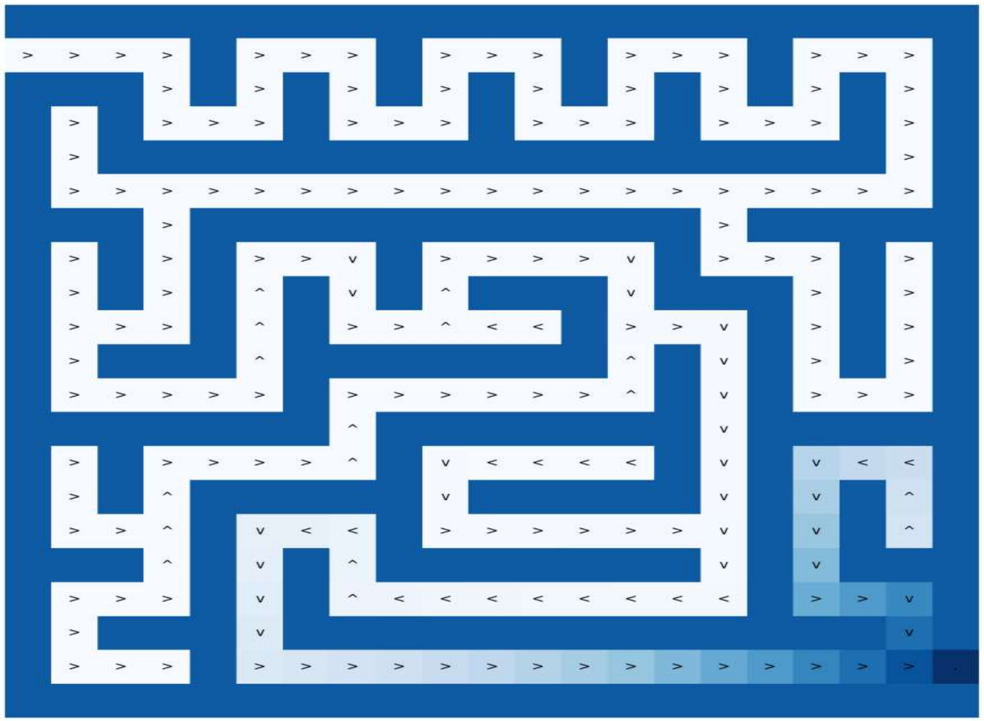
# Parallel programming

# HW4 assignment

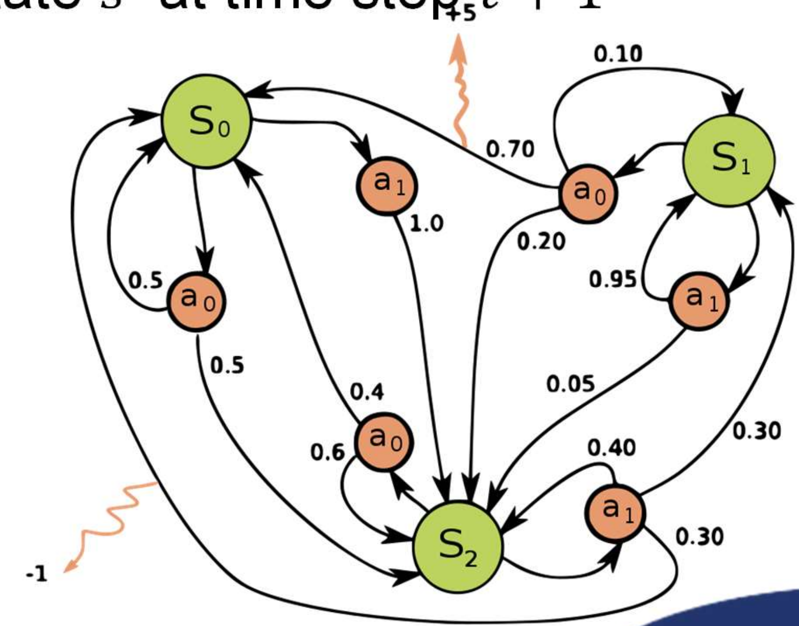


# Markov Decision Process (MDP)

- 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 $A_s$)
  - $\boldsymbol{R_a}(\boldsymbol{s}, \boldsymbol{s'})$ is the reward received after transitioning from state $s$ to $s'$
  - $\boldsymbol{P_a}(\boldsymbol{s}, \boldsymbol{s'})$ 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**.

# Policy Iteration

- Policy iteration is an iterative algorithm based on Dynamic Programing.
- 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 **stochastic** behavior of the environment
- Three steps of policy iteration
  1. Initialize random policy
  2. Policy Evaluation
  3. Policy Improvement

# Step 1

- Randomly initialize the policy.
- Randomly initialize actions at every state of the system

# Step 2

- Get an action for every state in the policy and evaluate the value function using Bellman's equation:
  - $V(s) = r(s) + \gamma \cdot \max_{a \in A}(\sum_{s'} P(s'|s,a) \cdot V(s'))$
  - $P(s'|s,a)$ is transition probability from state $s$ to state $s'$ by action $a$
  - $r(s)$ is reward of current state $s$
  - $V(s)$ (resp. $V(s')$) is value of state $s$ (resp. $s'$)
  - $\gamma$ is is the discount factor satisfying $\gamma \epsilon \langle 0,1 \rangle$

# 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 $s$)
- If the optimal action is better than the present policy action, then **replace** the current action by the best action

# Policy iteration algorithm

- Iterate through the steps 2 and 3, until convergence.
- 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

# Your task

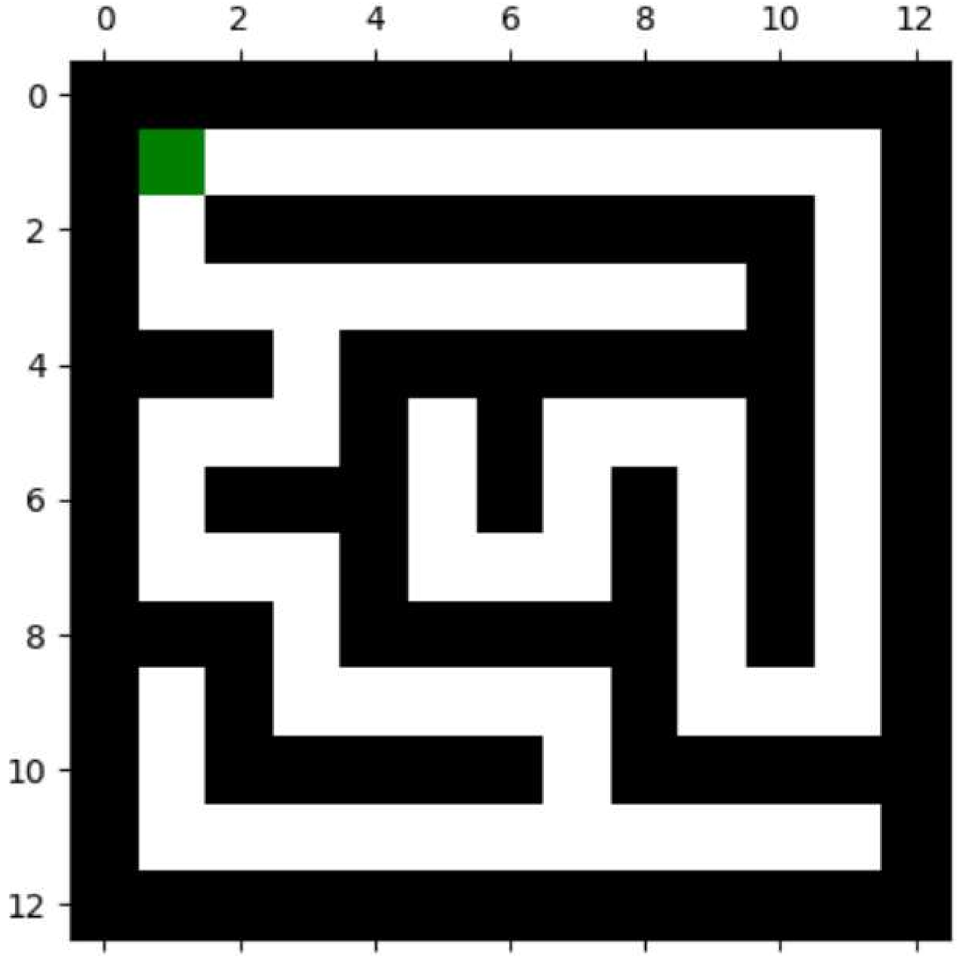
- Find optimal policy for given maze
- Use CUDA GPU with Numba library
- Use provided maze generator to get larger instances
- Evaluation
  - Jupyter notebook with python scripts and analysis
  - Graph 1: Speedup of parallel GPU version (scalability graph)
  - Graph 2: Showing the algorithm runtime based on the size of an input (performance graph)
  - Explain what was the most complicated part and why the results are as provided.
  - What is the limiting factor of the parallelization in your algorithm

# 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 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
```