Line-of-sight optimization for moving agents

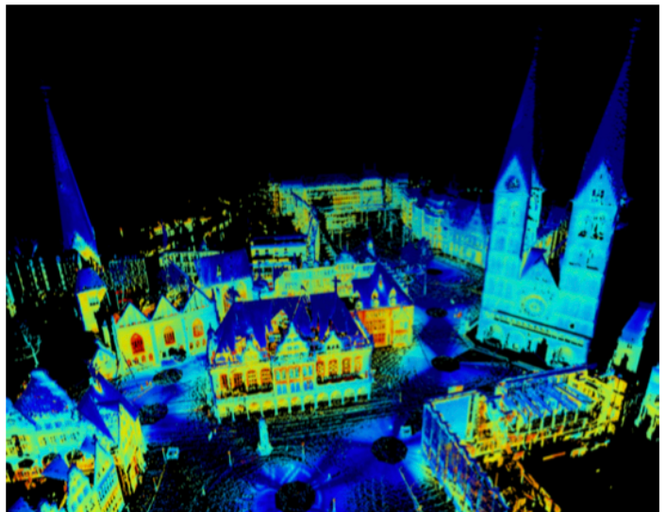
Louis Ly and Richard Tsai

The University of Texas at Austin

Research supported partially by the National Science Foundation and Simons Foundation

Overview

- Increasingly available range sensors for sampling the environment or the position of an adversary agent (lines-of-sight or depth field data)
- Autonomous robotic agents (e.g. UAVs) performing tasks involving line-of-sight “objectives”
  - exploration and surveillance
  - vision-based tracking (and learning) of moving targets
- Problems become challenging in complex environments
- Learn the dependence on the domain shape

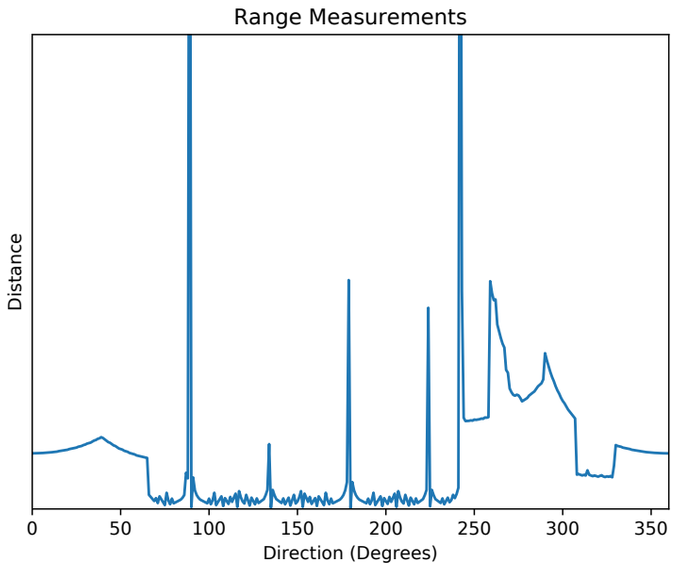


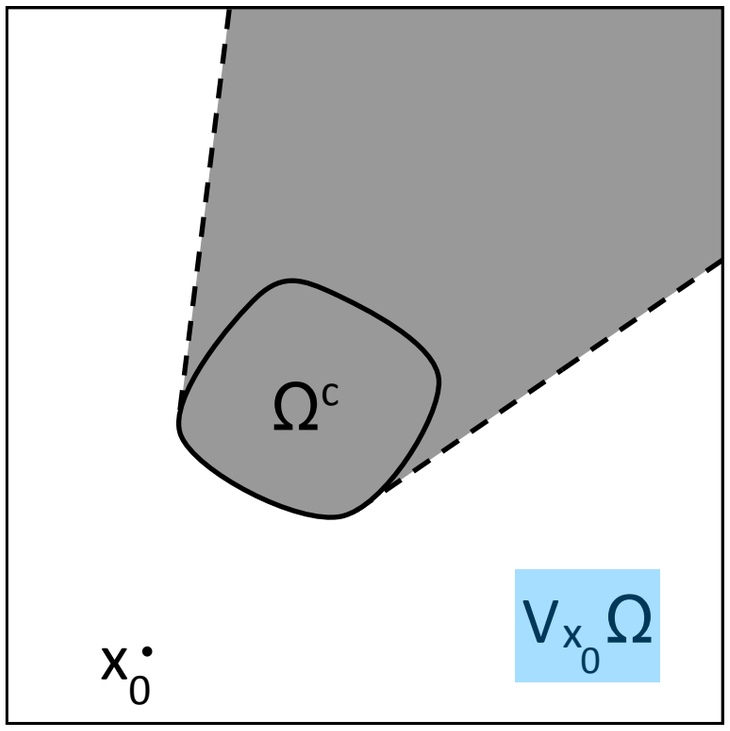
Exploration and surveillance

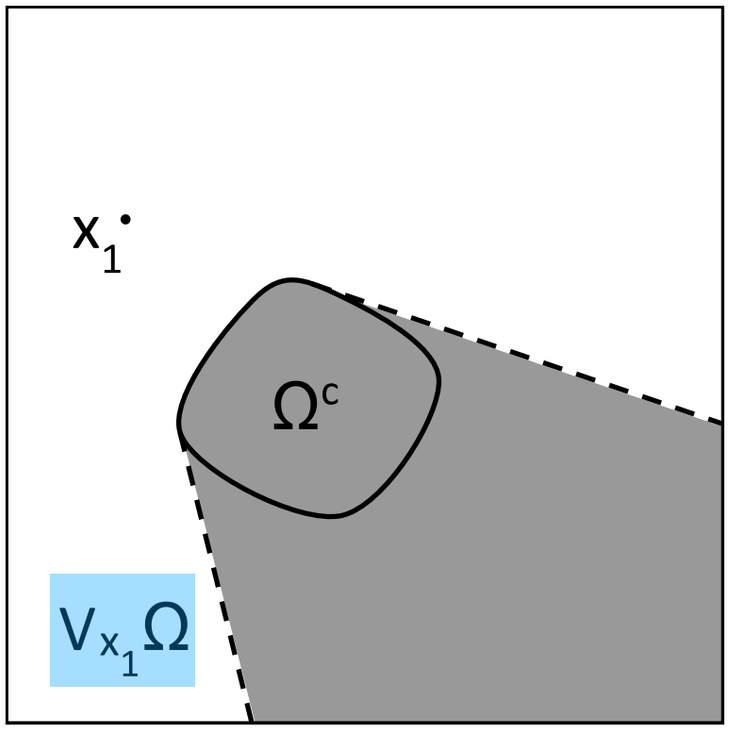
Problem formulation

Notation	Description
$D \subset \mathbb{R}^d$	Domain
$\Omega \subset D$	Free space
$x_i \in D$	Vantage points
$\mathcal{V}_{x_i}\Omega$	Visibility set from x_i

$$\Omega_k = \bigcup_{i=0}^{k} \mathcal{V}_{x_i}\Omega$$ Cumulatively visible set

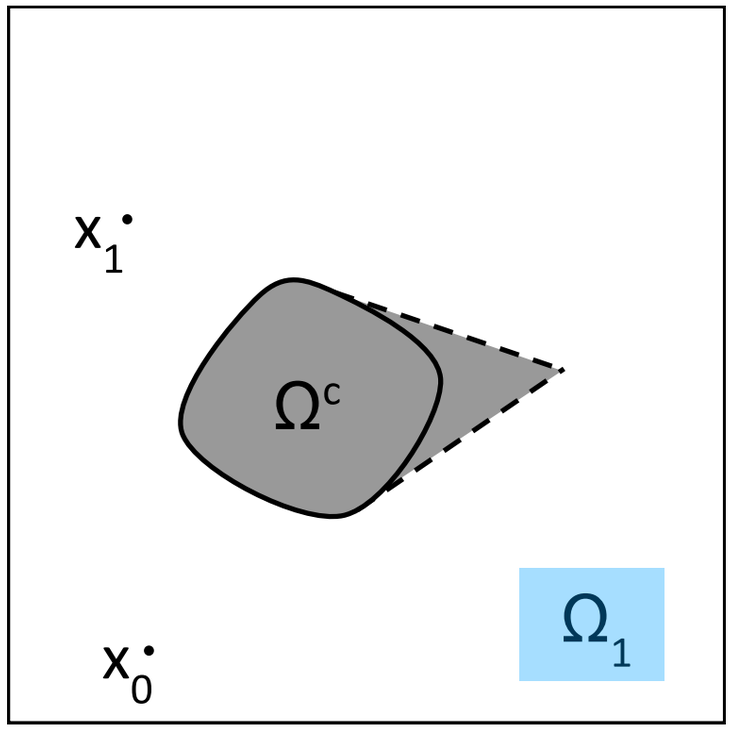






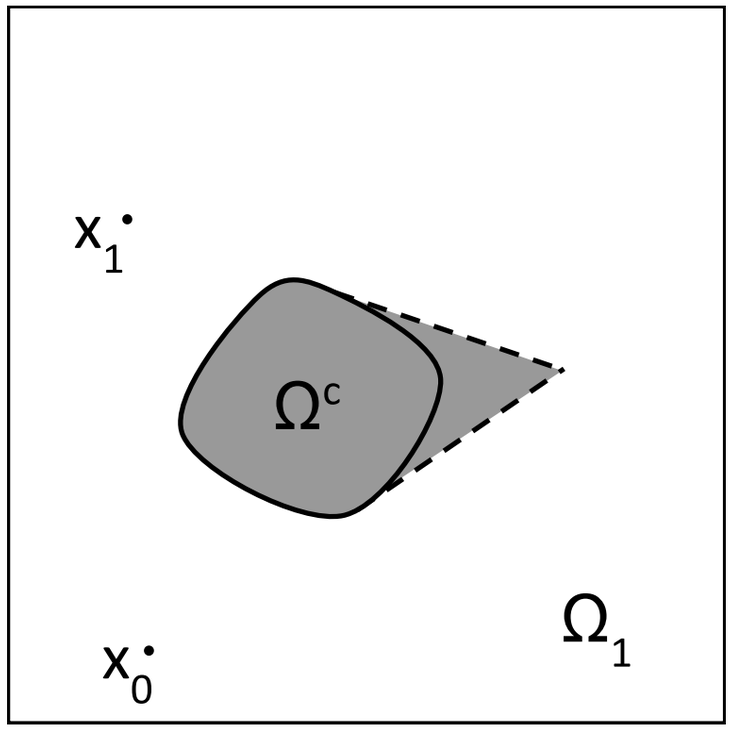
Problem formulation

Notation	Description
$D \subset \mathbb{R}^d$	Domain
$\Omega \subset D$	Free space
$x_i \in D$	Vantage points
$\mathcal{V}_{x_i}\Omega$	Visibility set from x_i
$\Omega_k = \bigcup_{i=0}^{k} \mathcal{V}_{x_i}\Omega$	Cumulatively visible set



“Sparse solution”

Notation	Description
$D \subset \mathbb{R}^d$	Domain
$\Omega \subset D$	Free space
$x_i \in D$	Vantage points
$\mathcal{V}_{x_i}\Omega$	Visibility set from x_i
$\Omega_k = \bigcup_{i=0}^{k} \mathcal{V}_{x_i}\Omega$	Cumulatively visible set
$I : \mathbb{R}^{m^d} \rightarrow \{0, 1\}$	Indicator function



$$\min_{I} ||I||_0 \quad \text{subject to} \quad \bigcup_{\{x_j | I(x_j)=1\}} \mathcal{V}_{x_j}\Omega = \Omega$$

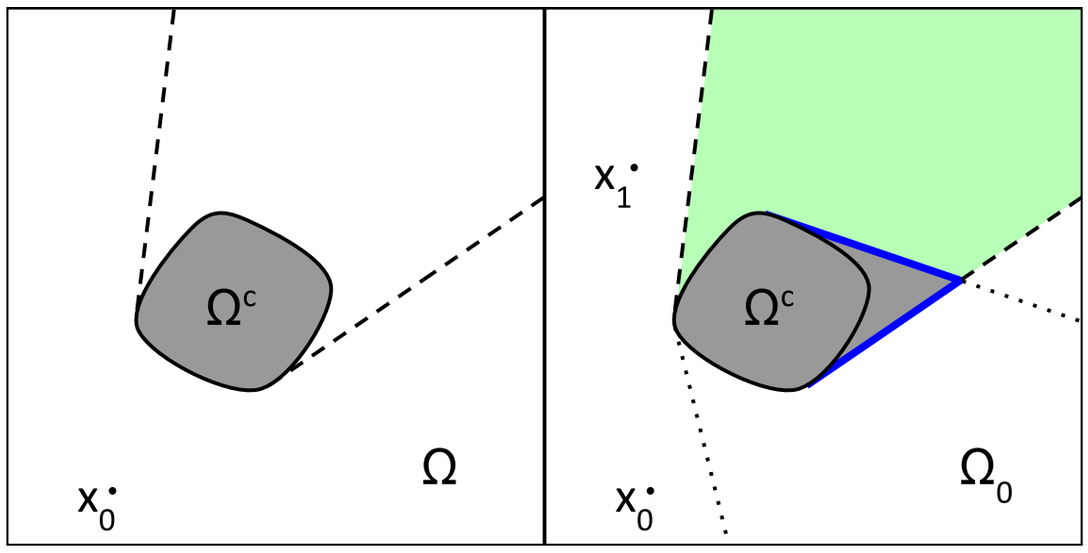
(formulated for surveillance of a known environment)

Greedy approach

$$\min_{I} \|I\|_0 \quad \text{subject to} \quad \bigcup_{\{x_i | I(x_i)=1\}} \mathcal{V}_{x_i}\Omega = \Omega$$

Assume environment Ω is known. Define gain function

$$g(x; \Omega_k, \Omega) := |\mathcal{V}_x\Omega \cup \Omega_k| - |\Omega_k|$$



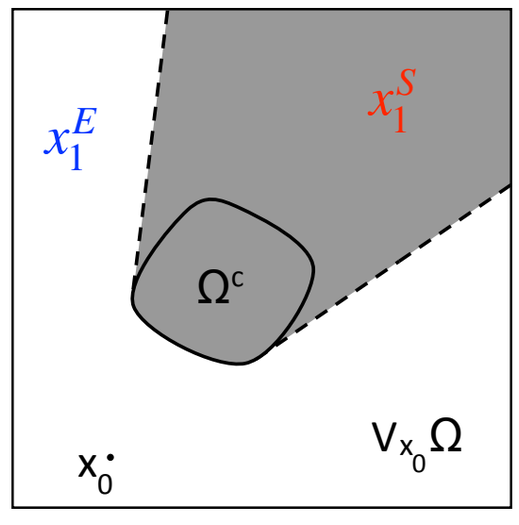
Algorithm:

$$x^{k+1} = \arg\max_{x} g(x; \Omega_k, \Omega) + \text{Reg}(x, \{x_j : j = 0,1,...,k\})$$

- Gain function g captures the volume of region (green) revealed by moving to x.

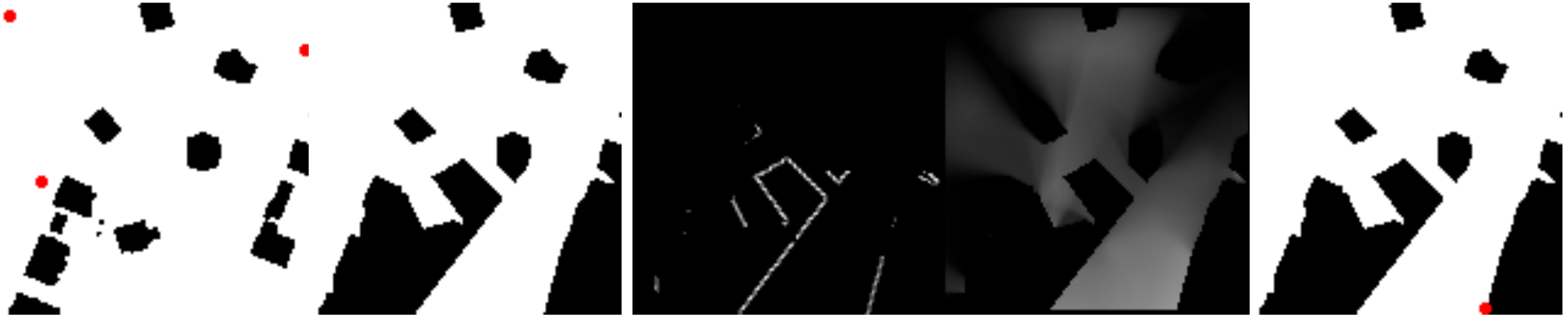
Our approach

- Greedy algorithm $$x^{k+1} = \arg\max_{x \in U_k} g(x; \Omega_k, \Omega) + \lambda R(x, x^k, x^{k-1})$$
- "Gain" (in new information) computed by a CNN $g \rightarrow g_\theta$
- The approach is not limited to depth information or the particular notion of "gain"
- Parts of the training data are computed
- Visibility level set method

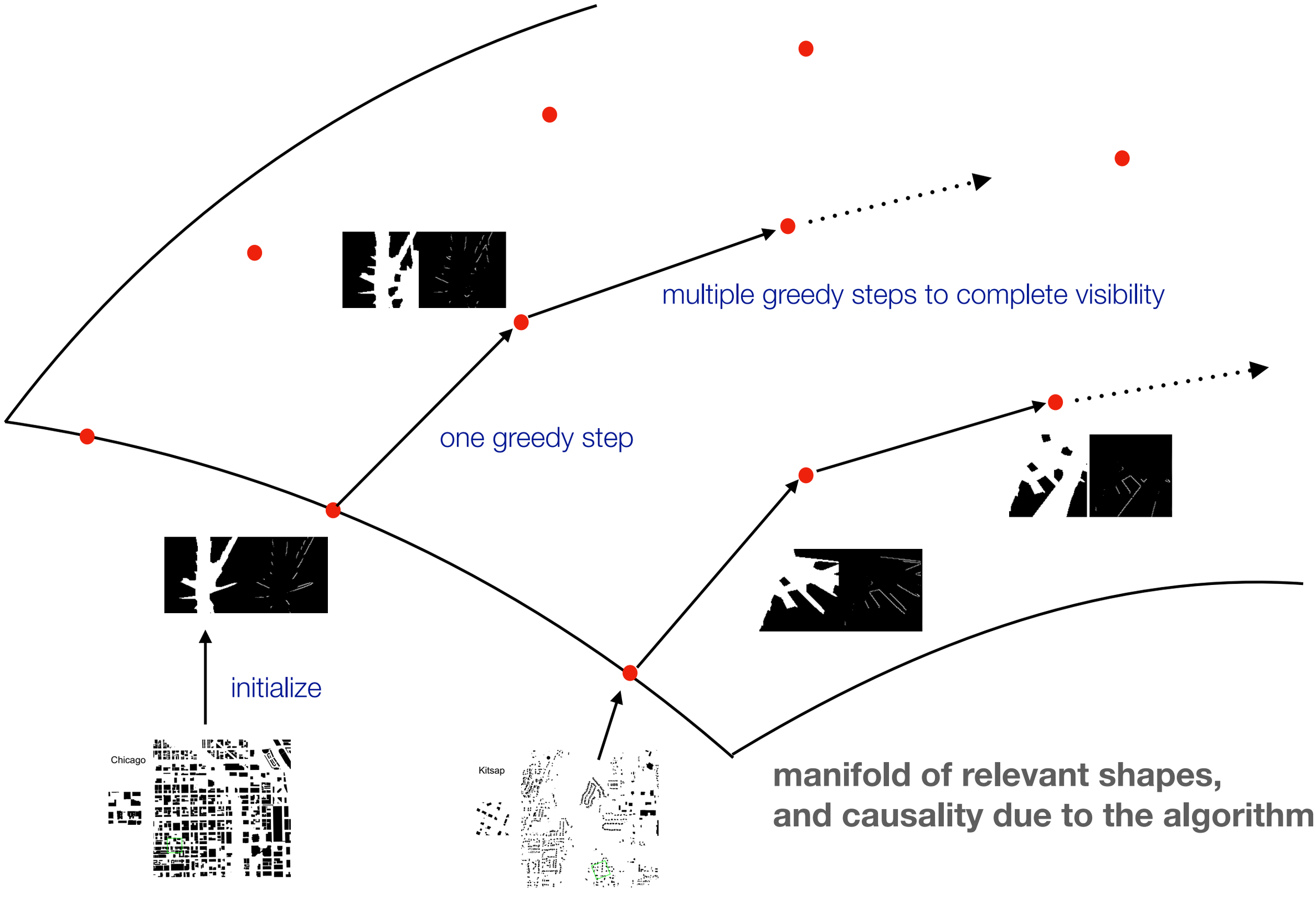


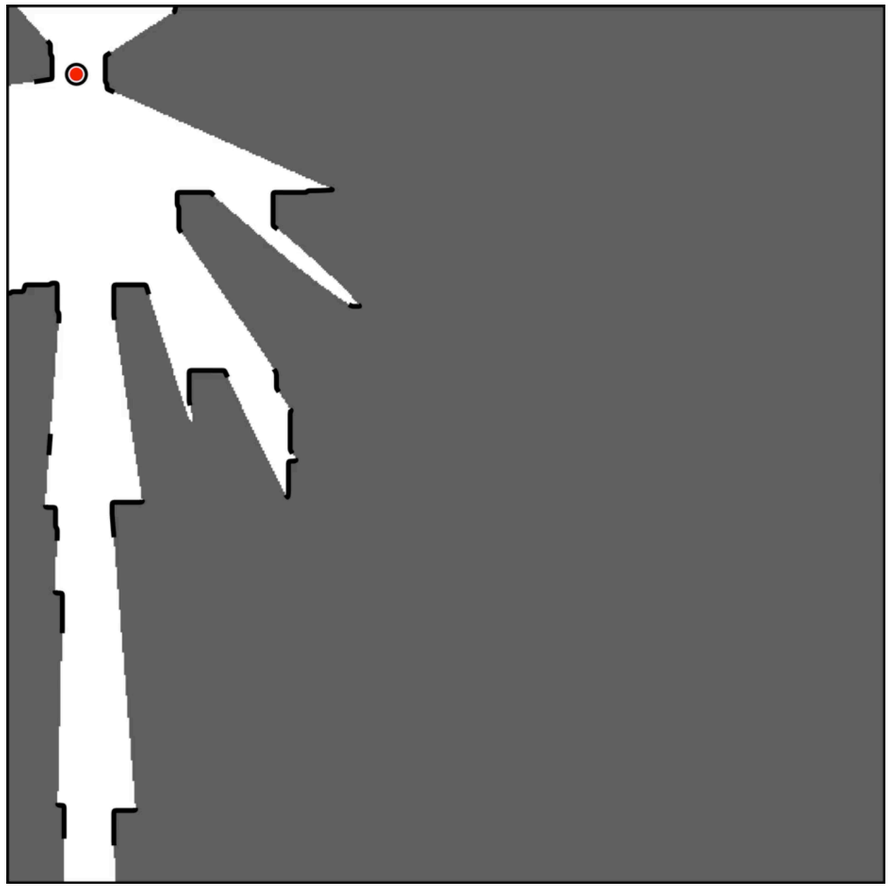
(S, ϕ) Ψ B $\mathcal{G}$ $\arg\max \mathcal{G}$

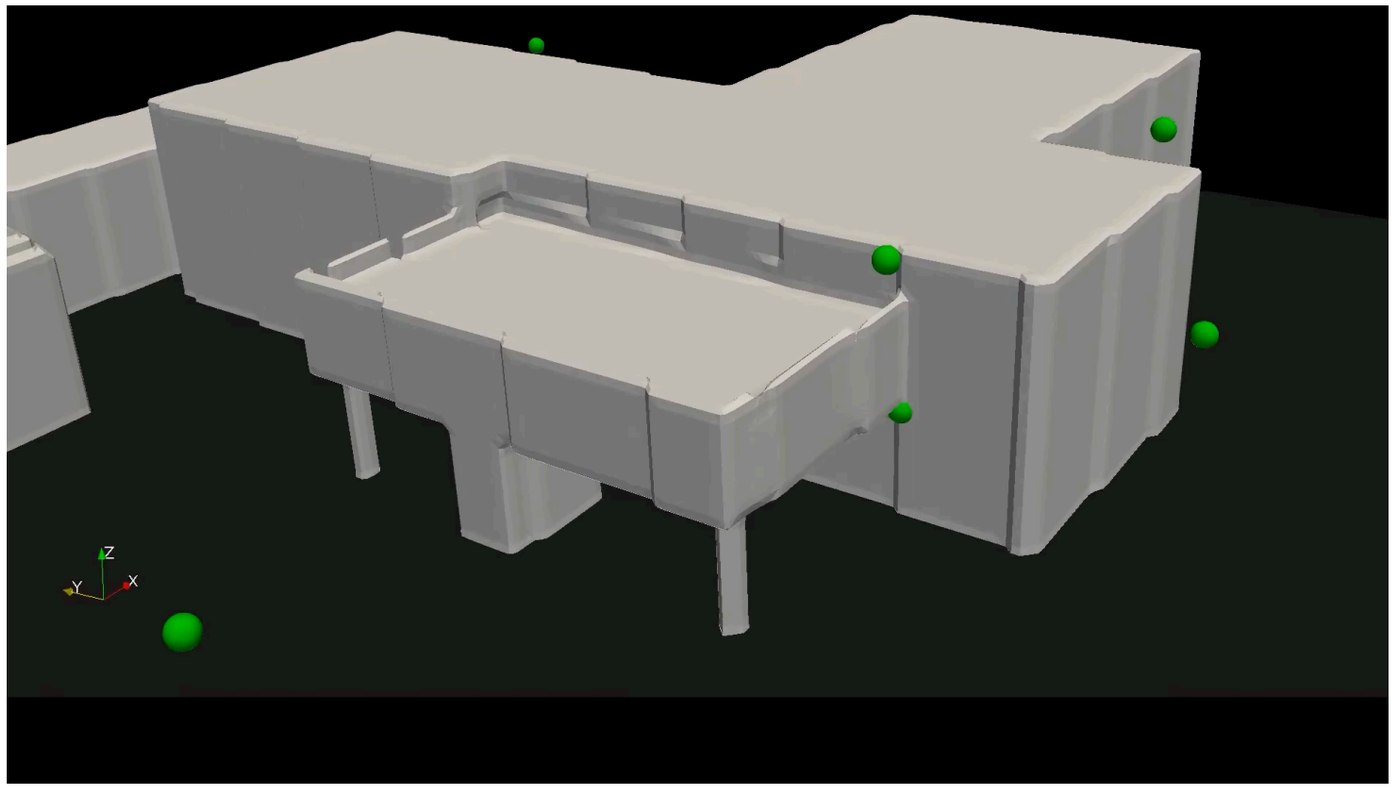
Cumulative Visibility Frontiers Gain Next Step



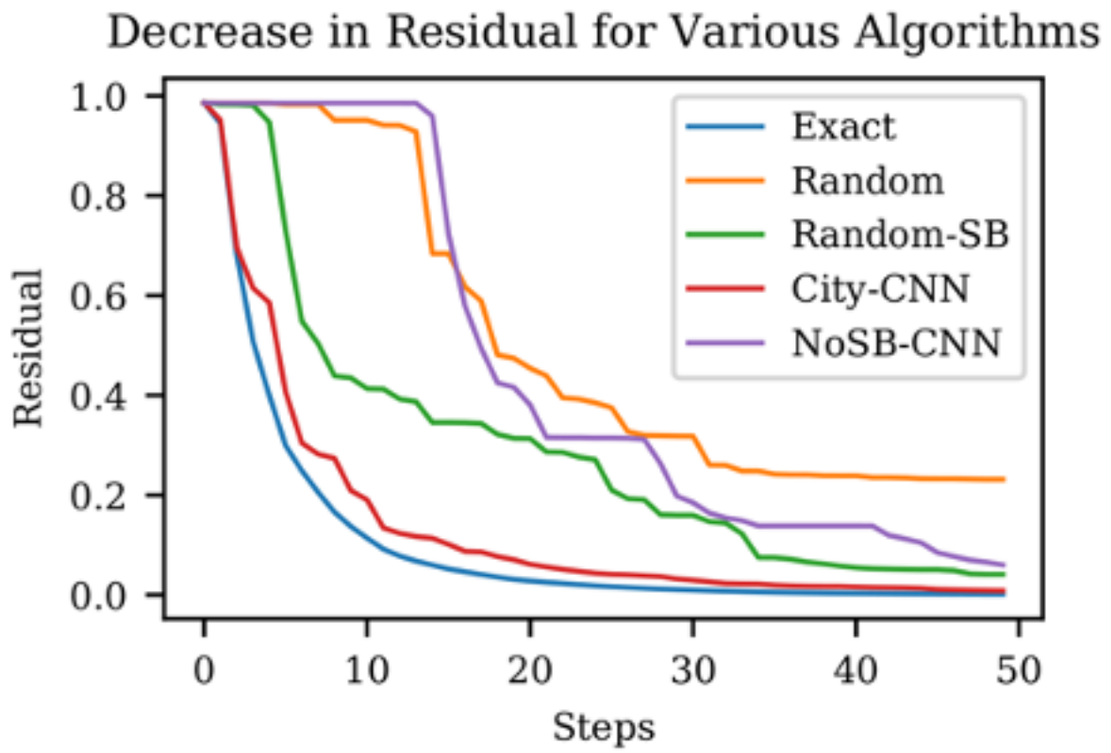
Sampling the shape space







Exploring Chicago


Decrease in Residual for Various Algorithms
Residual
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Steps
Exact
Random
Random-SB
City-CNN
NoSB-CNN

Some theory

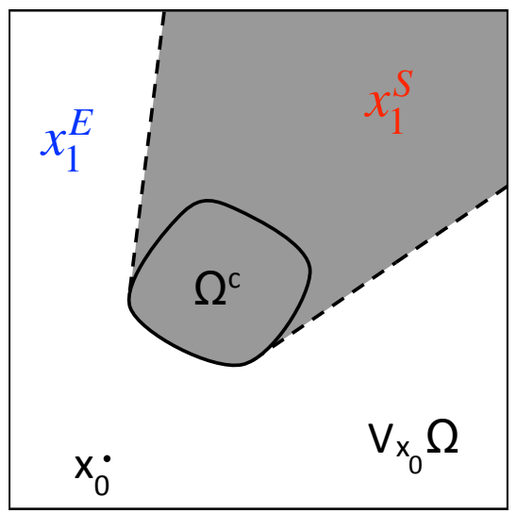
- Optimality of the greedy algorithm

- Fundamental differences between surveillance and exploration

  - Surveillance: next step can be anywhere in the free space

  - Exploration: next vantage point can only be in what has been explored by the previous steps.

- Observations



A bound for the known environment (surveillance)

Theorem 2.2.3. *Let O_k^* be the optimal set of k sensors. Let $O_n = \{x_i\}_{i=1}^n$ be the set of n sensors placed using the greedy surveillance algorithm (2.7). Then,*

$$f(O_n) \geq (1 - e^{-n/k}) f(O_k^*).$$

"If optimal solution requires k steps, then after k greedy steps, will have covered at least 63% of the free volume."

Optimality of Greedy algorithm [Krause-Colovin, 2014]

A bound for the unknown environment (exploration)

Depends on

- The arrangement of the obstacles
- The initial position

Exploration ratio

The ratio between a “greedy exploration step” and a “greedy surveillance step”.

$$\rho(A) := \frac{\sup_{x \in \mathcal{V}(A)} \Delta_f(x|A)}{\sup_{x \in \Omega_{\text{free}}} \Delta_f(x|A)}$$

$$\rho_x := \inf_{A\,:\,x \in A} \rho(A)$$

A bound for the unknown environment (exploration)

Theorem 2.2.4. *Let $O_k^* = \{x_i^*\}_{i=1}^k$ be the optimal sequence of k sensors which includes $x_1^* = x_1$. Let $O_n = \{x_i\}_{i=1}^n$ be the sequence of n sensors placed using the greedy exploration algorithm (2.8). Then, for $k, n > 1$:*

$$f(O_n) \geq \left[1 - \exp\left(\frac{-(n-1)\rho_{x_1}}{k-1}\right)\right]\left(1 - \frac{f(x_1)}{f(O_k^*)}\right] f(O_k^*).$$

optimal to greedy surveillance

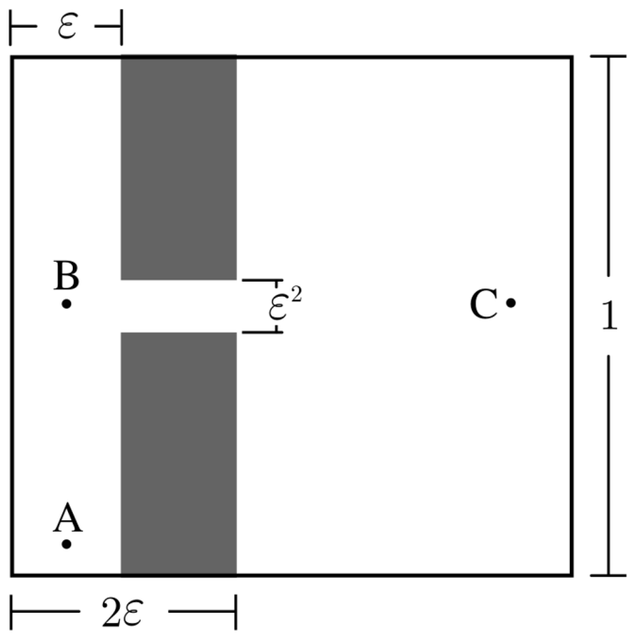
contribution from 1st vantage point

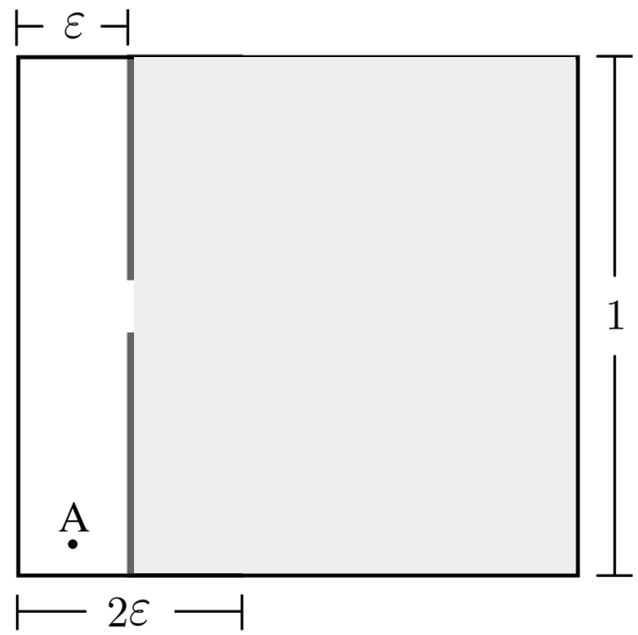
greedy surveillance to exploration

Narrow alley example

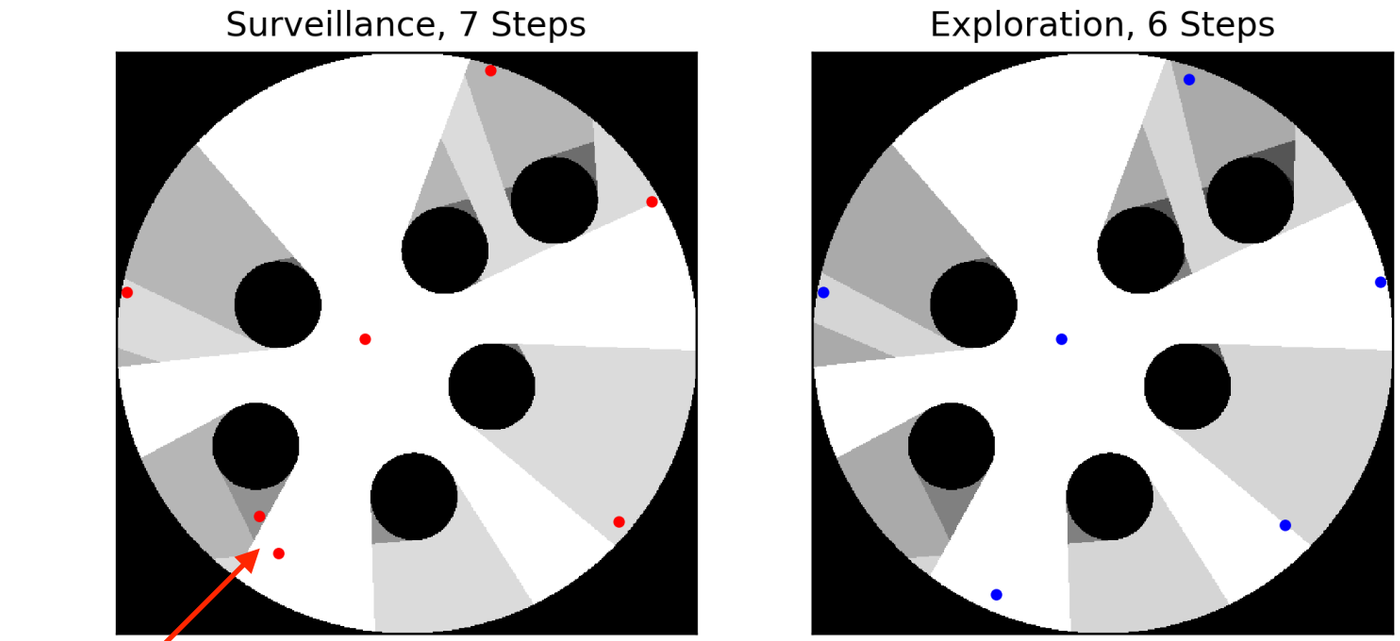
$$f(\{A, C\}) = \mathcal{O}(1)$$

$$\rho_A = \mathcal{O}(\varepsilon)$$



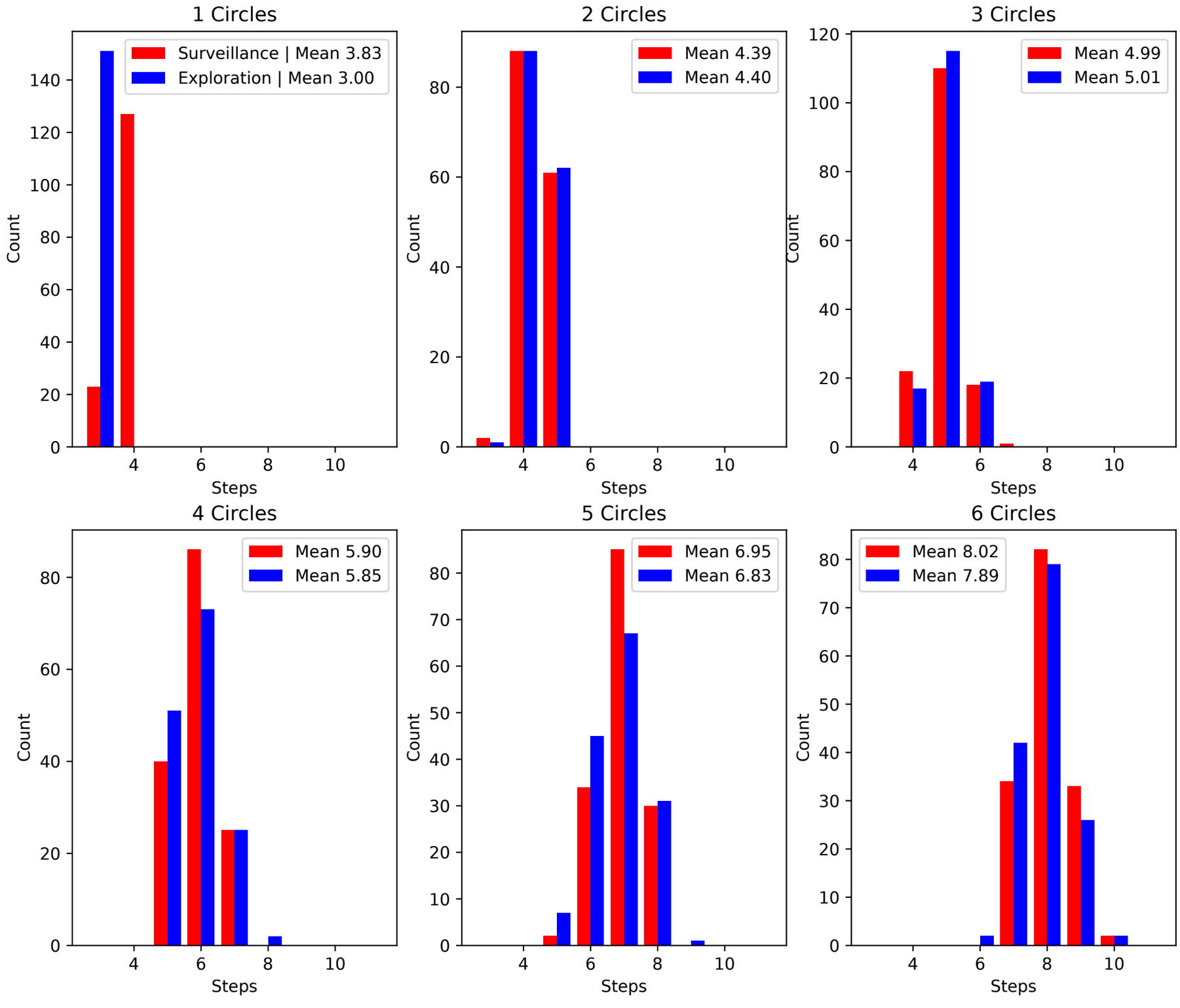


Some academic examples



- Both start from the same initial location.
- Similar observing locations.

Surveillance vs “exploration”

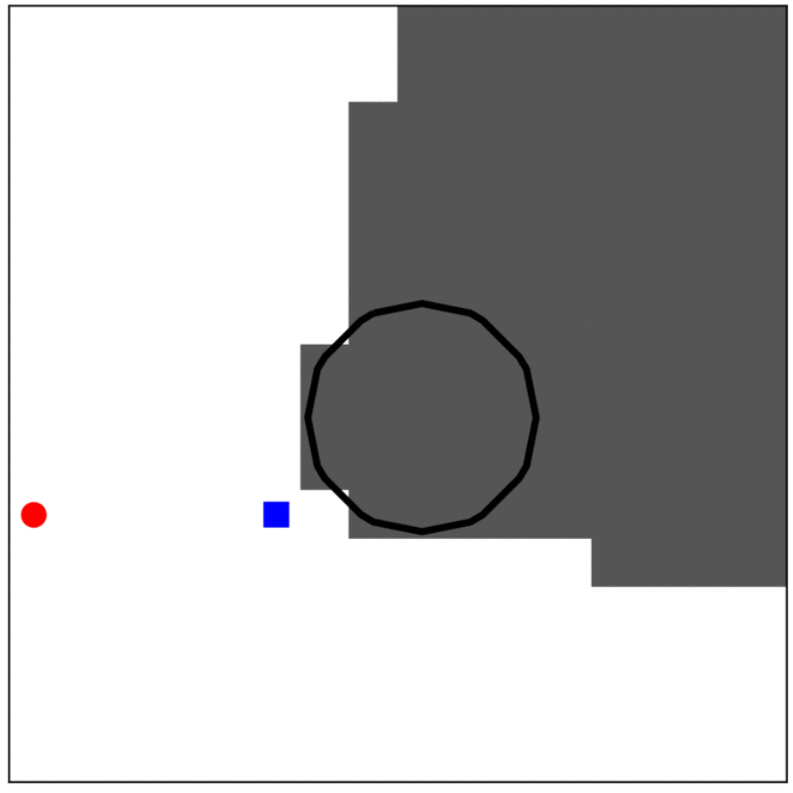


Optimal feedbacks for tracking moving agents

Surveillance-evasion games

Pursuer must maintain the evader under line-of-sight.

- Two players and non-cooperative
- Strict line-of-sight end-game condition
- Games allowing temporary occlusions
  - Total occlusion time budget
  - Maximum duration of temporary occlusion



- Multiple players
- This problem is very global due to the line-of-sights' global reach
- This talk will focus on the pursuer

Controlled dynamics for two players

$$\frac{d}{dt}P = f(P, \sigma_P(t)), \quad t > 0, \quad \sigma_P(t) \in A$$

$$P(0) = x \quad \Longrightarrow \quad P_x(t; \sigma_P)$$

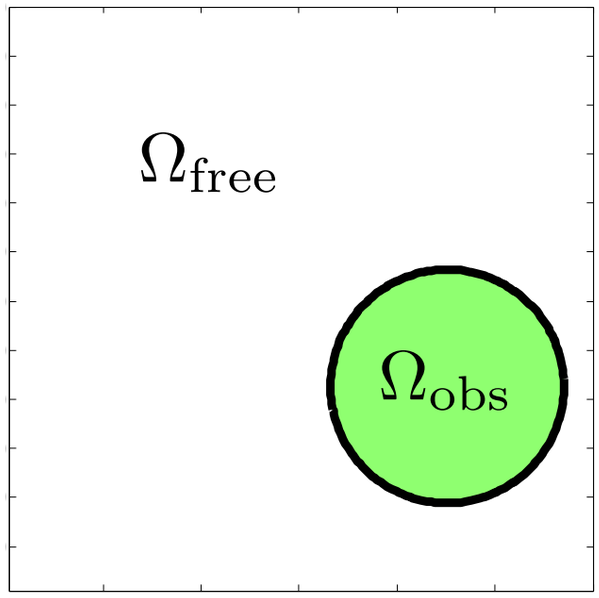
$$\frac{d}{dt}E = g(E, \sigma_E(t)), \quad t > 0, \quad \sigma_E(t) \in A$$

$$E(0) = y \quad \Longrightarrow \quad E_y(t; \sigma_E)$$

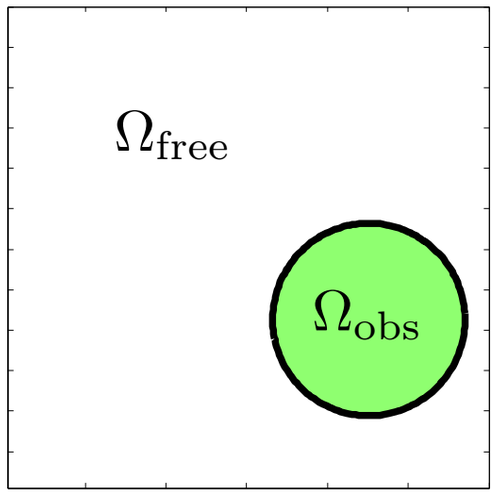
Feedback strategy ~ “policy”

$$\pi_P : (P, E) \mapsto A \qquad \frac{d}{dt}P = f(P, \pi_P(P, E)) \implies P_x(t; \pi_P)$$

$$\pi_E : (P, E) \mapsto A \qquad \frac{d}{dt}E = g(E, \pi_E(P, E)) \implies E_y(t; \pi_E)$$



Visibility based games

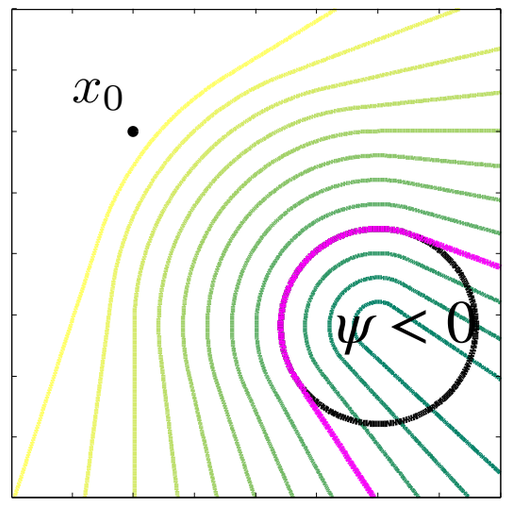


End-game states: $\mathcal{T}_{\text{end}} := \{(x, y) : \psi(x, y) \leq 0\}$

Feedback strategy game:

$$J[x, y, \pi_P, \sigma_E] := \inf\{t \geq 0 : (P_x(t; \pi_P), E_y(t; \sigma_E)) \in \mathcal{T}_{\text{end}}\}$$

$$u(x, y) := \sup_{\pi_P} \inf_{\sigma_E} J[x, y, \pi_P, \sigma_E]$$



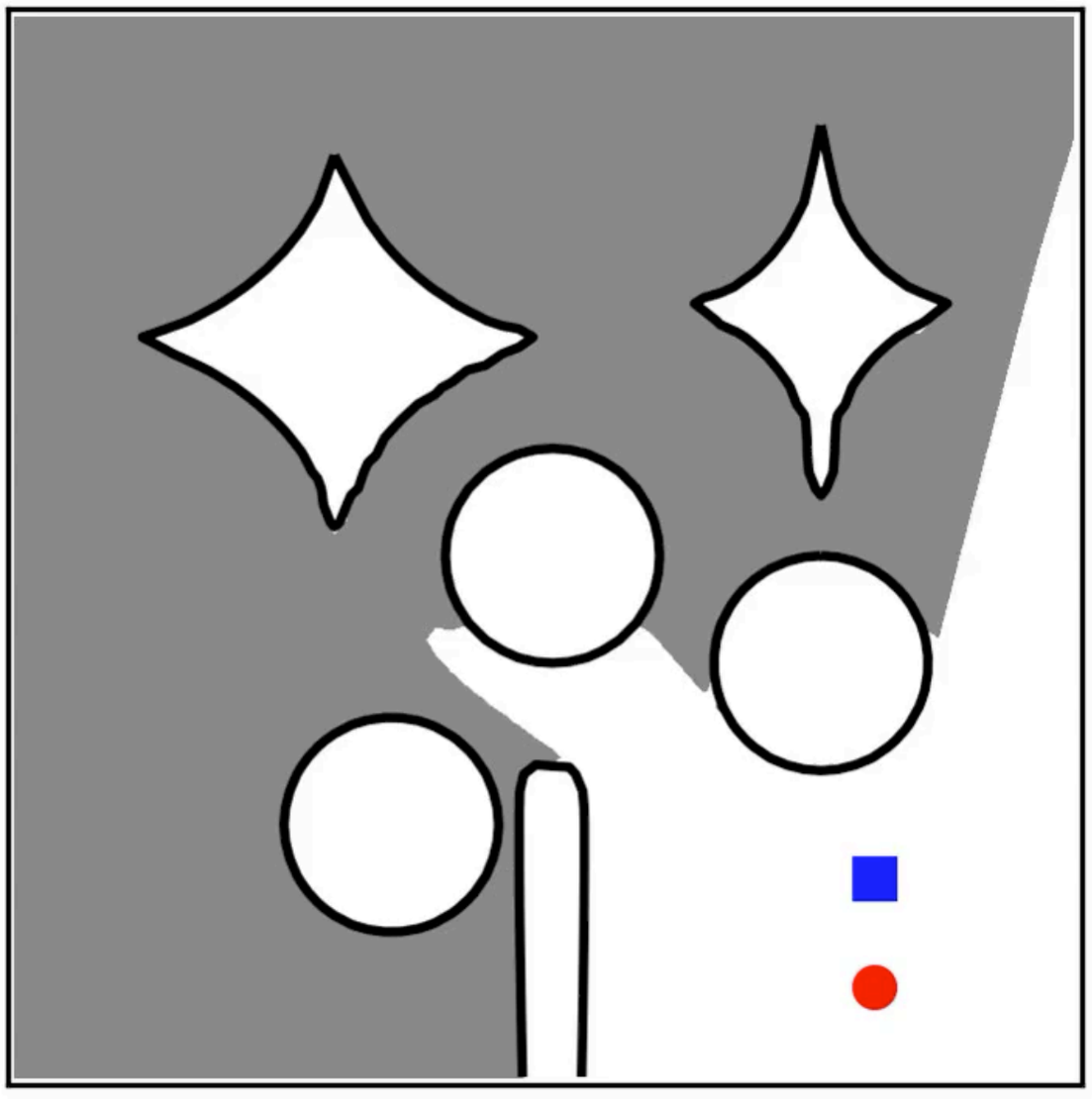
Static games: [Takei-T-Zhou-Landa 2014]

$$J[x, y, \sigma_P, \sigma_E] := \inf\{t \geq 0 : (P_x(t), E_y(t)) \in \mathcal{T}_{\text{end}}\}$$

$$V_s(x, y) := \sup_{\sigma_P} \inf_{\sigma_E} J[x, y, \sigma_P, \sigma_E] \qquad \tilde{V}_s(x, y) := \inf_{\sigma_E} \sup_{\sigma_P} J[x, y, \sigma_P, \sigma_E]$$

$$V_s(x, y) \leq u(x, y) \leq \tilde{V}_s(x, y)$$

A simulation involving optimal feedback



Observe that the blue guy knows
when to **move forward** to guard special position,
when to **pause**, when to **follow closely**

Learning near optimal policies

- But what to train on?

- Use Monte-Carlo-Tree-Search (MCTS) to
  - sample admissible trajectories in training domains and assess values
  - refine policies
  - play game until win/lose
  - (p^*,v^*) = (refined policies from MCTS, game outcome)
- This generates training data for the neural network
- Use MTCS online to further refine the policy

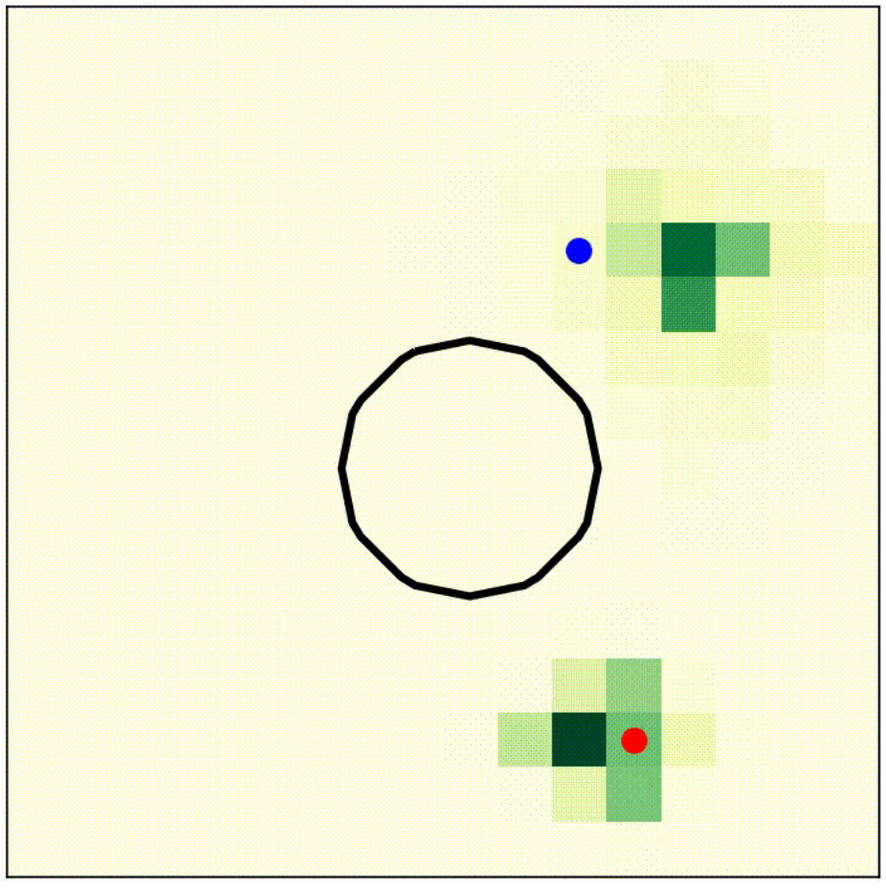
Search algorithms

- We consider AlphaGo style Monte-Carlo-Tree-Search algorithms
- Why?
    - High dimensional variable coefficient Hamilton-Jacobi-Isaacs equations are difficult to solve or impossible to formulate
    - Working policies for different geometries without “recomputing”
    - Potential to play the game online without a map of the environment (only what has been observed during the game play)
    - Highly relevant for developing multi-step greedy algorithms (reinforcement learning)

Search algorithms

- What have we discovered so far?
- Good heuristics accelerated the learning process
- Heuristics alone are not sufficient

MCTS



- MCTS is a function that outputs a refined policy p*, given
  - Current state: s
  - An evaluator function: f
  - The policy to be refined: p
  - # of iterations (higher means more lookahead steps)

- Evaluator function f(s), produces policy and value estimate
  - Policy is probability distribution over actions
  - Value is estimate of the likely winner

Training

- Neural network take in images representing the state:
    - Visibility, frontiers, positions, obstacles
- Network outputs policy and value estimate
- MCTS helps to sample relevant branches
- Train over various obstacle setups
- Goal: Play game on unseen maps

Improving the efficiency of self-play

- Initially, the evader is weak, uses random movements
- This means the pursuer always win, no need to learn anything
- Self-play works (for GO) but requires extensive training time
    - See Leela-Zero, a crowd-source attempt to reproduce Alpha Go Zero

Stronger adversaries provide more efficient training data

Use heuristics to help initialize the neural network, then fine tune
(this is against the Alpha Go Zero philosophy!)

Locally optimal evader

- When the pursuer is stationary, HJI becomes simple Eikonal

$$-c_P|\nabla_x u| + c_E|\nabla_y u| = 1, \quad (x,y) \in \Omega_{free} \backslash \mathcal{T}_{end}$$

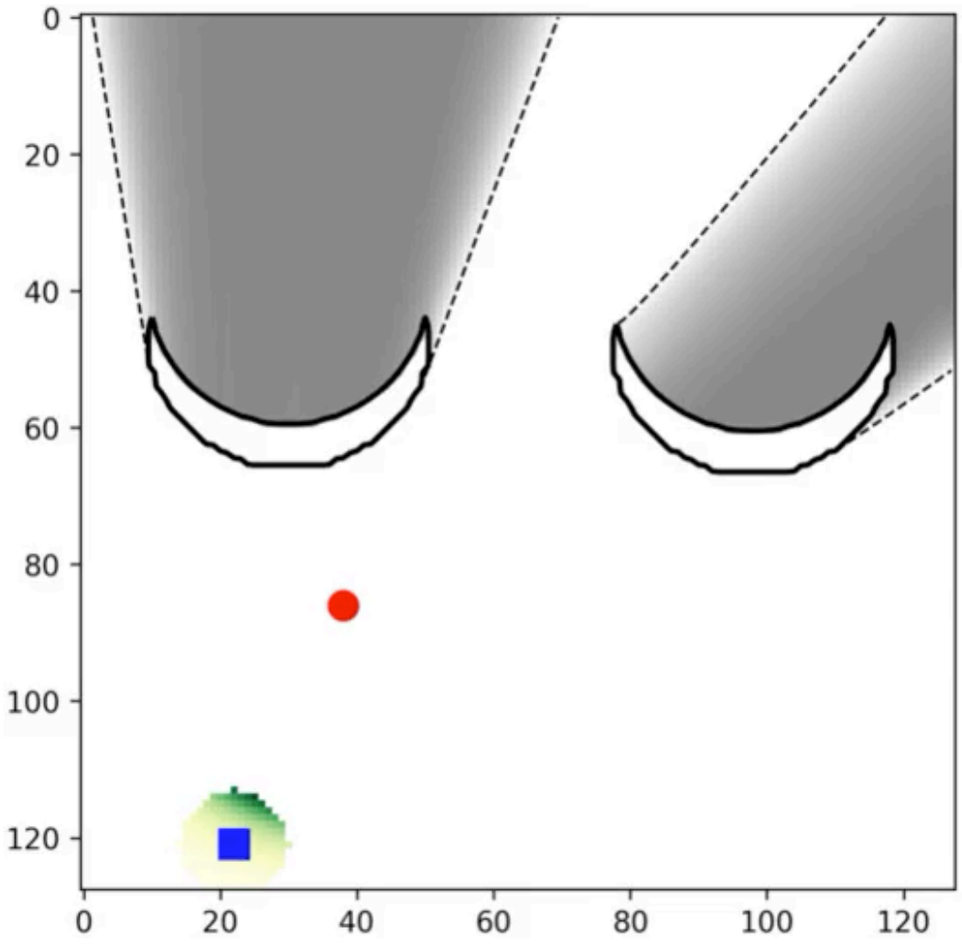
$$u(x,y) = 0 \quad (x,y) \in \mathcal{T}_{end}$$

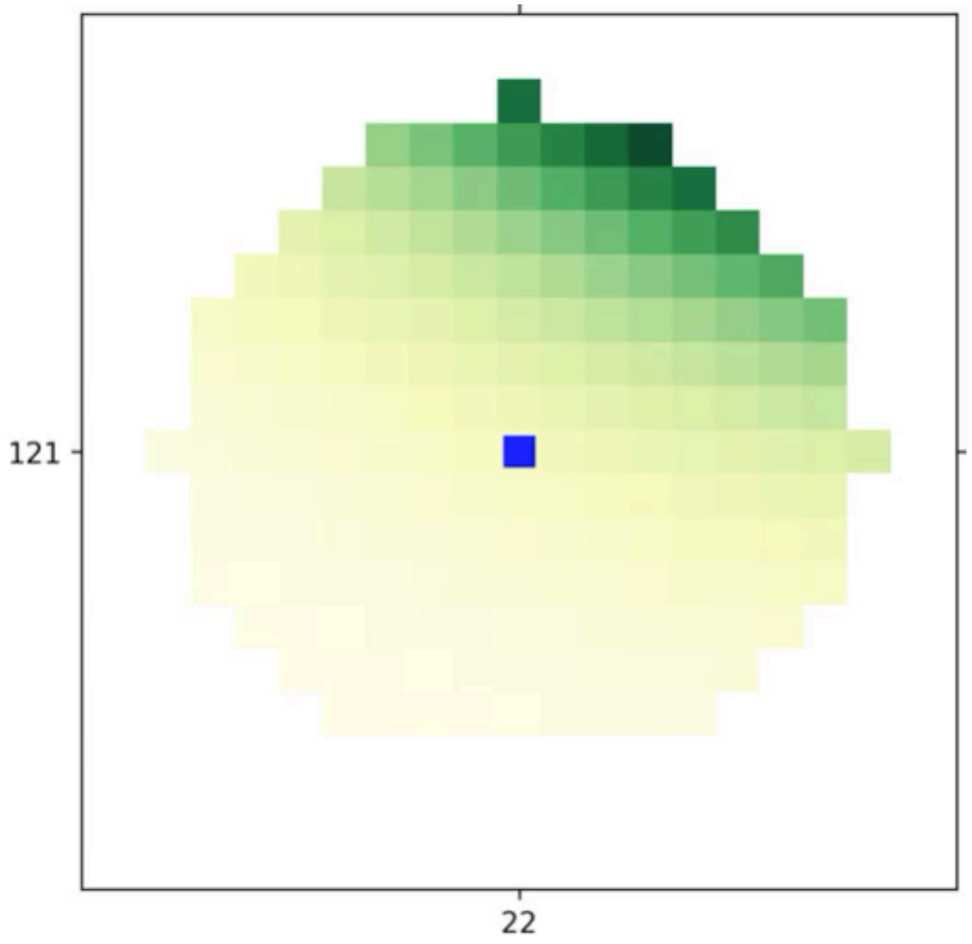
- u becomes the distance function to the end-game state.
- Evader should move towards the closest shadow boundary.

The get-closer (distance) strategy for P

- Minimize distance to E
- Keeps E from getting too far
- But being too close to obstacles means larger occlusions
- Cat-mouse game can fail at corners

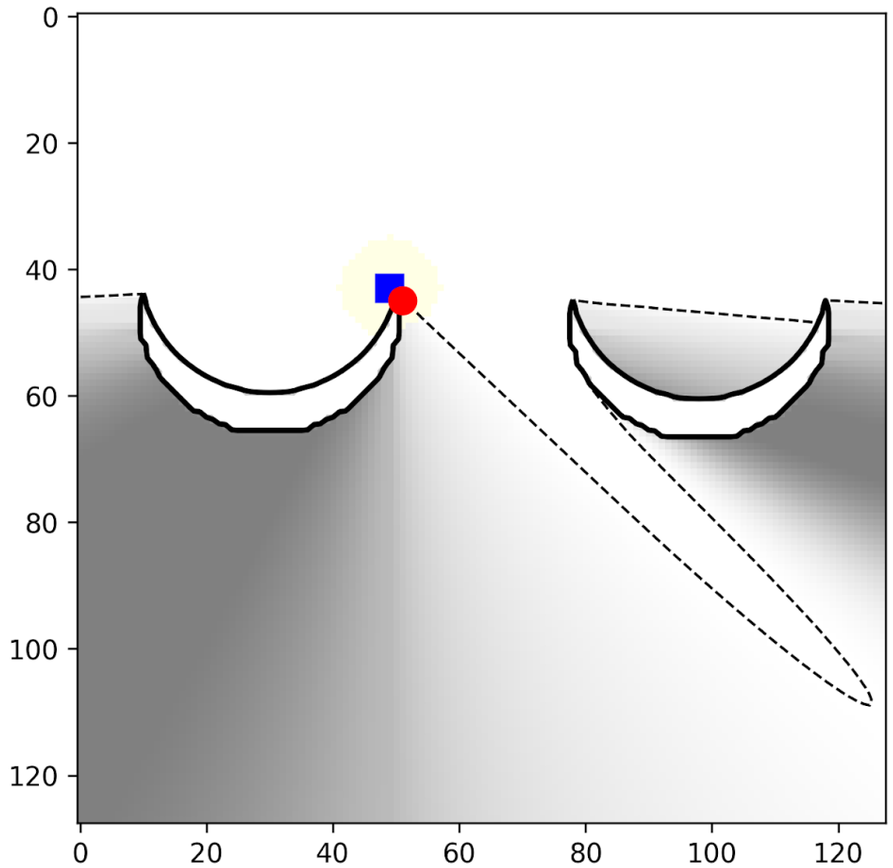
The get-closer strategy for P

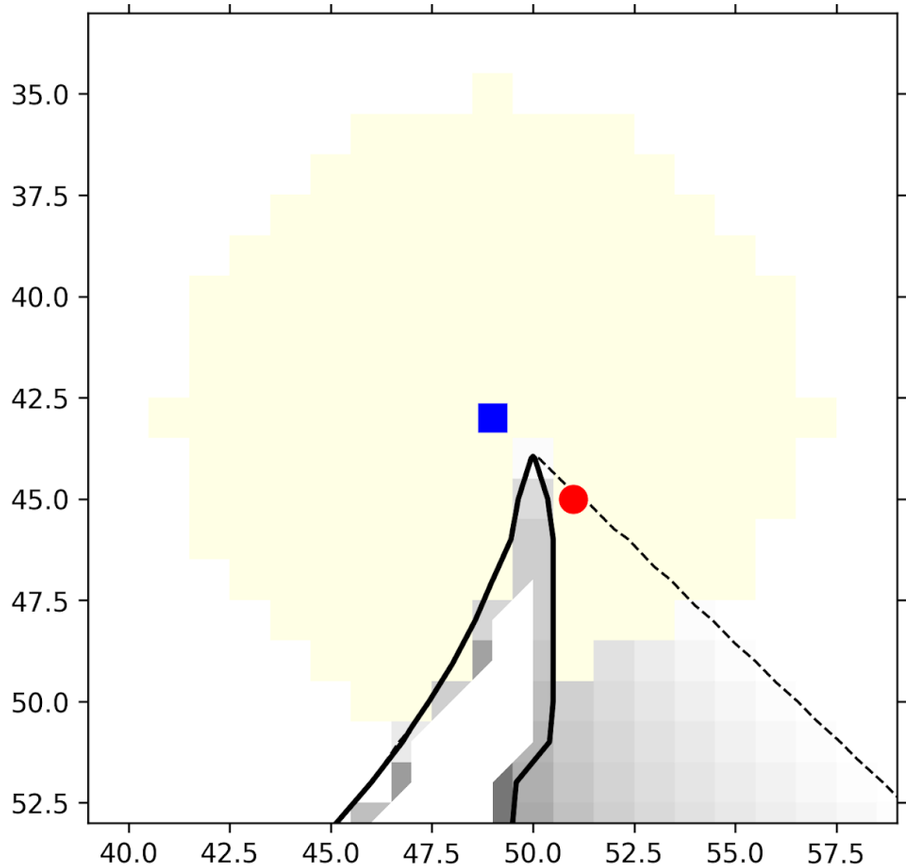




Softmax of (negative) distance is shown, green means higher probability.

The get-closer strategy for P





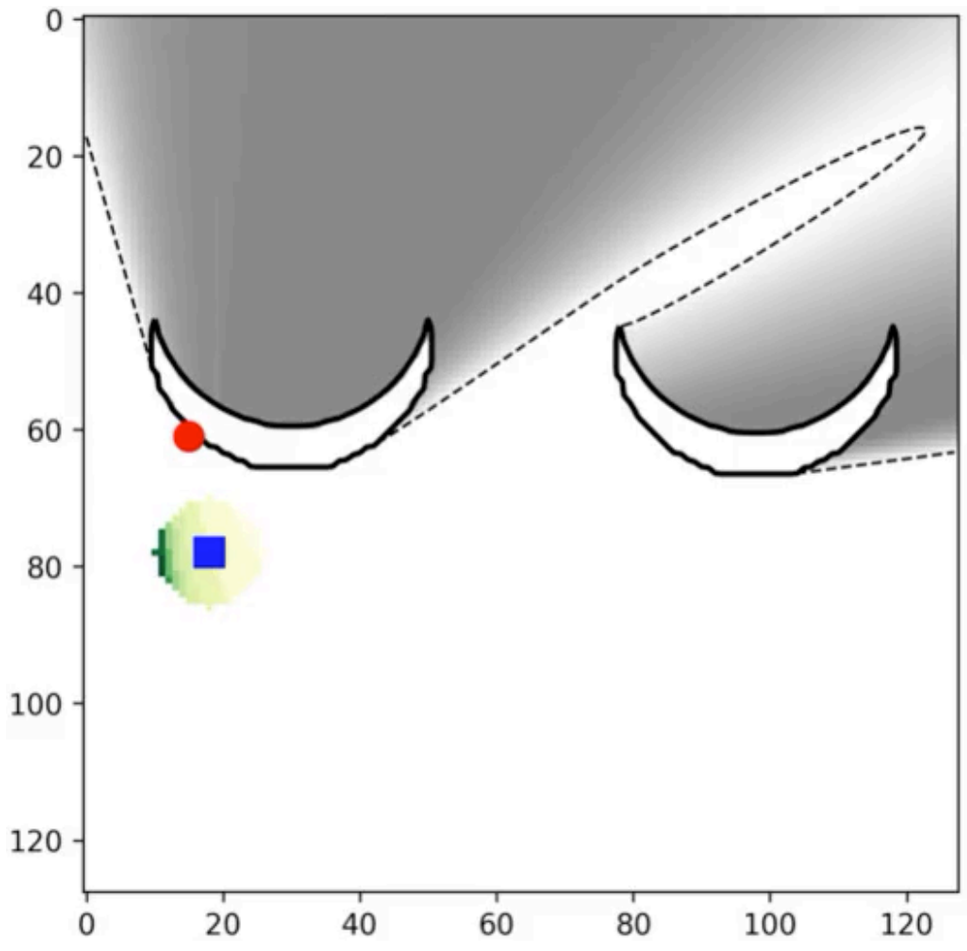
This policy leads to a cat and mouse chase where P moves to E's previous location.

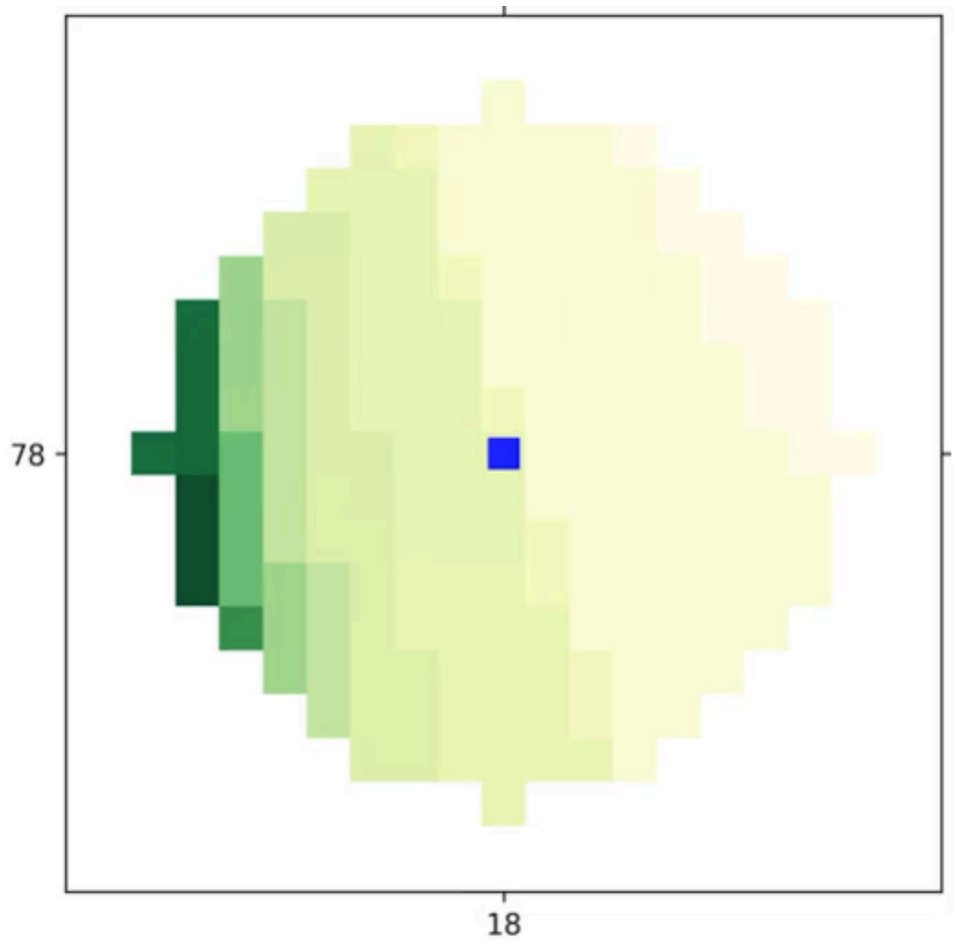
Near obstacles, occlusions are larger, and E wins by jumping behind obstacle.

The steer-shadow-away (shadow) strategy

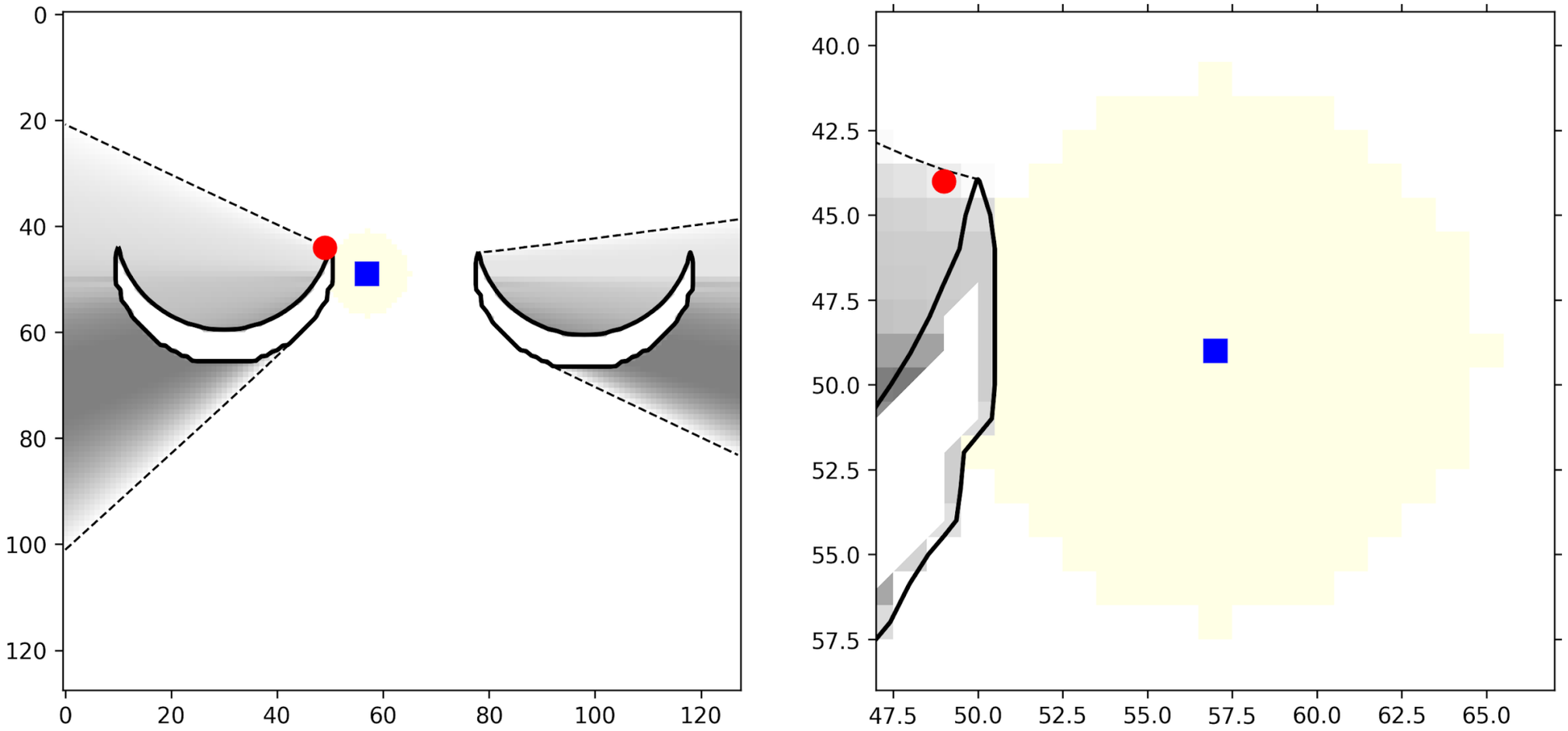
- Based on rationale for the static game
- Pick P's action that maximizes E's travel time to the corresponding shadow.
- By moving farther from obstacles, shadows become smaller.
- But moving too far can make it impossible to catch up to E

The steer-shadow-away strategy for P



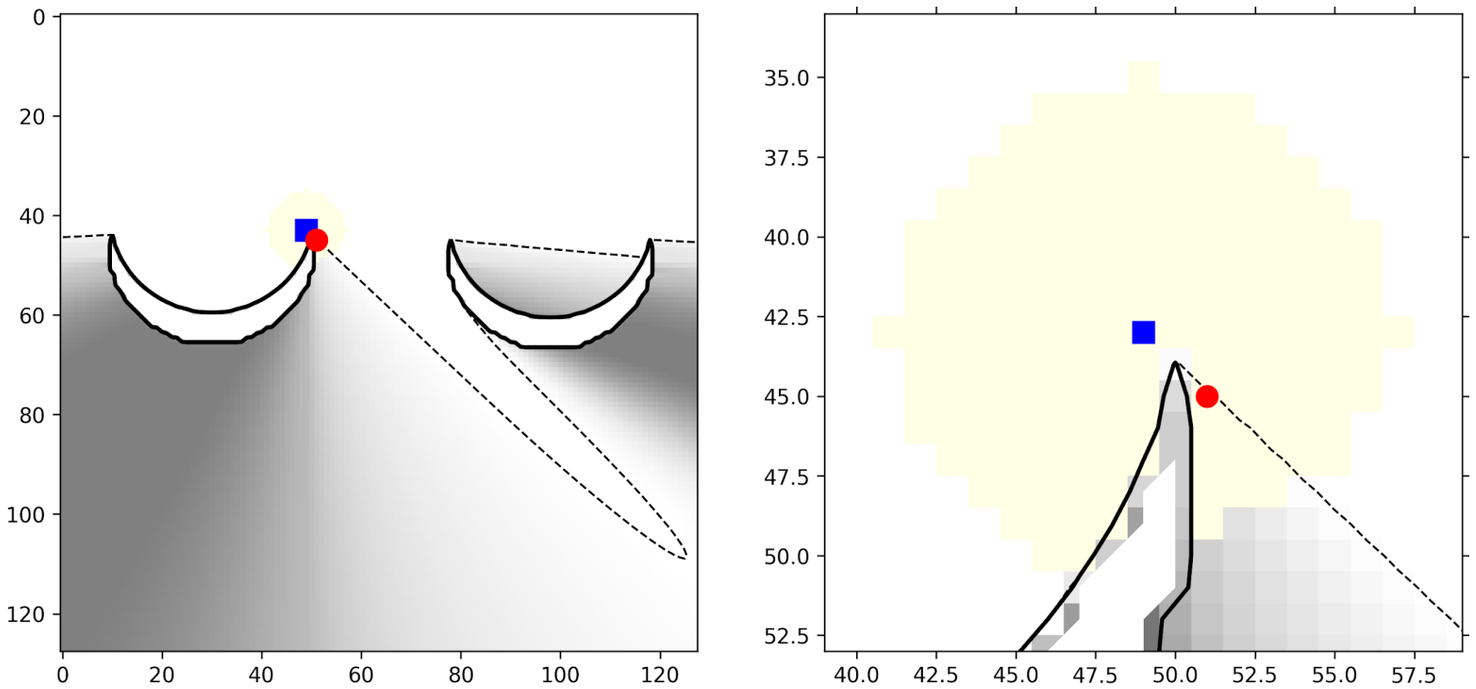


The steer-shadow-away strategy for P

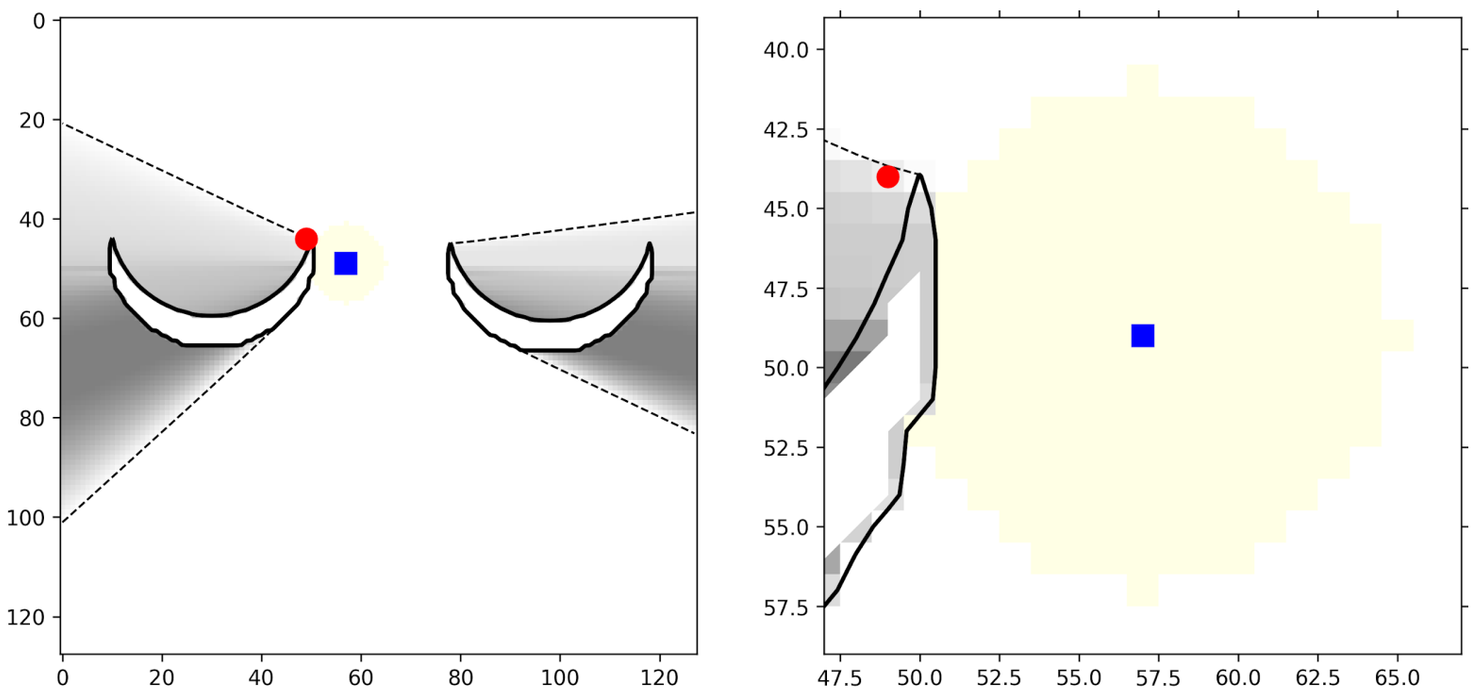


We avoid the cat-and-mouse game, because P stays further back.

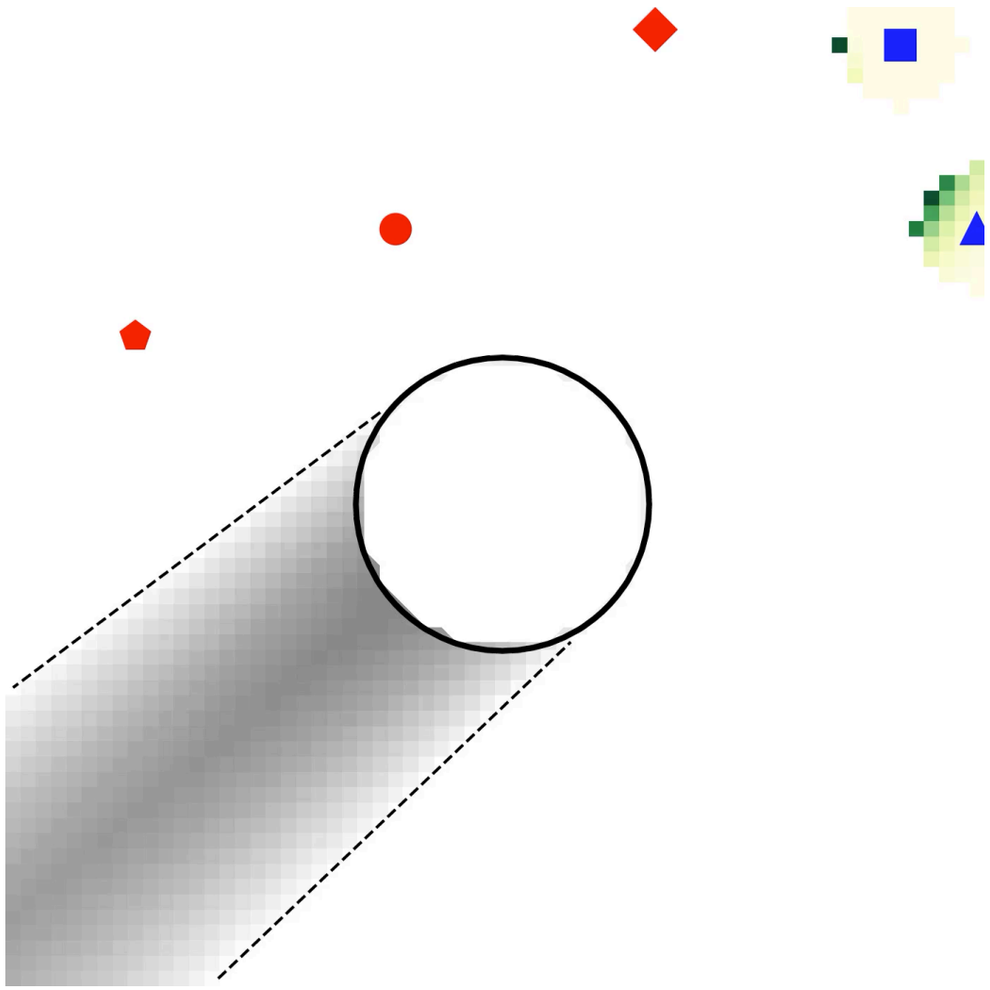
But in the open space, P might stray too far away and not be able to recover.



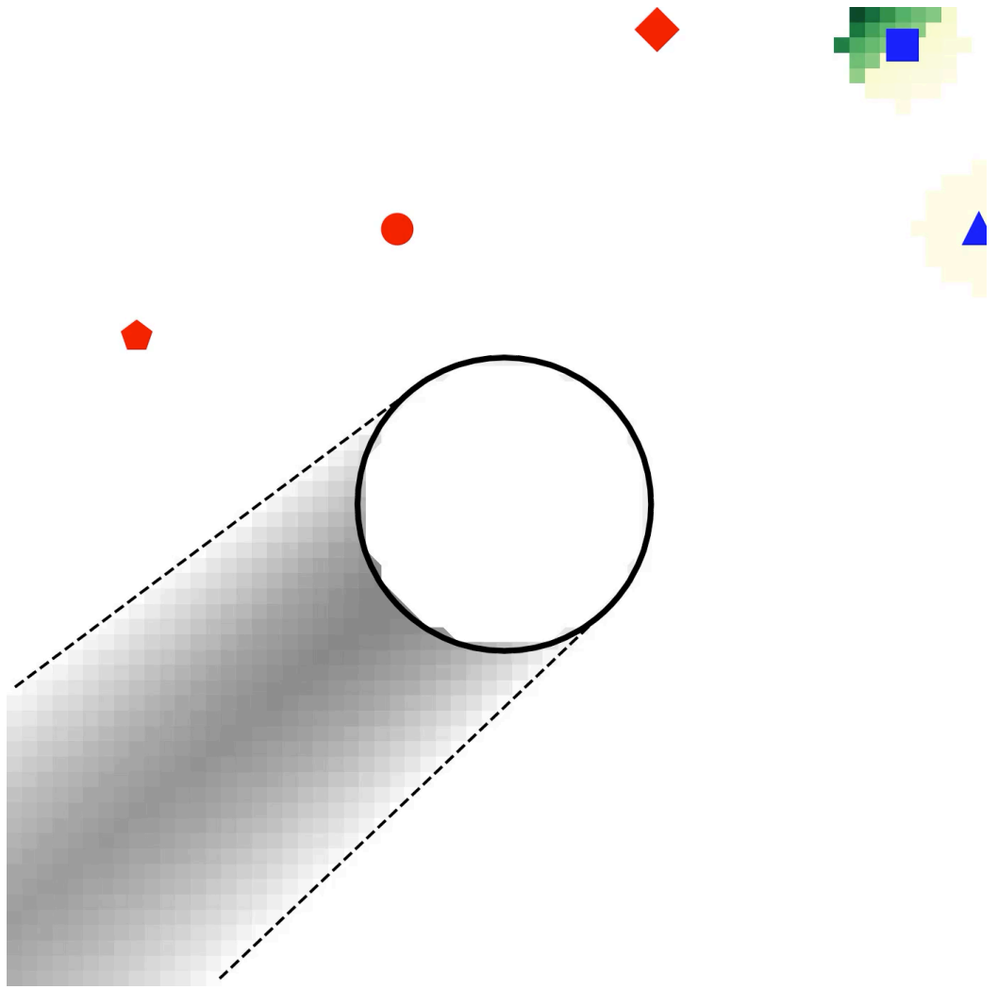
In both cases: pursuer does not fully utilize the dynamics of the frontiers



Dynamics from the two strategies

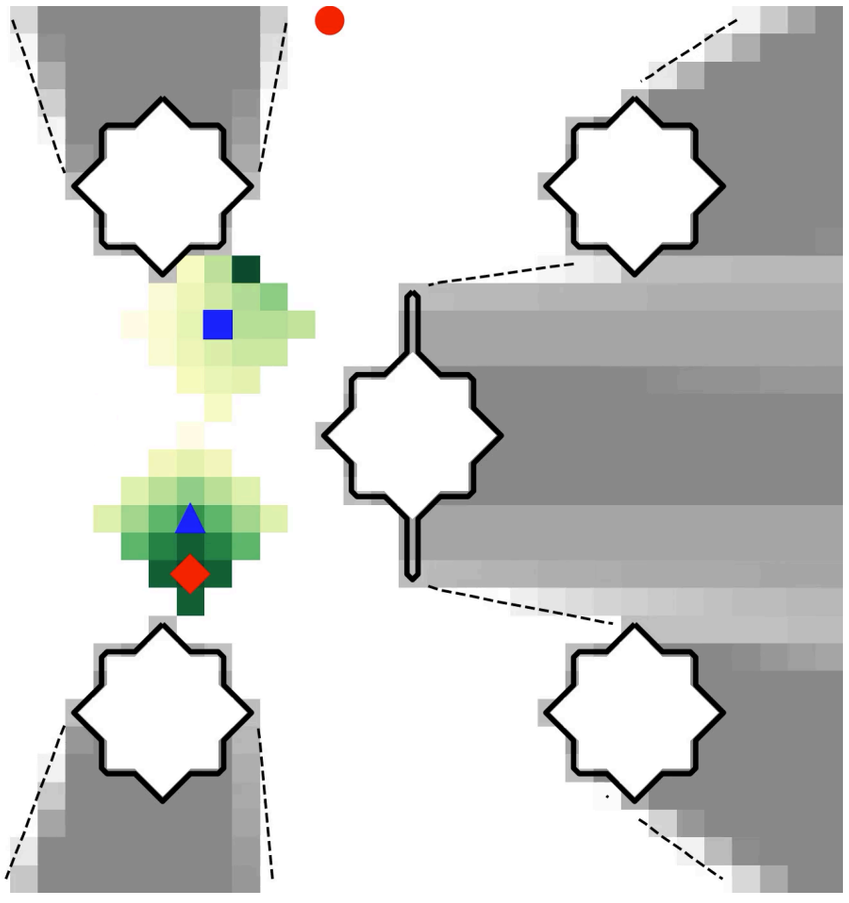


Pursuers lose using the get-closer policy

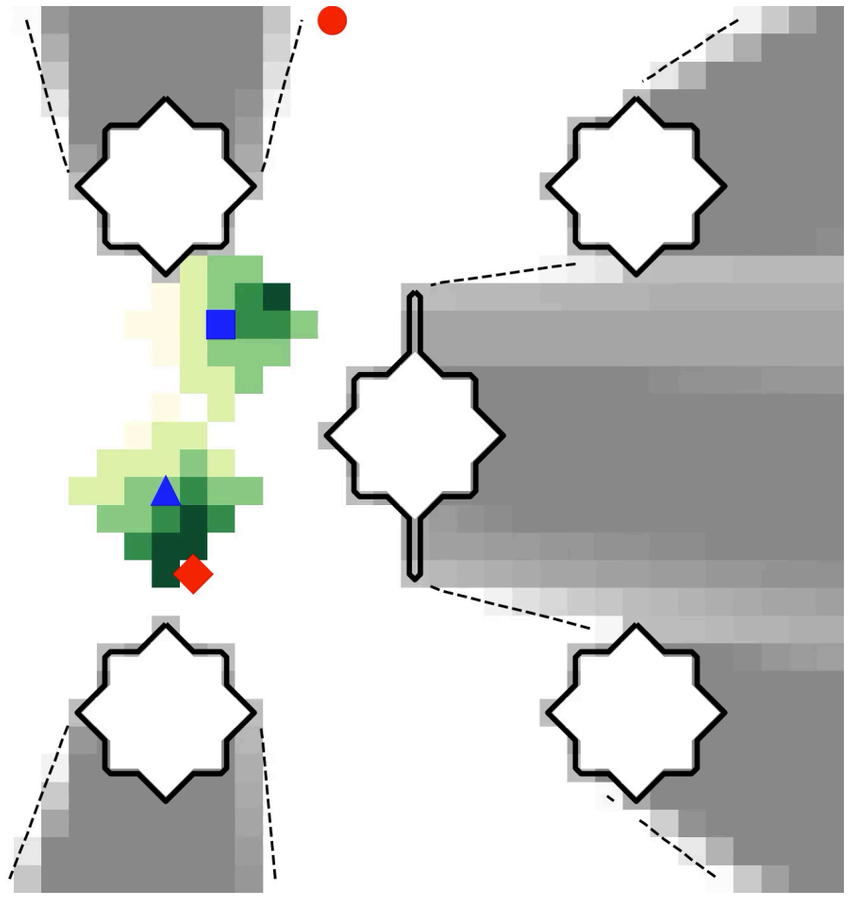


Pursuers win using the steer-shadow policy

MCTS-improved policy



MCTS(0, get-closer)



MCTS(500, get-closer)

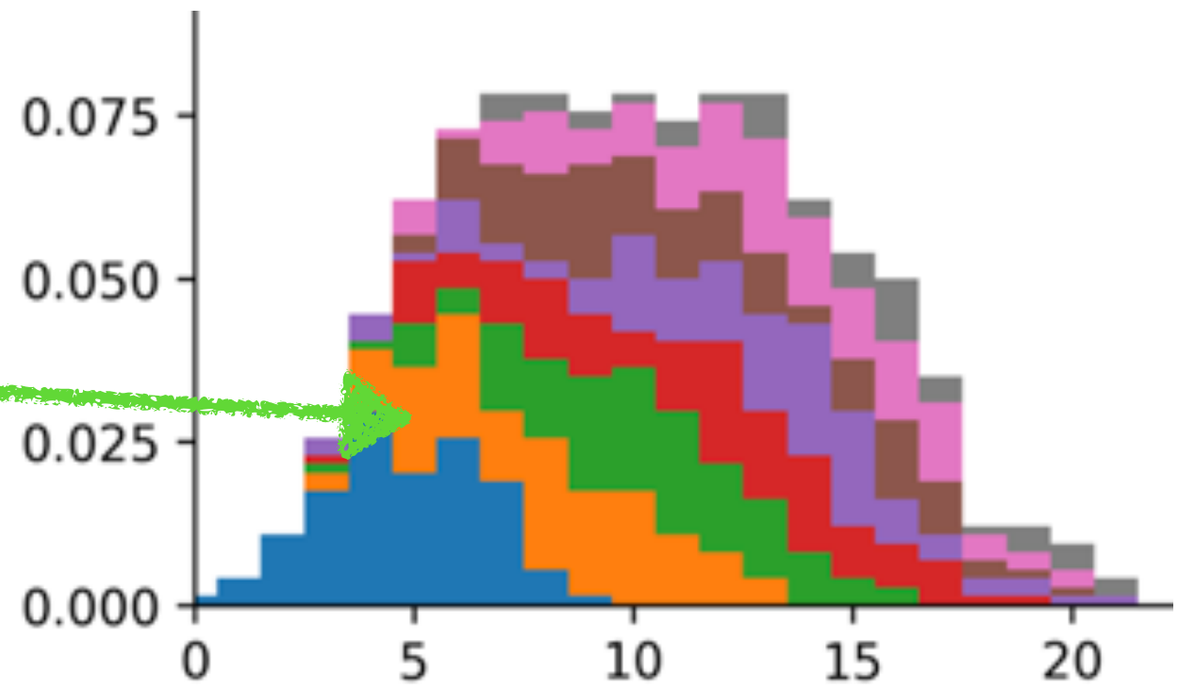
Online MCTS improves

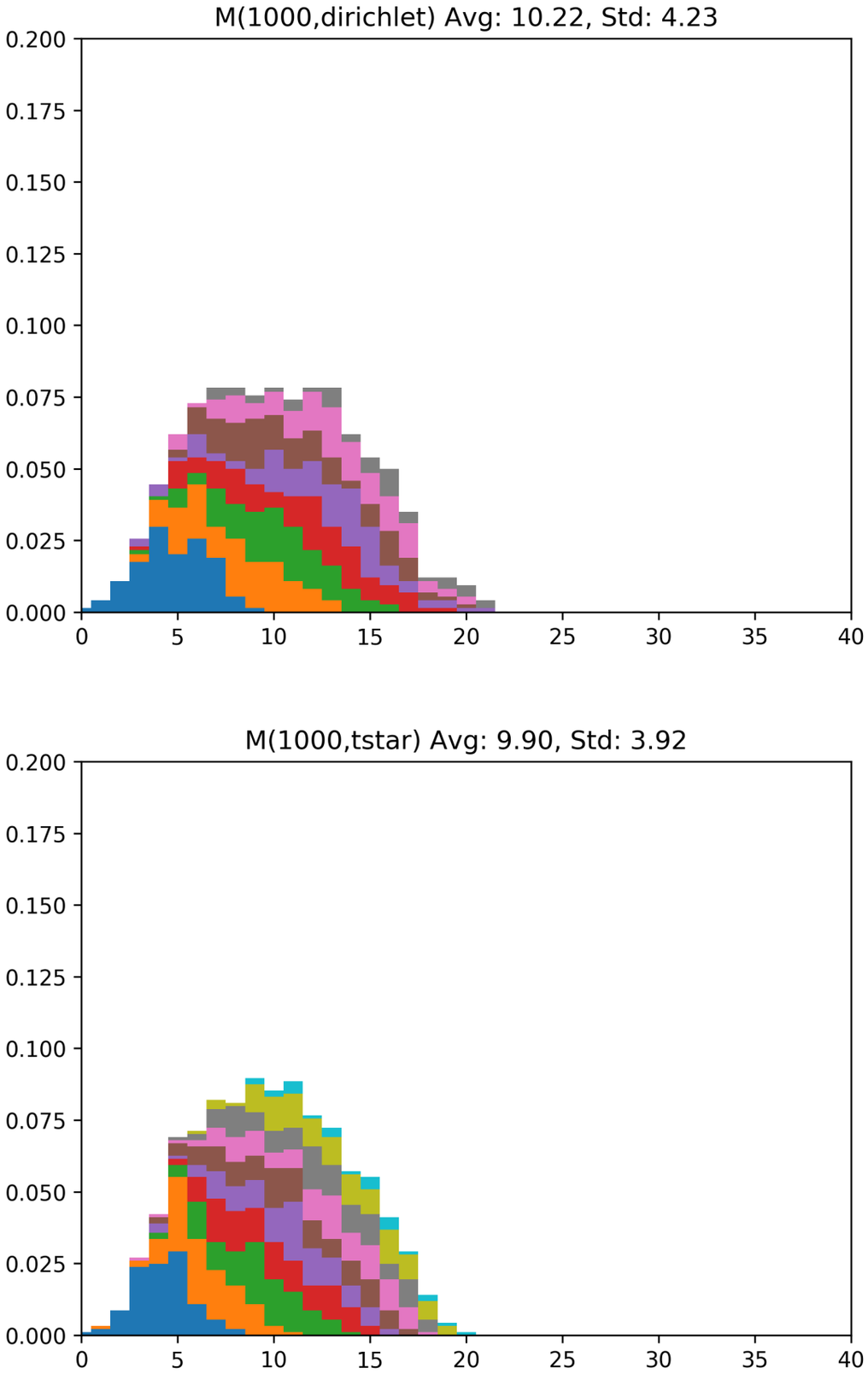
- **the near-sighted-ness of the policy**
- **the imperfect knowledge on the domain**

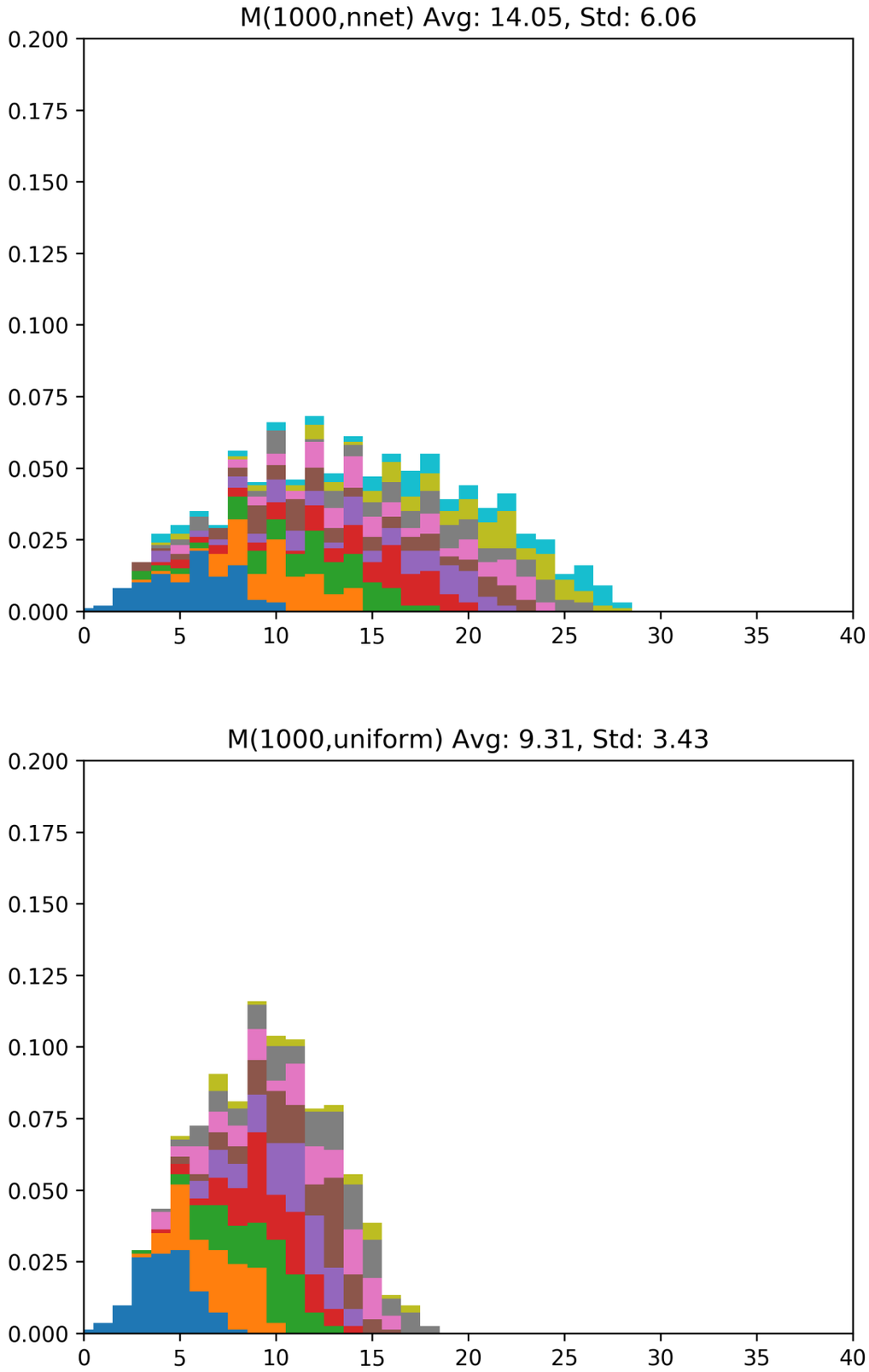
MCTS statistics

- A proxy for “how far” the MCTS is searching for making one move
- Record the game "time" of each leaf node for 100-1000 MCTS iterations.
- Colors: the number MCTS iterations. Blue for 100. Cyan for 1000
- Horizontal axis: game time
- Vertical axis: normalized count

Orange on top of blue means:
going from 100 MCTS to 200 MCTS,
how many additional leaf nodes were explored.




M(1000,dirichlet) Avg: 10.22, Std: 4.23
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40


M(1000,nnet) Avg: 14.05, Std: 6.06
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40

M(1000,tstar) Avg: 9.90, Std: 3.92
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40

M(1000,uniform) Avg: 9.31, Std: 3.43
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40

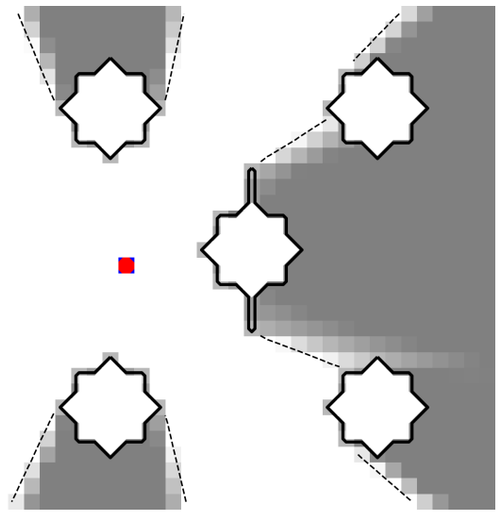
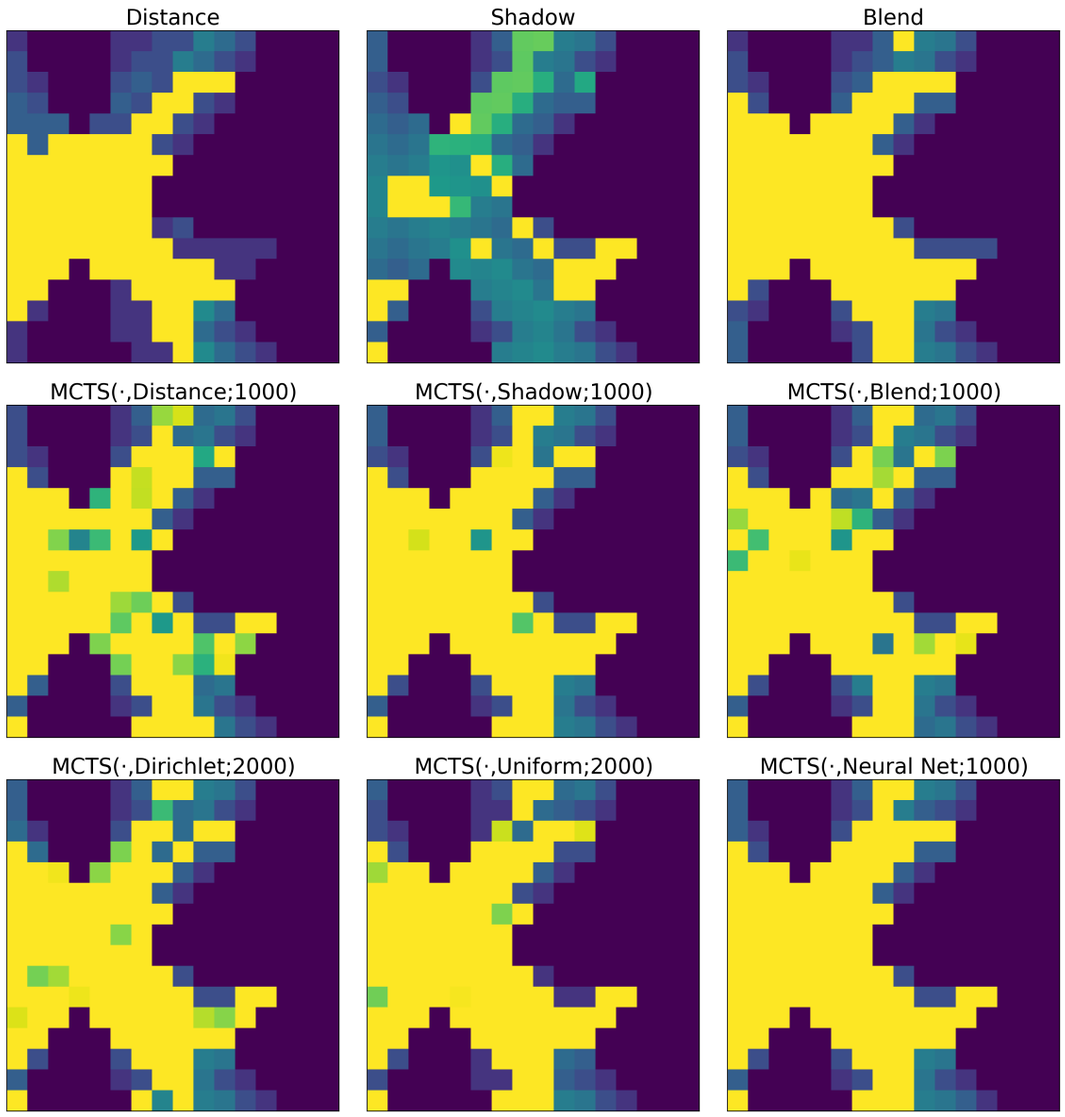


Table 3.2: Game statistics for the 1 pursuer vs 1 evader game with 5 circular obstacles, where $f_P = 2$ and $f_E = 1$.

Method	Win % (137 games)	Average game time
Distance	52.55	53.56
Shadow	16.06	22.36
Blend	63.50	64.45
MCTS(·, Distance; 1000)	55.47	62.40
MCTS(·, Shadow; 1000)	67.15	69.45
MCTS(·, Blend; 1000)	58.39	63.27
MCTS(·, Uniform; 2000)	60.58	65.84
MCTS(·, Dirichlet; 2000)	65.69	69.02
MCTS(·, Neural Net; 1000)	70.80	71.61



Comparable computational resource at run time for each method.

2v2 game

Slice of the computed value functions

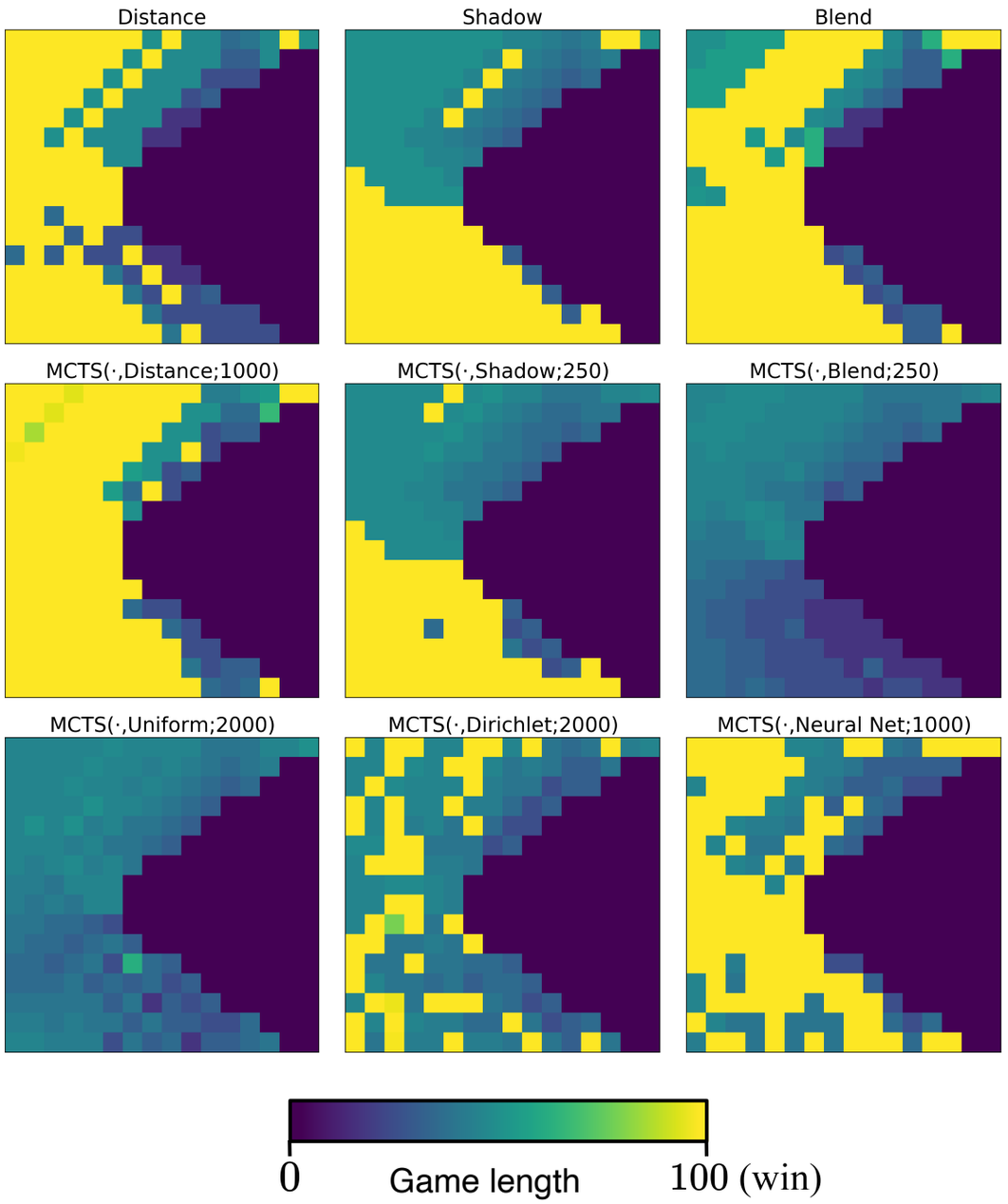


Table 3.3: Game statistics for the 2 pursuer vs 2 evader game with a circular obstacle.

Method	Win % (162 games)	Average game time
Distance	56.8	58.8
Shadow	46.3	50.1
Blend	65.4	67.9
MCTS(·, Distance; 1000)	73.5	76.6
MCTS(·, Shadow; 250)	40.7	44.5
MCTS(·, Blend; 250)	00.0	4.4
MCTS(·, Uniform; 2000)	00.0	5.3
MCTS(·, Dirichlet; 2000)	27.8	32.8
MCTS(·, Neural Net; 1000)	59.9	61.7

All players have the same maximum speed

Comparable computational resource at run time for each method.

Observations and some early thoughts

- We are trying to learning the optimal policy and its dependence on the domain geometry
- The games are played in previously unseen environments
- The performance of the heuristic policies tied to how tightly packed are the obstacles and the players
- The neural networks are probably trained with insufficient data (hasn’t learnt to use the distance policy in the 2v2 game, for example)
- The is probably a better network architecture for our purpose

Observations and some early thoughts

- MCTS can be used to refine non-optimal policies
- One needs sufficient number of MCTS steps to benefit from it
- MCTS can be expensive
- We are currently developing a multi-resolution MCTS to improve
- Search algorithms aims at getting good outcome for each instance of the game, instead of being optimal on average (stochastic control).

Thanks for your attention!

Stay safe.