(Don't) Plan To Escape the Maze

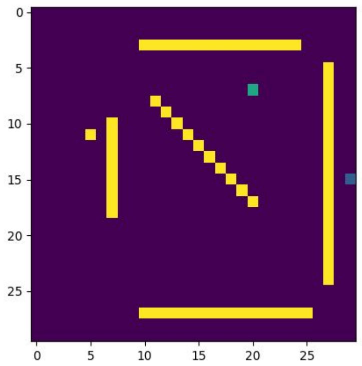
Konstantin Pakulev, Mohammed Deifallah,
Hekmat Taherinejad, Pavel Krasnik

Project objectives

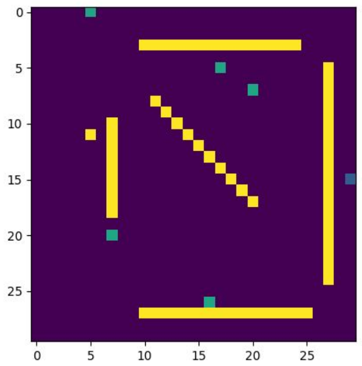
PS4 (*Escape the Maze*) is taken as a basis for our work

- Implement different planning algorithms
    - Value Iteration
    - Markov Decision Process
    - Monte-Carlo Tree Search
- Implement different policies for pursuers
- Explore their performance under
    - Different **number** of pursuers
    - Various **operating logic** of pursuers

Easy mode



Hard mode



Value Iteration

Method formulation:

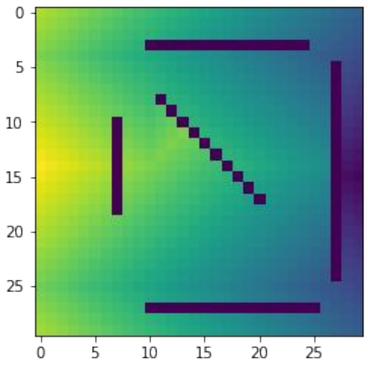
$$G_k^*(x_k) = \min_{u_k} \{l(x_k, u_k) + G_{k+1}^*(x_{k+1})\}$$

$$\text{where } x_{k+1} = f(x_k, u_k)$$

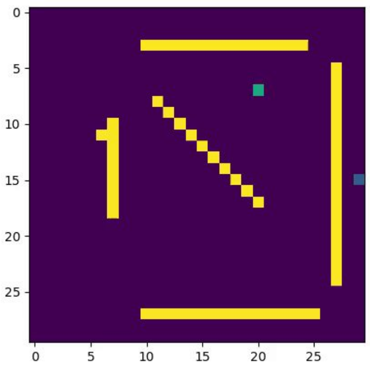
optimal cost-to-go

$$u^* = \arg \min_{u \in U(x)} \{l(x,u) + G^*(f(x,u))\}$$

recover the optimal plan



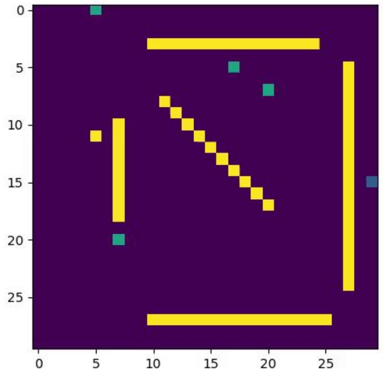
$G^*(x)$



VI (gets caught) playout

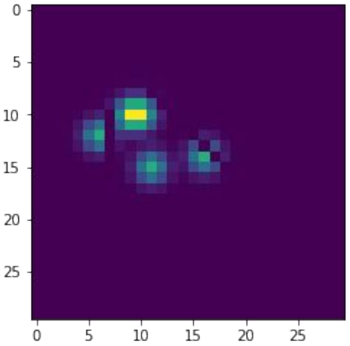
Improved VI

To avoid getting caught **the cost of the pursuers** should be taken into account

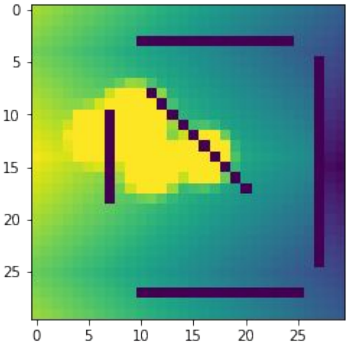


VI + **pursuer cost (escapes)** playout

Improved Value Iteration



Pursuers cost smoothed by a gaussian kernel

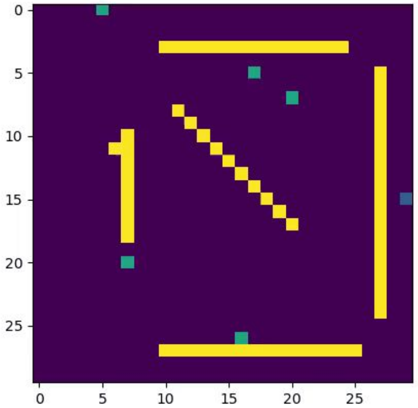


$G^*(f(x,u)) + P(f(x,u))$

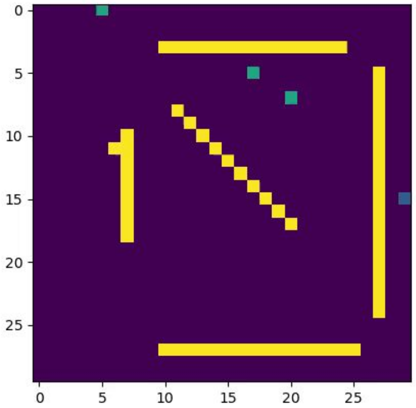
The recovery of the “optimal” plan then becomes:

$$u^* = \arg \min_{u \in U(x)} \left\{ G^*(f(x,u)) + P(f(x,u)) \right\}$$

Failure examples



VI + **pursuer cost (gets caught)** playout



VI + **pursuer cost (gets stuck)** playout

Markov Decision Process

Method formulation:

Bellman optimality equation:

$$v_*(s) = \max_{u \in U(s)} q_*(s,u) = \max_{u \in U(s)} \sum_{s',r} p(s',r|s,u)\left[r + \gamma v_*(s')\right]$$

After which the policy can be recovered by

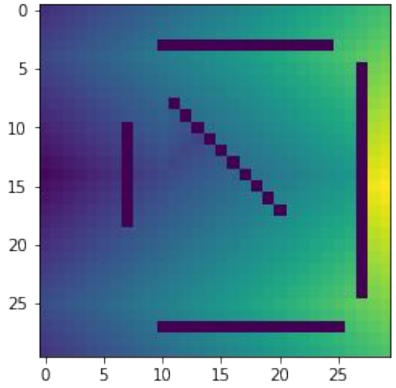
$$u^* = \arg \max_{u \in U(s)} q_*(s,u)$$

For our problem we simplified the equation to:

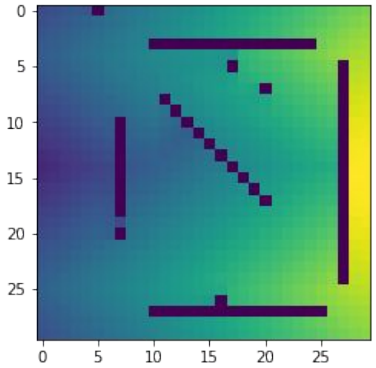
$$v_*(s) = r + \gamma \max_{u \in U(s)} \sum_{s'} p(s'|s,u) v_*(s')$$

And the policy is changed to:

$$u^* = \arg \max_{u \in U(s)} - q_*(s,u) + R(s,u)$$

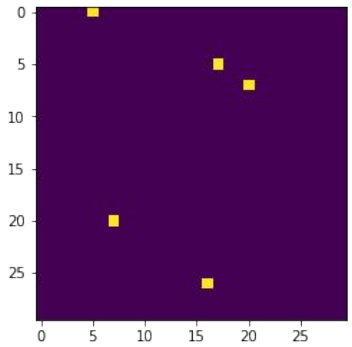


$R(s,u)$

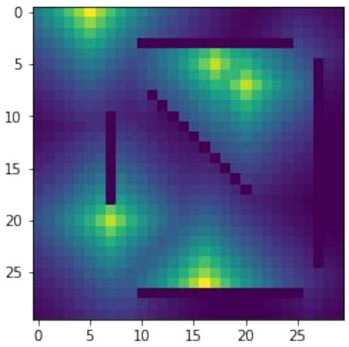


Solving the Bellman equation doesn't influence **pursuers**

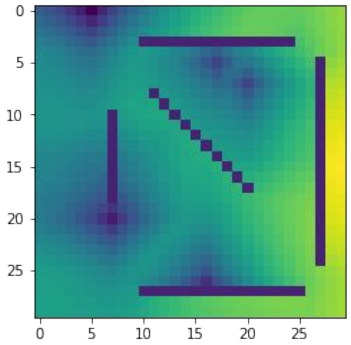
Markov Decision Process. Pursuers



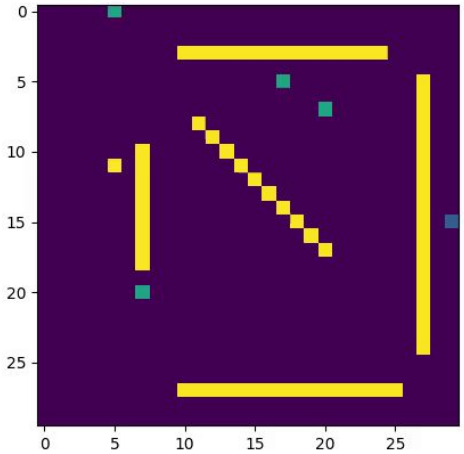
Locations of **pursuers**



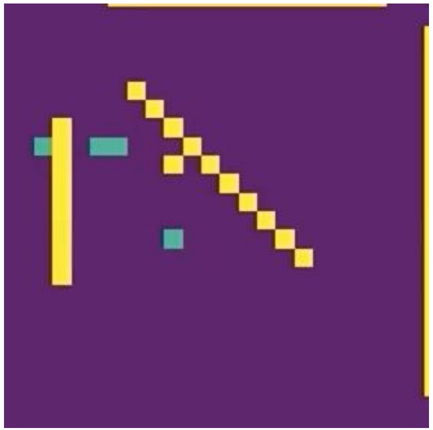
$q_*(s,u)$



$-q_*(s,u) + R(s,u)$

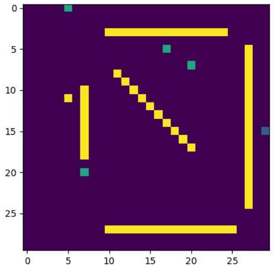


MDP (escapes)
playout



MDP tricks the
pursuer to escape

MDP (gets stuck)
playout:



Monte-Carlo Tree Search

Reward function from the PS4:

$$R_T(x_e, x_p) = \begin{cases} 0 & \text{if } x_e = x_p \\ 100 + \sum_t^T \frac{0.1}{d(x_e(t) - x_{goal})} - T & \text{if } x_e = x_{goal} \\ \sum_t^T \frac{0.1}{d(x_e(t) - x_{goal})} - T & \text{otherwise} \end{cases}$$

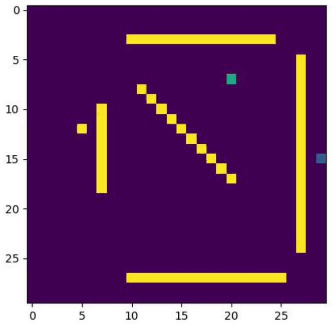
Simulations with ***high number of iterations*** in our environment have **high chance to fail** providing **no information**

Solutions:

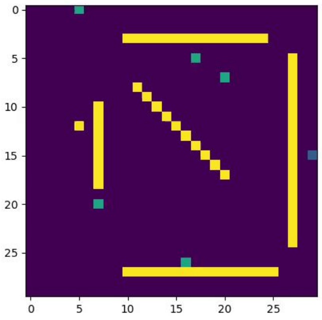
- **Relax** the failure reward
- Make **shorter simulation**

In our case the reward is modified to:

$$R_T(x_e, x_p) = \begin{cases} -100 + \sum_t^T \frac{0.1}{d(x_e(t) - x_{goal})} & if \;\; x_e = x_p \\ 100 + \sum_t^T \frac{0.1}{d(x_e(t) - x_{goal})} - T & if \;\; x_e = x_{goal} \\ \sum_t^T \frac{0.1}{d(x_e(t) - x_{goal})} - T & \text{otherwise} \end{cases}$$

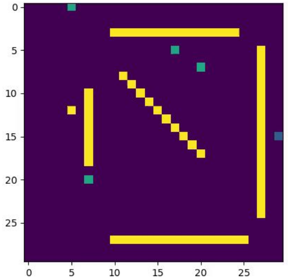


MCTS **(escapes)** playout



MCTS **(escapes)** playout with less simulation iterations

MCTS **(gets stuck)** playout:



Deep Q-Network

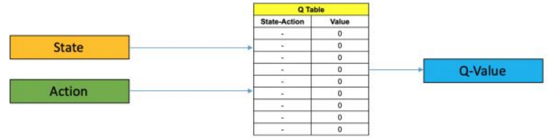
In the Q-learning algorithm, we compute the Q-table which contains the Q-values of any state-action pair using the Q-value iteration. In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output.

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

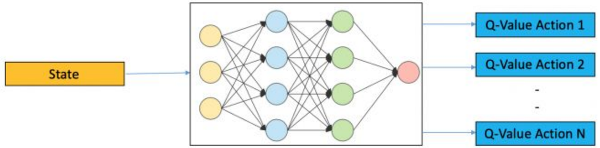
$$Q(s,a) \rightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a) \dots \dots \dots \gamma^n Q(s''^{\dots n},a)$$

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho(.)}\left[(y_i - Q(s,a;\theta_i))^2\right] \text{ where } y_i = r + \gamma \max_{a'} Q(s',a';\theta_{i-1})$$

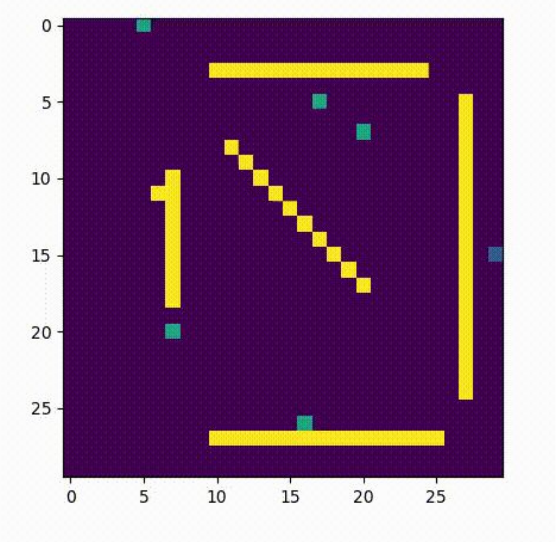


Q Learning



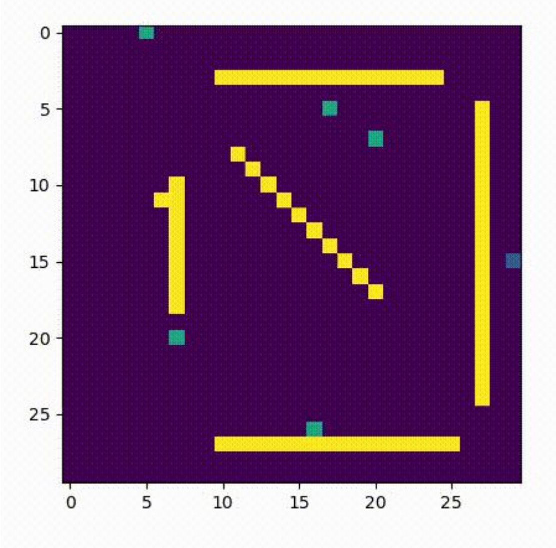
Deep Q Learning

Variants of pursuers



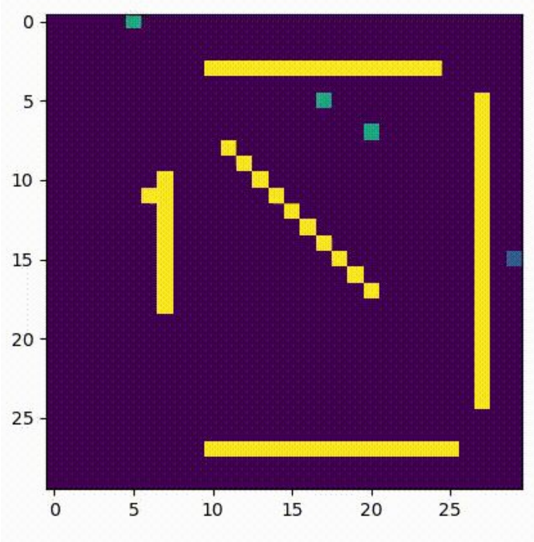
Pursuers with **heuristic (manhattan distance)**

$$d(x,y) = \sum_{i=0}^{n} |x_i - y_i|$$



Pursuers with **heuristic (euclidean distance)**

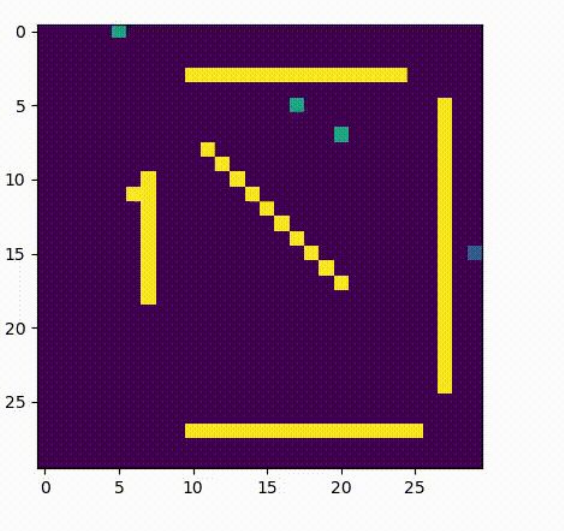
$$d(x,y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$



Pursuers with **Probabilistic Roadmap**

Sample-based. Agents **act as a whole**

Probabilistic Roadmaps (Pursuer)



Idea:

1. The agent is the set of pursuers as whole.
2. Sample C-space.
3. Get approximate result via graph search.
4. At proximity use accurate algorithm (A*).

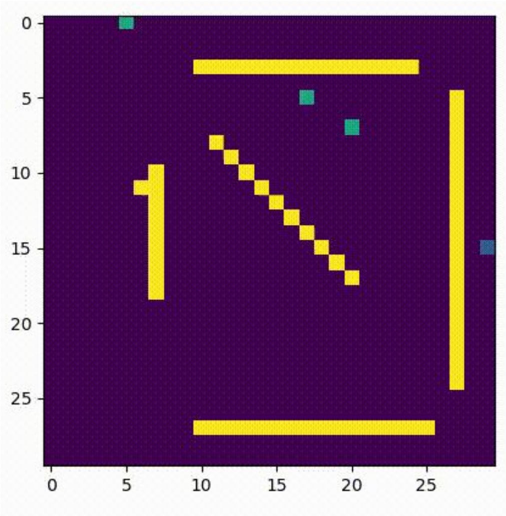
Disadvantages:

- Curse of dimensionality: we need ~5^6 samples for 3 pursuers
- Too much for graph search
- Long initialization
- Random process

Probabilistic Roadmaps (Pursuer)

Advantages:

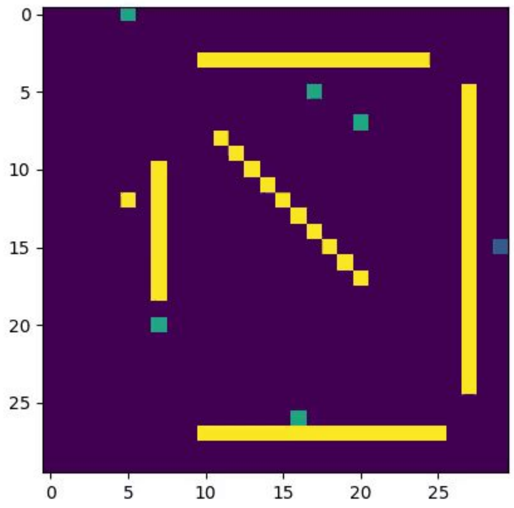
1. Simplification of space.
2. Easy to configure sophisticated constraints
3. Due to randomness result looks natural



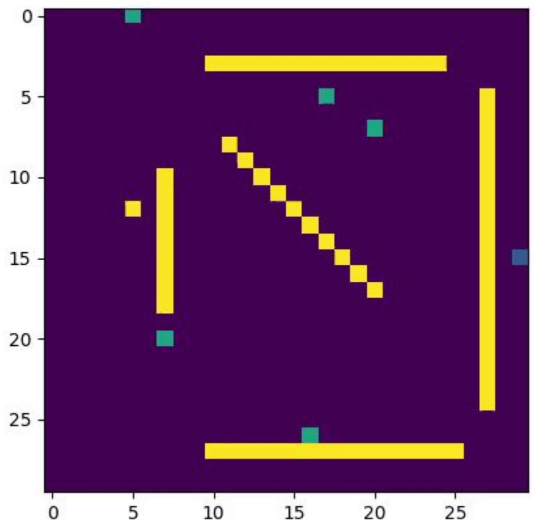
Conclusion:

- Very niche algorithm
- Using lattice instead of uniform looks promising

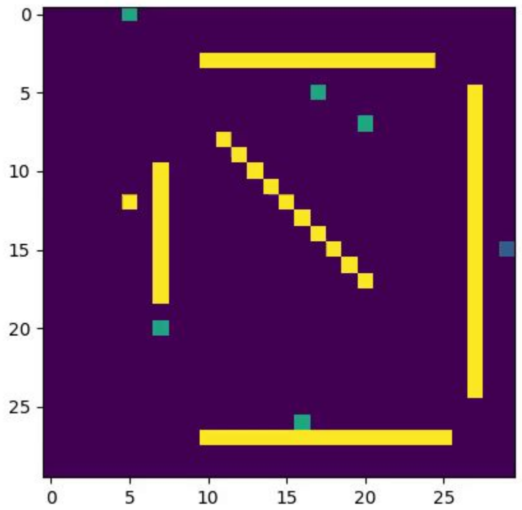
(Not) Escaping the maze



MVI (gets caught)
vs
Pursuers with heuristic
(euclidean distance)



MDP (gets caught)
vs
Pursuers with heuristic
(euclidean distance)



MCTS (gets caught)
vs
Pursuers with heuristic
(euclidean distance)

Conclusions

- In the presence of **pursuers** with a just a little **more complex logic** the **difficulty increases dramastically**
- For both **MDP** and **MCTS** the choice of the **reward function** significantly changes the **behaviour** of the **escaper**
- **MDP** is not very suitable for **long term minimization**/**maximization** planning
- **MCTS** is noticeably faster than **MDP**