioning from state s to state s' by action a
  - $V(s)$ (resp. $V(s')$) is value of state s (resp. s')
  - γ is the discount factor satisfying $\gamma \epsilon \langle 0,1 \rangle$

Step 1 - Simplified

- Initialize values, arbitrarily for non-terminal states and a reward for reaching the terminal state in terminal states

Step 2 - Simplified

- Update the value for every non-terminal state using simplified Bellman's equation:
  - $V(s) = \gamma \cdot \max_{a \in A}(\sum_{s'} P_a(s, s') \cdot V(s'))$
  - $P_a(s, \mathrm{s}')$ is the transition probability from state s to state s' by action a
  - $V(s)$ (resp. $V(s')$) is value of state s (resp. s')
  - γ is the discount factor satisfying $\gamma \epsilon \langle 0,1 \rangle$
    - Use e.g. γ = 0.99

Step 3

- For every state, get the best action from value function as
  - $\pi(s) = \text{argmax}_{a \in A}\{\sum_{s'} P(s'|s,a) \cdot V(s')\}$
  - $\pi(s)$ is a new policy (optimal action for state s)

Value iteration algorithm

- Repeat step 2 until convergence (difference between old and new value is smaller than some δ).
- Extract optimal policy from converged values (step 3)

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
- Trying to move into a wall = staying in place

Your task

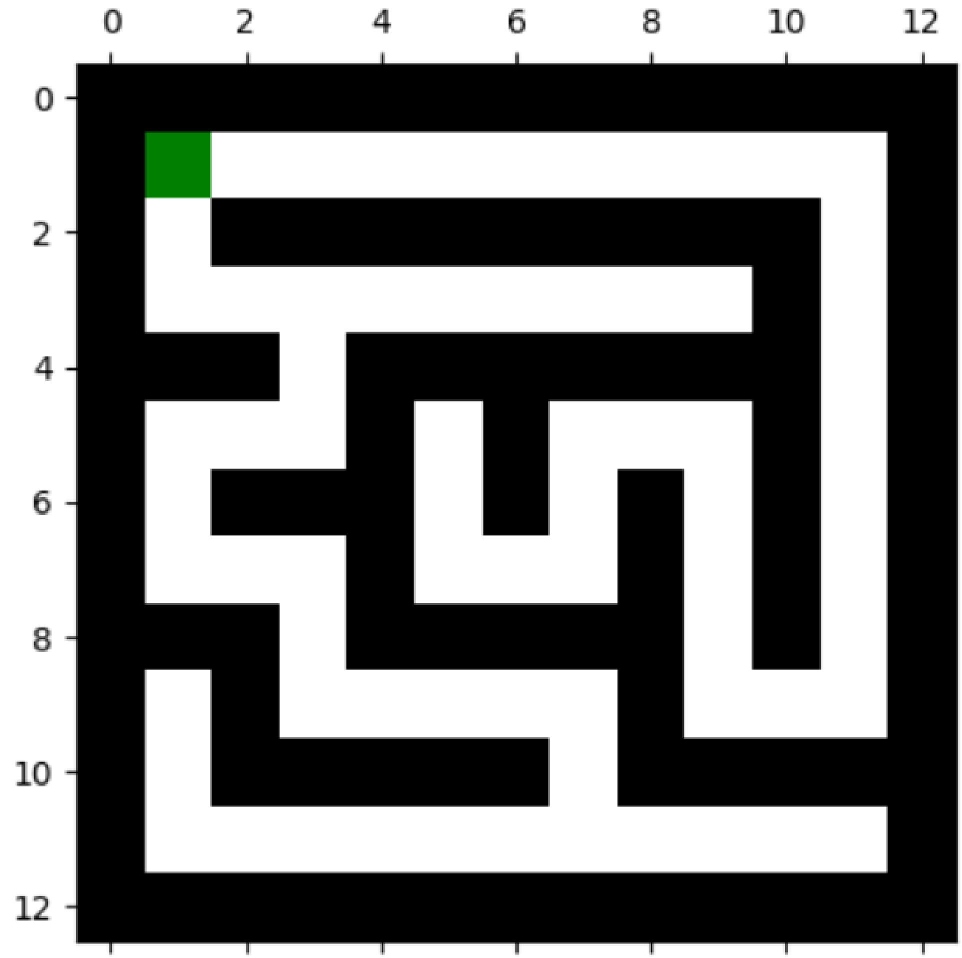
- Find optimal policy for given maze
- Use GPU with Numba library
- You can use provided maze generator to get another instances

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
    - 0 is „Go Up"
    - 1 is „Go Right
    - 2 is „Go Down"
    - 3 is „Go Left"
    - 5 is policy for unaccessible states (walls)
    - 6 is for final state

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 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
```