ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

AI CENTER

Planning and Acting in Dynamic Environments

Lukáš Chrpa

Intelligent Acting

- Intelligent entities (agents) **reason about how to act** to achieve their goals
- **Reactive** acting
  - Rule based, Reinforcement Learning
  - Fast
  - Aims for short-term goals (rewards)
- **Deliberative** acting
  - Planning
  - Slow
  - Aims for longer-term goals

Automated Planning

- We have **Domain Definition languages** (e.g. PDDL)
- We have **Planning Engines** (e.g., FF, LAMA, LPG, FDSS, BFWS,...)
- So, we can generate **Plans** (quite easily)

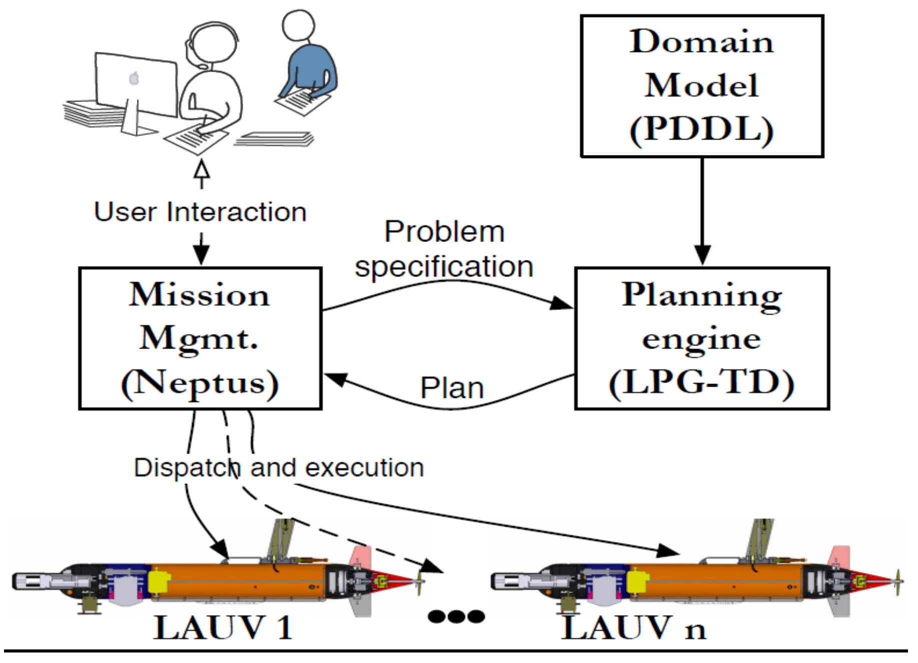
- But what about their **execution**

Task Planning for AUVs

- Necessity to control **multiple heterogeneous AUVs** for fulfilling user-defined tasks (e.g. sampling an object of interest)
- System has to be **flexible** (e.g. a user can add a new task) and **robust** (e.g. handling vehicles' failures)
  - Automatized response on task changes by user and/or exceptional circumstances during plan execution

“One shot” planning Modular Architecture [Chrpa et al., 2015]

- **User specifies tasks** in NEPTUS (the control system developed in LSTS, Univ. of Porto)
- NEPTUS **generates a planning problem** and sends it to the LPG-td planning engine
- LPG-td **returns a plan** to NEPTUS
- NEPTUS **distributes the plan** to each of the vehicles

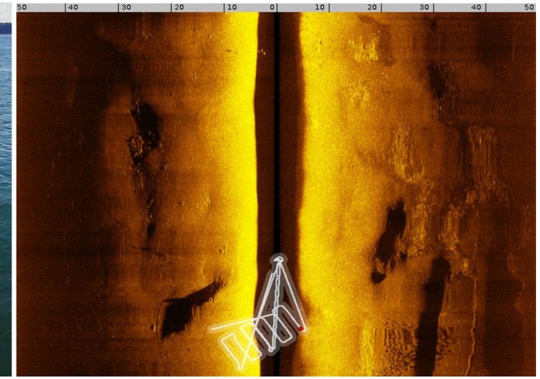


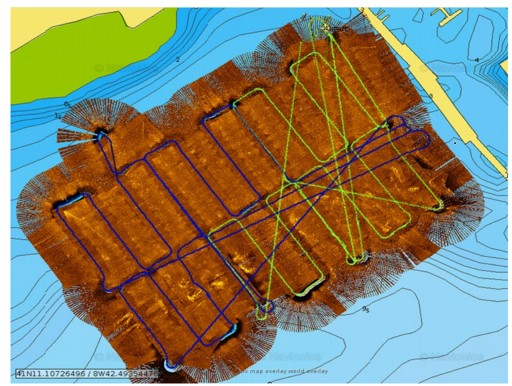
Domain Specification (sketch)

- The user specifies **tasks** by
  - **Locations/areas** of interest
  - Required **payloads** (e.g. camera, sidescan)
- The vehicle can perform the following **actions**
  - **Move** (moving between locations)
  - **Sample/Survey** (sampling the location/surveying the area of interest by a required payload)
  - **Communicate** (communicate task data with control center while being in its “depot”)

Experimental Settings

- Evaluated in Leixões Harbour, Porto
- Mine-hunting scenario was used
- 3 light AUVs, 2 carried sidescan, one carried camera
- In phase one, areas of interest were surveyed
- In phase two, contacts identified in phase one sampled to identify them as mines, or false positives





Planned vs. Execution time

- The plans were **executable**
- **High discrepancies**, especially for move and survey actions
- **Rough time predictions** that were done only on distance and type of vehicle

Vehicle	Action	Time Difference (s)
Noptilus-1	move	47.80 ± 49.11
	survey	23.15 ± 23.26
	sample	1.33 ± 0.58
	communicate	0.16 ± 0.17
Noptilus-2	move	39.57 ± 35.66
	survey	107.88 ± 141.10
	sample	N/A
	communicate	0.25 ± 0.07
Noptilus-3	move	59.90 ± 57.05
	survey	24.00 ± 0.00
	sample	9.57 ± 13.64
	communicate	0.11 ± 0.16

Additional Requirements [Chrpa et al., 2017]

1) Users can **add, remove or modify tasks** during the mission
   - Plans have to by (dynamically) amended

2) Vehicles might **fail to execute an action**
   - Tasks have to be (dynamically) reallocated to another AUV

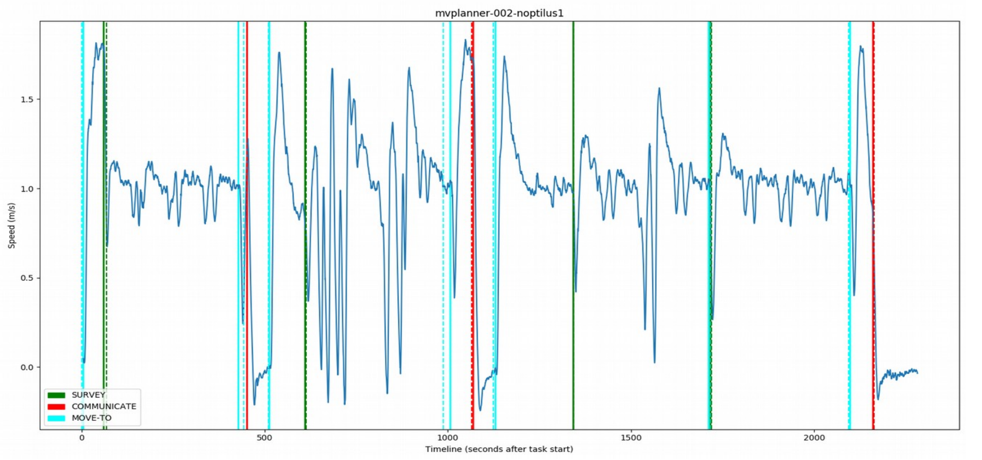
3) **Communication** with the control center is possible only **when a vehicle is in its “depot”**
   - The user defines a **maximum “away” time** for each vehicle (the vehicle has to return to its “depot” in that time)

Execution

- Preprocessing
  - Splitting large surveillance areas into smaller ones
- Planning
  - NEPTUS generates a problem specification in PDDL, runs LPG-td, then processes and distributes the plan among the vehicles
- Execution
  - Each vehicle is responsible for executing its actions
  - Move actions are translate into timed-waypoints for mitigating the differences between planned and actual times
  - When in depots vehicles communicate status of completed tasks (success/failure) – failed tasks are “re-inserted”
- Replanning
  - If a new planning request comes (e.g. a user added a new task), vehicles continue to execute their current plans until they come back to their depots, then they receive new plans

Results of the Field Experiment

- Plans were successfully executed
- During one of the executions one AUV (Noptilus 3) failed (depth sensor fault) – tasks were automatically re-inserted and allocated to a different AUV, which completed them



Most planned/actual differences are quite small (less than 3 seconds).

Around time 1000 a noticeable difference occurred (vehicle had to ascend during the survey). The delay was eliminated by accelerating during the following move action.

Executing Plans

- **In theory** (static environment)
  - Actions in a plan are always applicable (one by one)
  - After all actions are executed the goal is reached
- **In practice** (dynamic environment)
  - Actions might become **inapplicable** (at some point) because of **external factors**
  - **Goal might not be reached** even if all the actions were executed
  - The agent might “fall” into a **dead-end state**

Planning vs Execution (the AUV case)

- Issues we considered (to some extent)
  - User intervention (e.g. adding tasks)
  - Task failures
  - Vehicles delays
  - Lack of communication
- Issues we didn’t consider
  - Ships passing the area (or other non-deterministic events)
  - Currents, obstacles
  - …….

Non-deterministic events

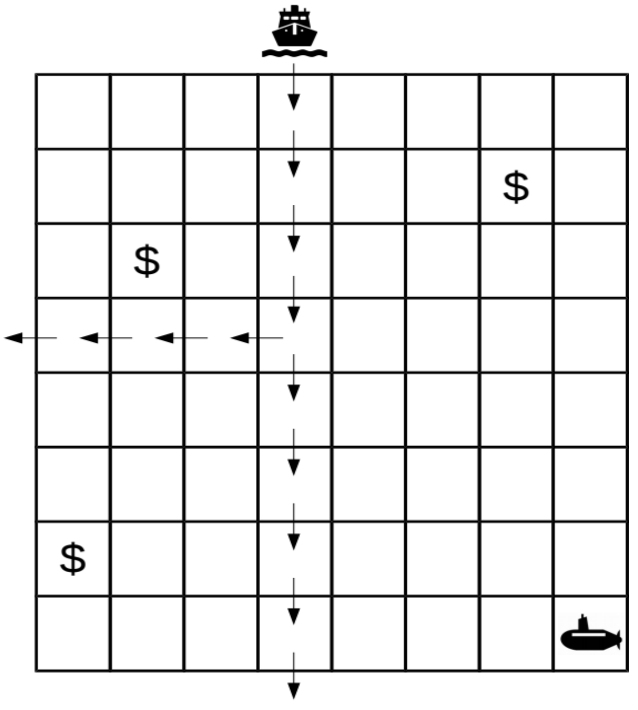
- **Events** are encoded similarly to actions – they have **preconditions, add** and **delete effects**
- A non-deterministic event can occur if its precondition is met (but doesn't necessarily have to)
- We assume, for simplification, a "two-player" like scenario
  - The **controller** applies an **action** (including "noop")
  - The **environment** applies a set of independent **events** (including "noop")

Planning with non-deterministic Events

- Generate “strong plans” (handling all non-deterministic alternatives)
  - computationally very expensive
- Naive Planning and Replanning
  - relax the non-determinism
  - replan if something is “wrong”
  - prone to dead-ends
- Enhancing (classical) planning techniques by reasoning with safe or “dangerous” states

The AUV Domain

- An AUV moves and collects resources in a grid-like environment
- Ships can move in certain grid cells
- Ships are not controlled by the agent
- If a ship runs over the AUV, the AUV is destroyed
- The movement of ships is represented by **non-deterministic events**



Navigating between Safe States
[Chrpa et al., AAAI 2020]

- A **safe state** is a state in which no sequence of events lead to dead-end
- A **robust plan** is a plan that can always be applied and goal reached despite event occurrence
- A **reference plan** is the initially generated plan such that the number of consecutive “unsafe” actions is minimized as safe states should be “reasonably close” to each other
- The idea is that **planning and acting** consists of generation and execution of **robust plans** between **safe states**

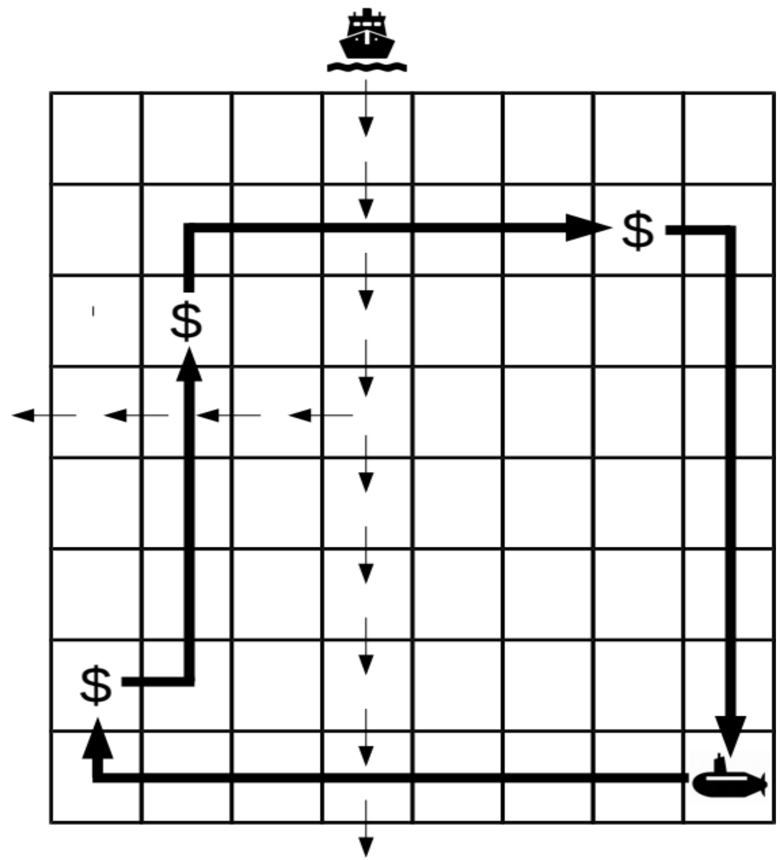
Robust Plans

- We approximate **robust plan** generation by **pessimistic** assumption of **action applicability** and **optimistic** assumption of **event applicability**
  - p+ – atoms that could have been added by events (but not deleted by actions)
  - p- – atoms that could have been deleted by events (but not added by actions)
  - event applicability $pre(e) \subseteq s \cup p+$
  - action applicability $pre(a) \subseteq s \setminus p-$

Safe State Reasoning in Planning and Acting

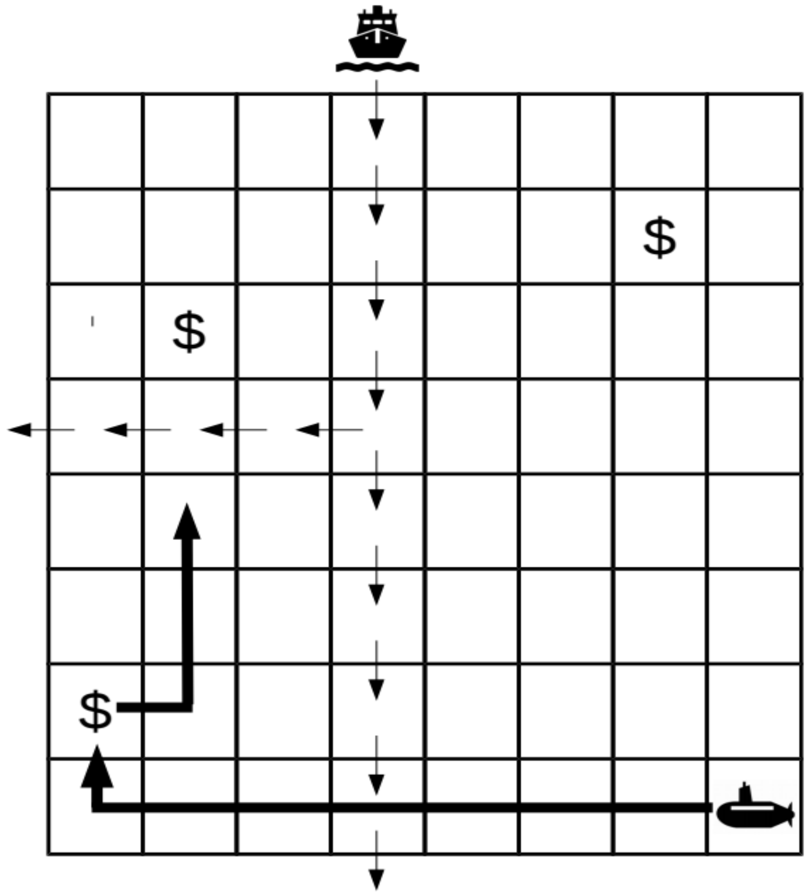
- Try to generate a robust plan (if possible, just execute it !)
- Try to generate a reference plan with increasing unsafeness limit (if it fails, stop)
- Iterate until the goal is reached
  - Identify k actions forming a robust plan and finishing in a safe state
  - If k>0, apply the k actions
  - If k=0, try to generate a robust plan to the next safe state, if it exists, execute it, otherwise wait

Example



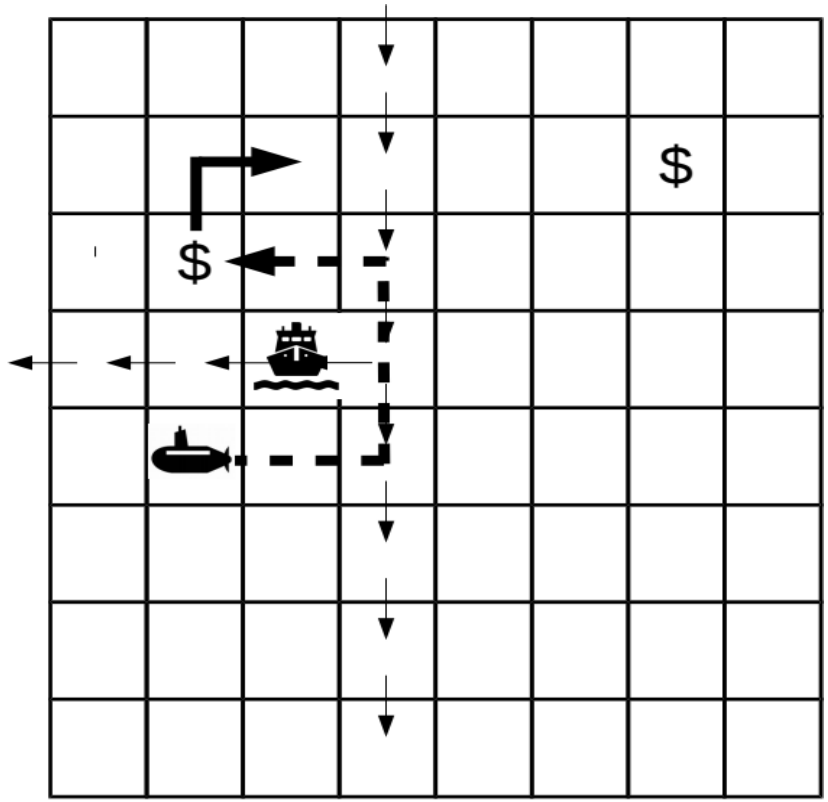
A reference plan (with the unsafeness limit of 1)

Example



The maximum length robust plan from the reference plan

Example



A robust plan around the ship (to the next safe state)

Observations

- The approach **guarantees not “falling”** into **dead-ends**.
- **Planning time is very low** (compared to e.g. FOND planning)
- It might be the case that we might never find a robust plan to connect given safe states and hence **the agent might get stuck**

Dark Dungeon domain

- The hero has to **navigate through the dungeon** full of traps and monsters
- The hero can **use the sword** (if s/he found it) to **eliminate monsters**
- The hero can **disarm traps** but must be **empty handed**
- **Monsters can move** (they cannot be in a room with a trap or another monster) and eventually **eliminate empty handed hero**

Reasoning about “dangerous” states [Chrpa et al., 2017,202?]

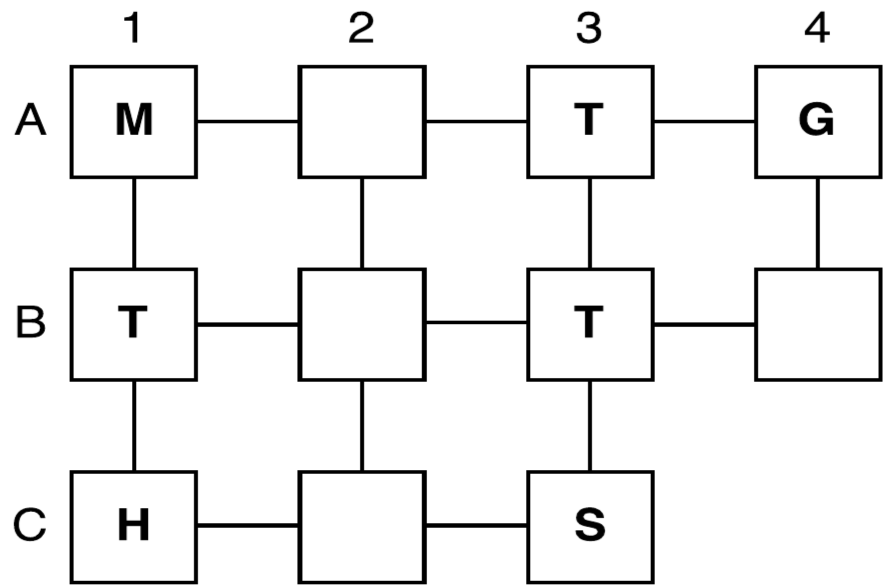
- Considering all non-deterministic alternatives might not be feasible and safe states are sparse
- However, the **controller** should still **avoid dead-ends**
- The **controller** needs to know if it is in a **dangerous state**, i.e., a state “close” to a dead-end state, so it can avoid “falling” into it

Dangerous States

- A state is
  - **0-dangerous** if it's a dead-end state
  - **n-dangerous** if events (without controller's actions) might transform it to a dead-end state in *n* steps
  - **Safe** (∞-dangerous) otherwise
- The **dang** function determines how dangerous the state can be (the worst case scenario) after executing a given sequence of actions

An example of dangerousness

- The initial state (I) is 4-dangerous
- dang(I, 〈right〉) = 2
- dang(I, 〈right,up〉) = 0
- dang(I, 〈right,right〉) = 2
- dang(I, 〈right,right,pickup〉) =∞



Meta-reasoning on Dangerous states

- **When in “dangerous” state** (the value of *dang* less than a given threshold) **the controller**:
  - **Reactively escapes the danger**, i.e, executes actions maximizing the value of *dang*
  - **Plans towards a safe state**
  - **Plans towards eliminating the source of the danger**
- **After escaping the danger** (the value of *dang* is above the threshold), **the controller plans towards the goal**

Considered Agents (baseline)

- **R1 –** behaves reactively according to given rules
- **N1 –** re-plans whenever an event changes the state of the environment
- **N2 –** re-plans when the current action is inapplicable

Considered Agents (clever)

- **C1** – if the current state is “dangerous” (2-dangerous or worse), then it plans to eliminate the source of danger
- **C2 -** if the value of the *dang* function is small (2 or less), then it plans to eliminate the source of danger
- **C3** - if the current state is “dangerous” (2-dangerous or worse), then it reactively moves to a safer state (3-dangerous or better), and then it plans to eliminate the source of danger

Results

Ag.	W	L	T/O	SR	Ws	Wt	PC	PF
N1	4879	706	15	0.87	45.5	48.8	136.5	6.49
N2	4086	1512	2	0.73	38.6	1.2	4.1	0.03
R1	3695	562	1343	0.66	45.2	0.0	0.0	0.00
C1	5040	555	5	0.90	49.7	13.2	36.1	3.38
C2	5113	483	4	0.91	50.6	11.3	40.2	3.04
C3	4785	706	109	0.85	53.3	15.6	30.7	8.90

Agents' (W)ins, (L)osses, and time-outs (T/O); their success rate (SR), winning steps (Ws, thousands) and wining time (Wt, seconds); number of planner calls and planner fails (PC and PF, thousands)

- C1-C3 and N1 have good success rate (85% or more)
- N2 and R1 have a small "winning" time but low success rate (less than 75%)
- N1 has a high "winning" time and a lot planner calls
- C1 and C2 have success rate above 90% while

Results cont.

movement prob.	N1	N2	R1	C1	C2	C3
0.0	0.999	0.999	0.731	0.997	0.997	0.919
0.1	0.916	0.714	0.674	0.928	0.927	0.884
0.2	0.856	0.661	0.665	0.888	0.901	0.857
0.5	0.714	0.544	0.569	0.787	0.826	0.759

The success rate of the different types of agents in dungeons with different monster movement probabilities

- N2's success rate is reduced considerably with increasing "dynamicity"
- C1-C3's success rates decrease "more slowly" than for N1 and N2
- C2's success rate is above 80% even for "more dynamic" environments

Summary

- External factors (e.g., events) are often part of the environment
- One can still (to some extent) leverage classical (or deterministic) planning
  - (PO)MDPs or FOND techniques usually don't scale well
  - MCTS might be less informative if not many alternatives are "viable"
  - Reinforcement Learning might not be efficient for longer-term goals/rewards