Towards Systematically Engineering Adaptive Systems using Machine-Learning Techniques

Martin Wirsing** *
Ludwig-Maximilians-Universität München

**In cooperation with
Lenz Belzner and Thomas Gabor
Maiborn & Wolff LMU München

*Thanks to Rolf Hennicker, Alexander Knapp, Matthias Hölzl and all other former members of the ASCENS project

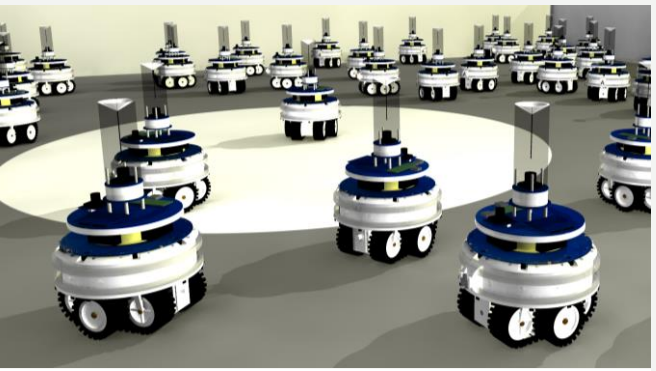
Seminar Dynamische und Adaptive Systeme, WS 2019/20, LMU München

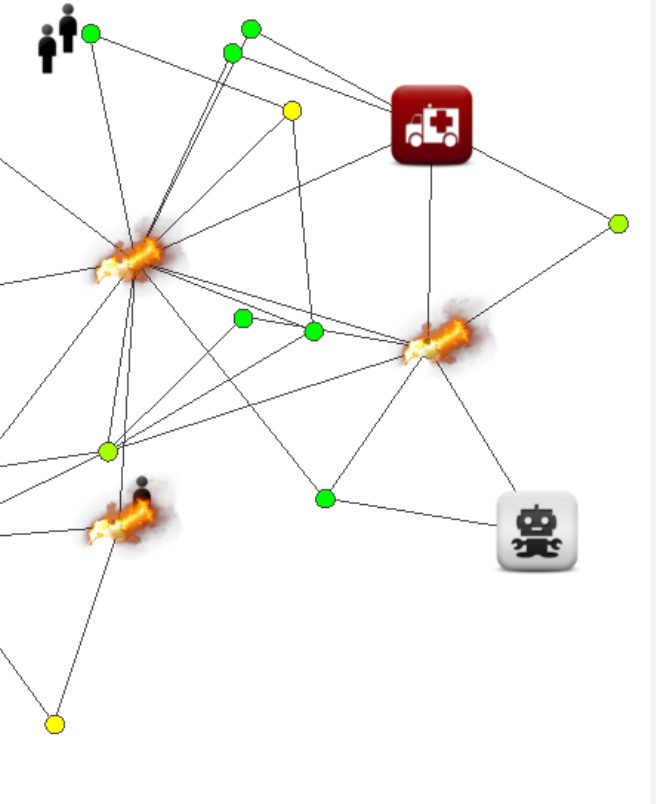
Contents

- What is an adaptive system?
- A development lifecycle for adaptive systems and ensembles
- Adapting behaviours by online planning
- Safe learning: Policy synthesis from requirements
- Concluding remarks

1. What is an adaptive system?

Adaptive system





- **An adaptive system adjusts its behaviour to multiple situations:**
  - **change of**
    - **environment**
      - human/other agents behavior
      - network infrastructure
    - **goals/requirements**
    - **the system itself**
  - or any combination thereof

- We distinguish between
  - **black-box adaptation and**
  - **white-box adaptation**

Adaptation Requirements: Black-Box Adaptation

- **Goal satisfaction in an environment**

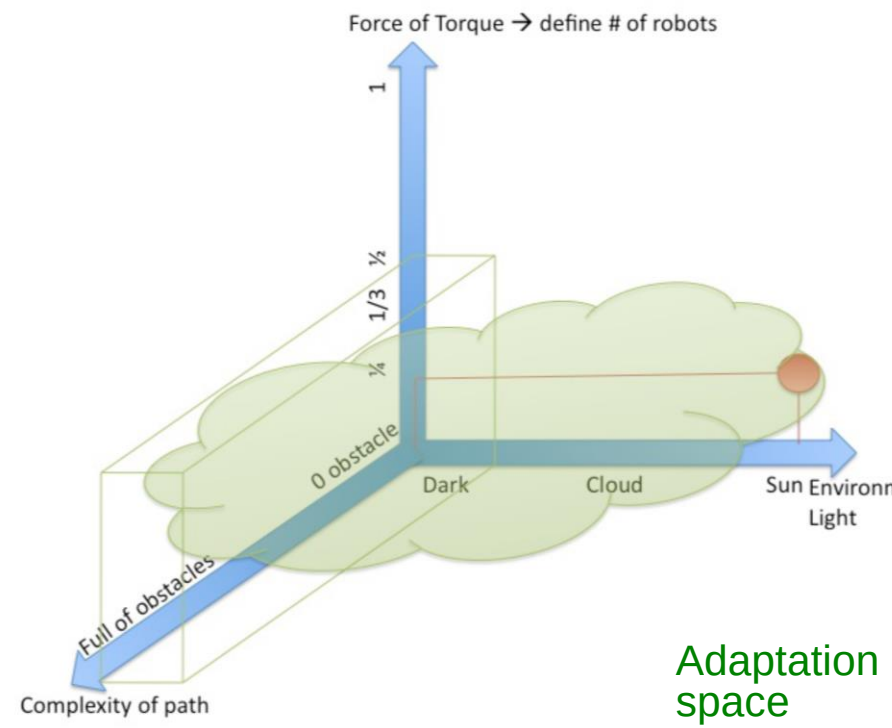
  For any environment η, system S, goal γ:

  $\eta, S \models \gamma$

- An **adaptive system adjusts to** a change of
  - goals/requirements: $\eta, S \models \gamma'$
  - environment: $\eta', S \models \gamma$
  - the system itself: $\eta, S' \models \gamma$

- **Adaptation space**
  - **Adaptation domain $\mathcal{A} \subseteq$ Env × G**
  - *S* **can adapt to $\mathcal{A}$**, written $S \Vdash \mathcal{A}$:

    $S \Vdash \mathcal{A}$ iff $\forall (\eta, \gamma) \in \mathcal{A}.\ \eta, S \models \gamma$

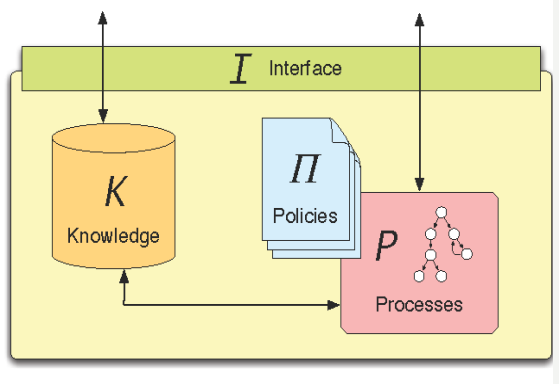


Adaptation space

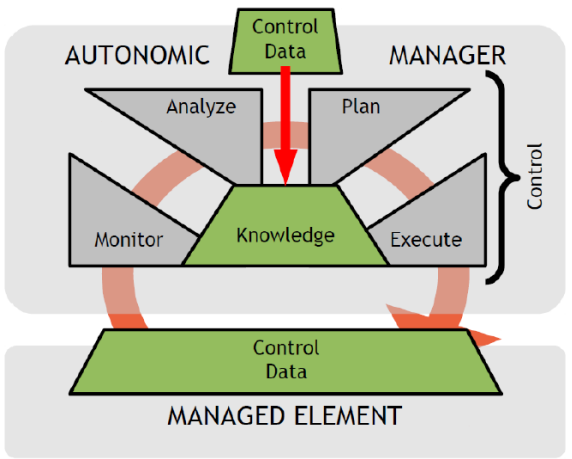
[M. Hölzl, MW: Towards a System Model for Ensembles. Formal Modeling: Actors, Open Systems, Biological Systems 2011]

Adaptation Modelling: White-Box Adaptation

- **White-box adaptation**
  - Modify control mechanism at runtime
    so that the (adaptation) requirements are satisfied



- **Approaches**
  - **Classical adaptation techniques**
    E.g. programming using modes, reconfiguration, policies
  - **AI adaptation techniques**
    E.g. using planning, learning, reasoning,
    swarm algorithms



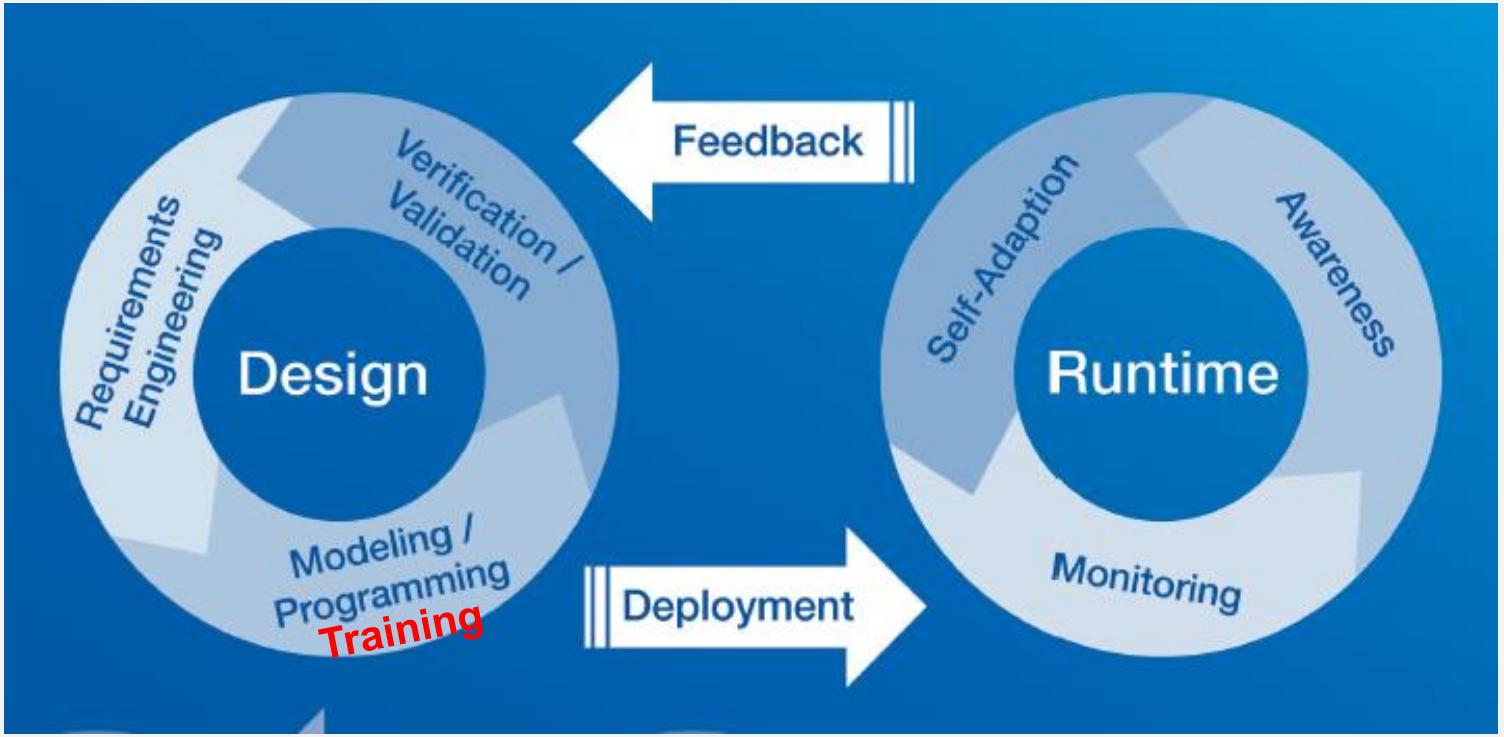
[Bruni, Corradini, Gaducci, Lluch Lafuente, Vandin : A Conceptual Framework for Adaptation. FASE 2012; Bruni, Corradini, Gaducci, Hölzl, Lluch Lafuente, Vandin, MW: Reconciling White-box and Black-Box Adaptation, ASCENS book 2015]

2. Recap:

DevAdapt

A development lifecycle for adaptive systems

ASCENS Approach: „DevAdapt“ Lifecycle



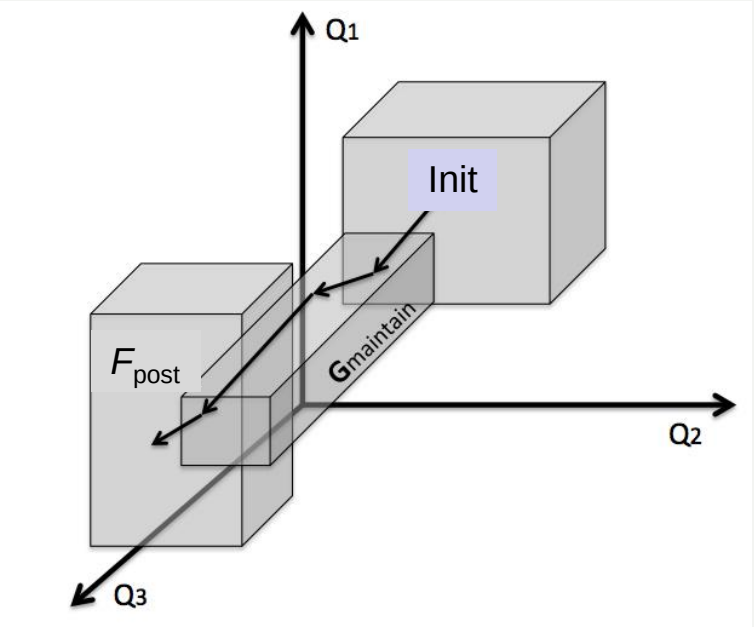
Agile lifecycle: DevAdapt

- Iterations at development time and runtime connected by deployment and feedback
- 3 feedback loops
- Cf. DevOps life cycle for integrating development and operations

[N. Koch et al.: Life Cycle for the Development of Autonomic Systems: The e-Mobility Showcase, 3rd Workshop Self-Awareness in Autonomic Systems, 2013; Compuware Mainframe DevOps, https://www.compuware.com/lifecycle-overview/]

Develop “SOTA/Gem” **requirements specification** consisting of

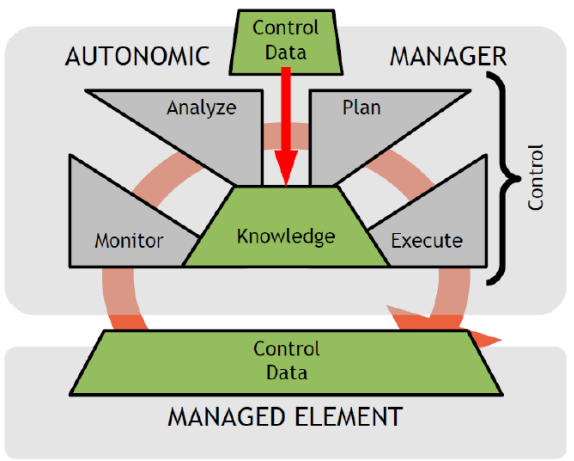
- **environment specification**
- **goal-oriented specification** of the adaptive system with
  - Maintain goals of form $G_{maintain}\ \phi$
  - Achieve goals of form $F_{post}\ \phi$
- **adaptation/awareness requirements** (adaptation space)



[M. Hölzl, MW: Towards a System Model for Ensembles. Formal Modeling: Actors, Open Systems, Biological Systems 2011; D. Abeywickrama, M, Mamei, F. Zambonelli: Engineering Collectives of Self-driving Vehicles: The SOTA Approach. ISoLA (3) 2018: 79-93]

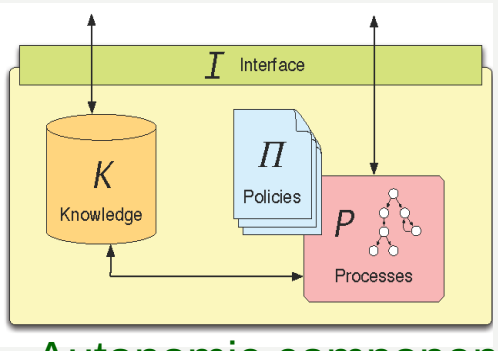
Modelling

- **Architecture**
  - Adaptation patterns such as
    - Reactive component
    - MAPE-K architecture of autonomic manager

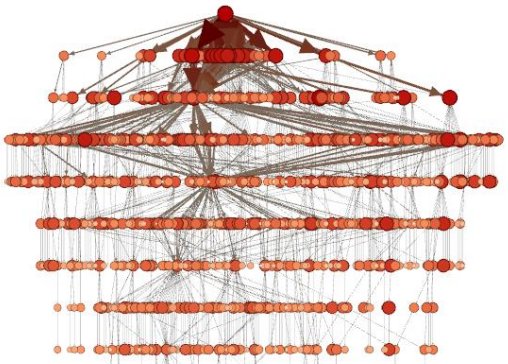


MAPE-K Autonomic manager

- **Modelling and programming/training: White-box adaptation**
  - Classical adaptation, e.g.
    - Component-based programming with SCEL
  - Machine learning adaptation, e.g.
    - Reinforcement learning
    - Deep Neural Networks



Autonomic component

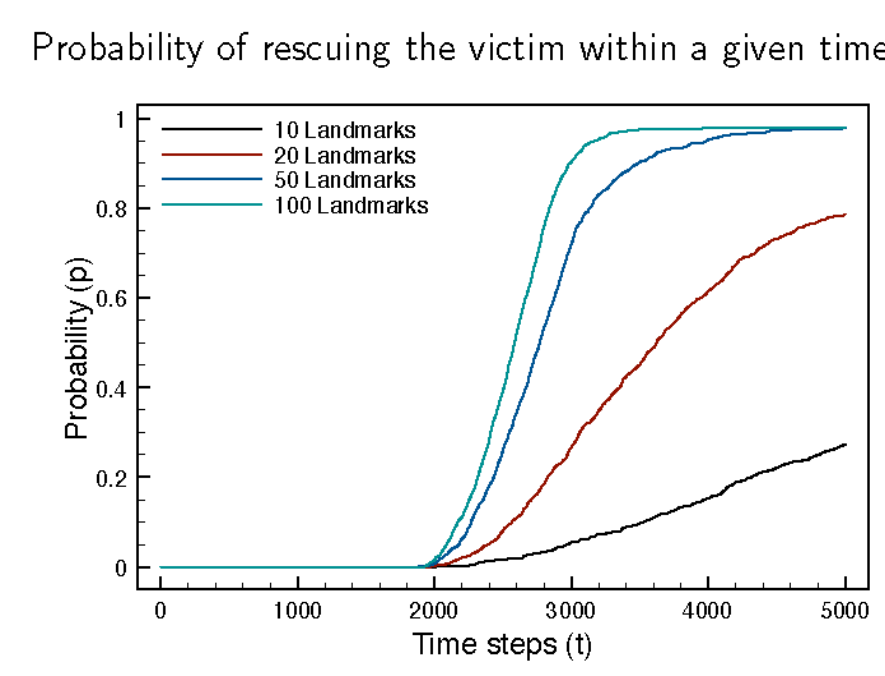


[Puviani, Cabri, Zambonelli: Patterns for self-adaptive systems: agent-based simulations. EAI Endorsed Trans. Self-Adaptive Systems 1, 2015; De Nicola , Loreti, Pugliese Tiezzi: SCEL: a Language for Autonomic Computing, ACM TAAS, 2014; Bruni, Corradini, Gaducci, Lluch Lafuente, Vandin : A Conceptual Framework for Adaptation. FASE 2012]

Validation and Verification

- Show that the **system satisfies the (adaptation) requirements** specification by
  - **Testing**
  - **Simulation**
  - **(Statistical) Modelchecking**

  using **compositional techniques, fluid flow analysis, constraint programming, …**



- **Note: Environment assumptions and adaptation space requirements must also be validated**

[J. Combaz,S. Bensalem, F.Tiezzi ,A. Margheri, R. Pugliese, J. Kofron: Correctness of Service Components and Service Component Ensembles. ASCENS Book 2015:105-158; R. De Nicola , M. Loreti, R. Pugliese, F. Tiezzi: SCEL: a Language for Autonomic Computing, ACM TAAS, 2014; L. Bulej, T. Bures, I. Gerostathopoulos, V. Horký, J. Keznikl, L. Marek, M. Tschaikowski, M. Tribastone, P. Tuma: Supporting Performance Awareness in Autonomous Ensembles. ASCENS Book 2015: 291-322]

3. Adapting Behaviours by Simulation-based Online Planning

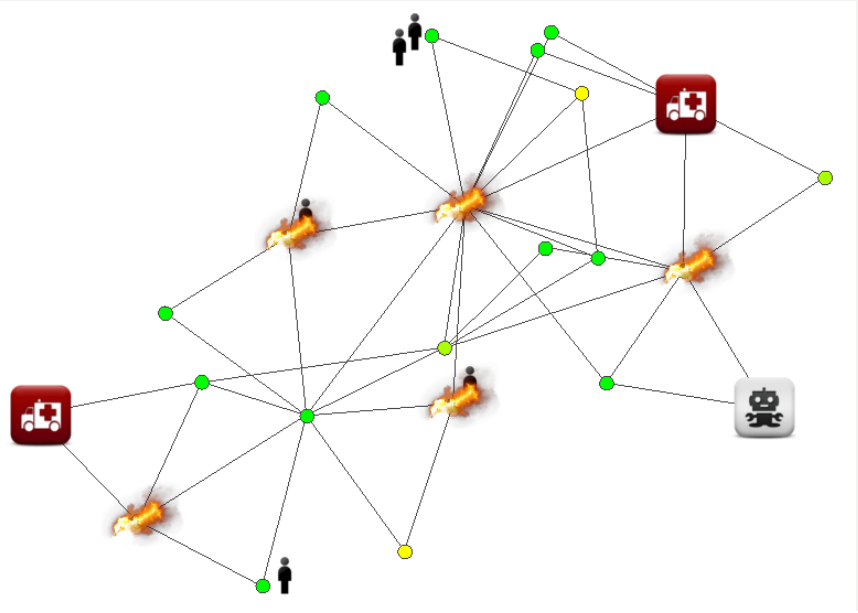
[Belzner, R. Hennicker, MW: Onplan: A framework for simulation-based online planning. In C. Braga and P. C. Ölveczky, eds., FACS 2015 LNCS 9539, Springer, 2015. 1-30.]

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Search and Rescue Case Study Requirements

Environment

- Victims, fires and ambulances
- Unknown topology
- Unknown initial situation

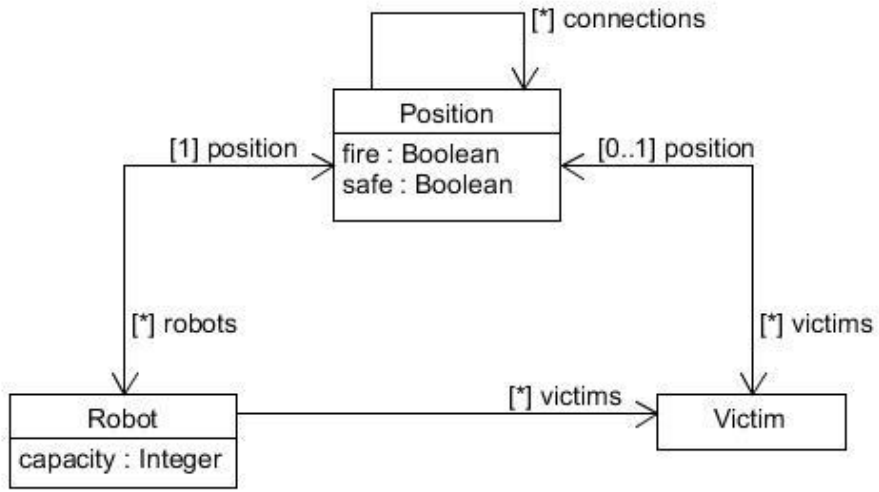


Agent actions

- Noop, Move
- Load or drop a victim
- Extinguish fire if adjacent

Agent Requirements

- Find victims and bring them to an ambulance
  - *F ($AND_{i=1,...n}$ $victim_i$ at ambulance)*
- Extinguish fire if adjacent
  - *G (robot.position.fire ➔ action.robot = extinguishFire)*

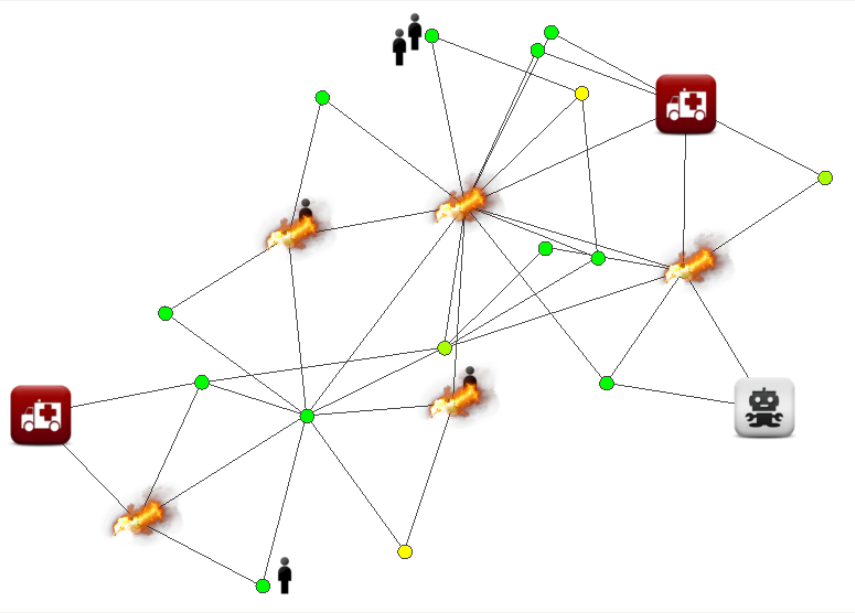


Adaptation Space

Environment and system may change

- **in a nondeterministic way**
  - Fires ignite and cease
  - Actions of agent may fail

  by unexpected events
  - New fires break out and agents drop victims



Goals of agent may change

- Goal “Save victims from fire”
  - $F\ (AND_{i=1,\ldots n} \neg\ victim_i\ at\ fire)$

  changes to

  “Save victims and bring them to ambulance”

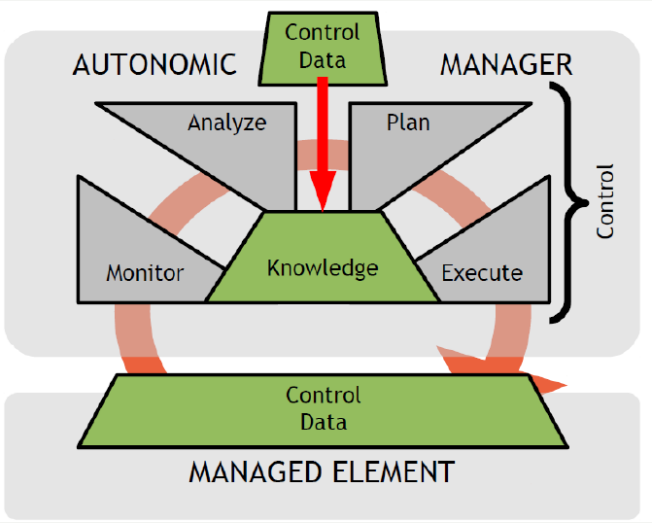
Generic Modelling

Online Planning

- Perform planning and execution of actions iteratively at runtime

Architecture

- MAPE-K loop with parallel planning



Online Planning Pseudo-Code

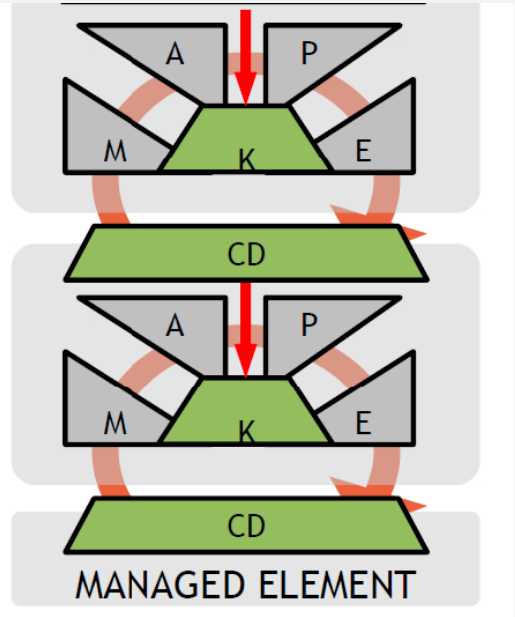
```
while true do
  observe state
  execute || plan
end while
```

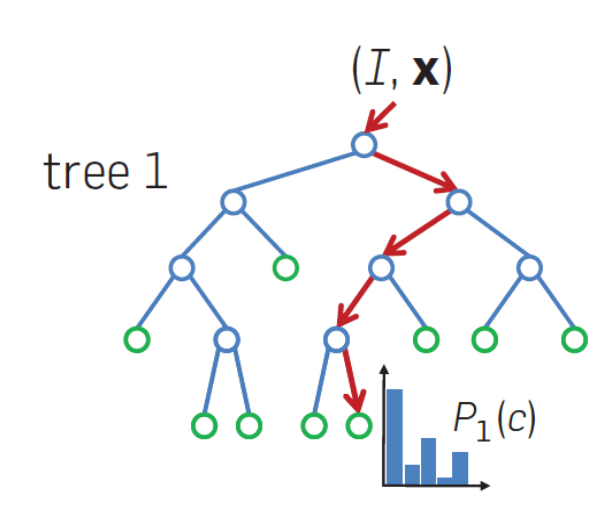
Modelling: Instantiation

Simulation-based Online Planning

simulates future episodes at runtime

- **Architecture**
  - Iterated MAPE-K loop
- **Simulation-based planning**
  - Simulate by generating *n* episodes and their rewards
  - After each episode update the current policy according to strategy
- **Strategies**
  - Monte Carlo Tree Search
  - Cross Entropy Planning

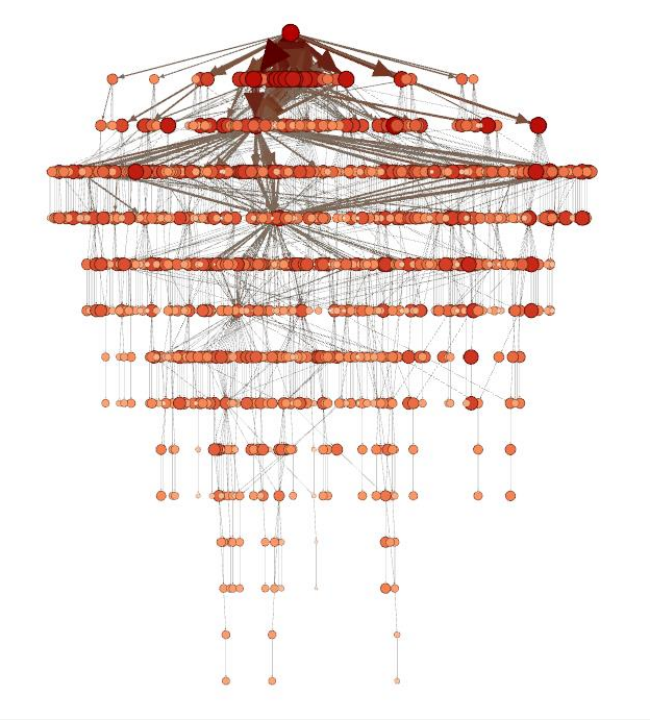


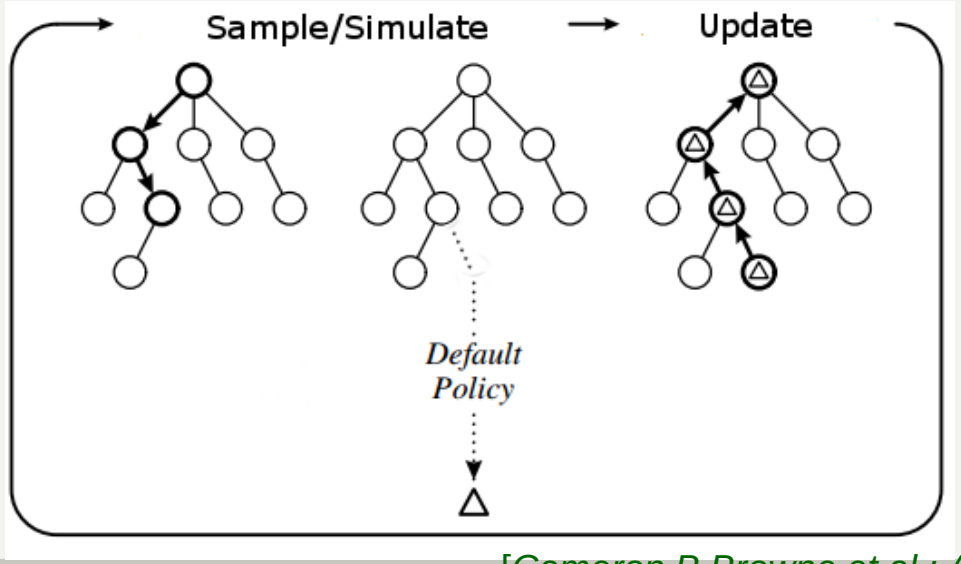


Monte Carlo Tree Search for Discrete Domains

Simulation with Monte Carlo Tree Search

- Policy as tree
  - Nodes represent states and action choices
  - Add a node per simulation
  - Aggregate simulation data in nodes
    - Reward and frequency
  - Sample actions w.r.t. aggregated data





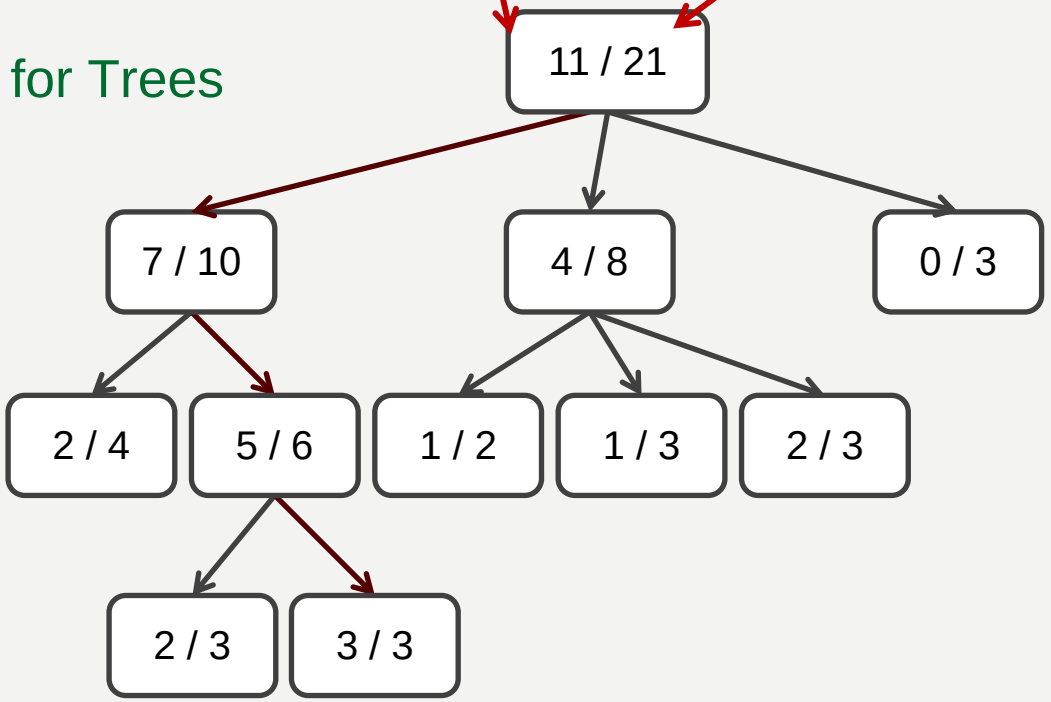
[*Cameron B Browne et al.: A survey of Monte Carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on, 4(1):1 - 43, 2012.*]

Strategy Inside the Tree

E.g. use Upper Confidence Bounds for Trees

- Select actions that maximize

$$UCT_j = X_j + 2C\sqrt{\frac{2\ln n}{n_j}}$$



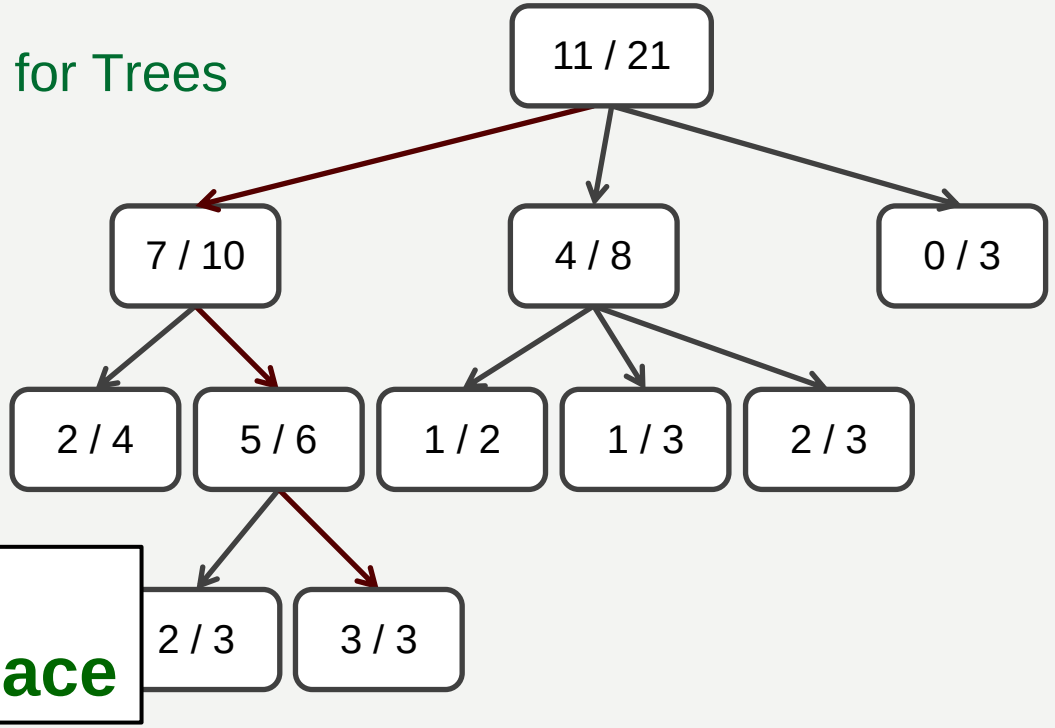
[Kocsis, Levente, and Csaba Szepesvári. Bandit-based Monte Carlo planning. Machine Learning: ECML 2006. Springer: Berlin Heidelberg, 2006, 282-293].

Strategy Inside the Tree

E.g. use Upper Confidence Bounds for Trees

- Select actions that maximize

$$UCT_j = X_j + 2C\sqrt{\frac{2\ln n}{n_j}}$$



Exploit observations

Explore solution space

X_j: Average reward of child node j

n: Nr. of episodes from current node

n_j: Nr. of episodes from child node j

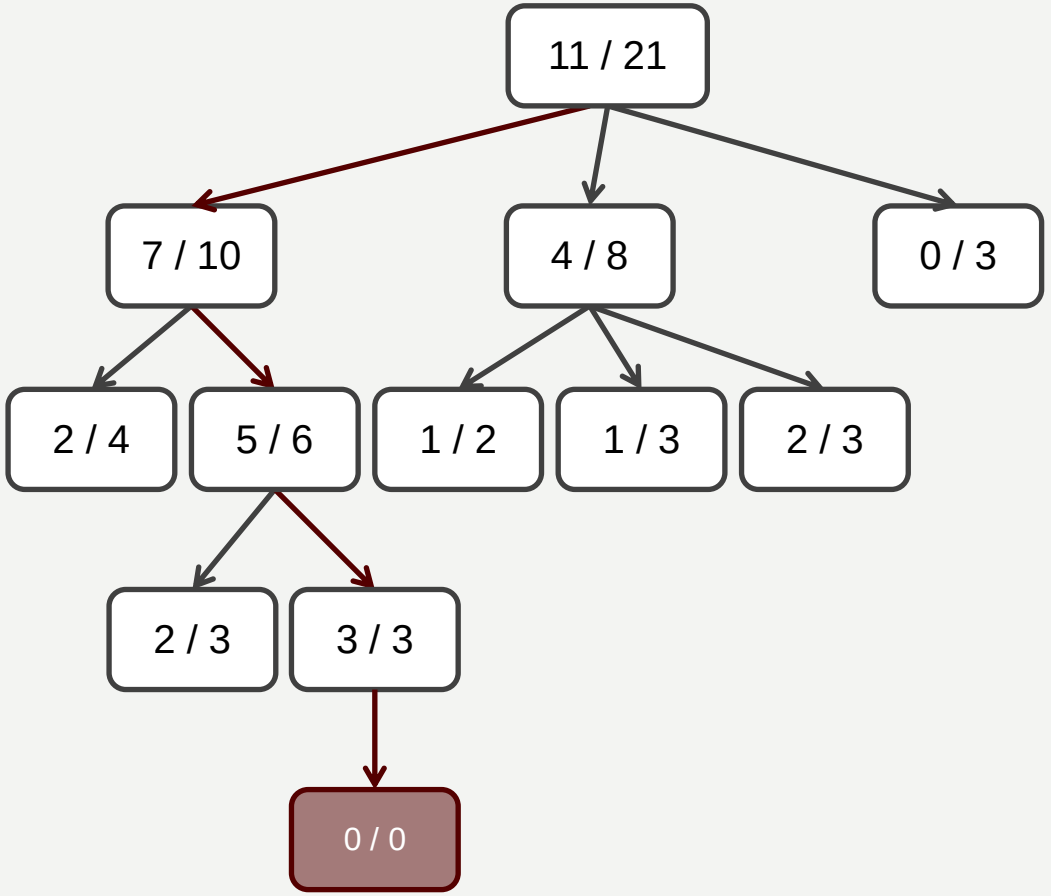
C: UCT exploration constant

[Kocsis, Levente, and Csaba Szepesvári. Bandit-based Monte Carlo planning. Machine Learning: ECML 2006. Springer: Berlin Heidelberg, 2006, 282-293].

Expand the Tree

Add a new node

- When an episode leaves the tree



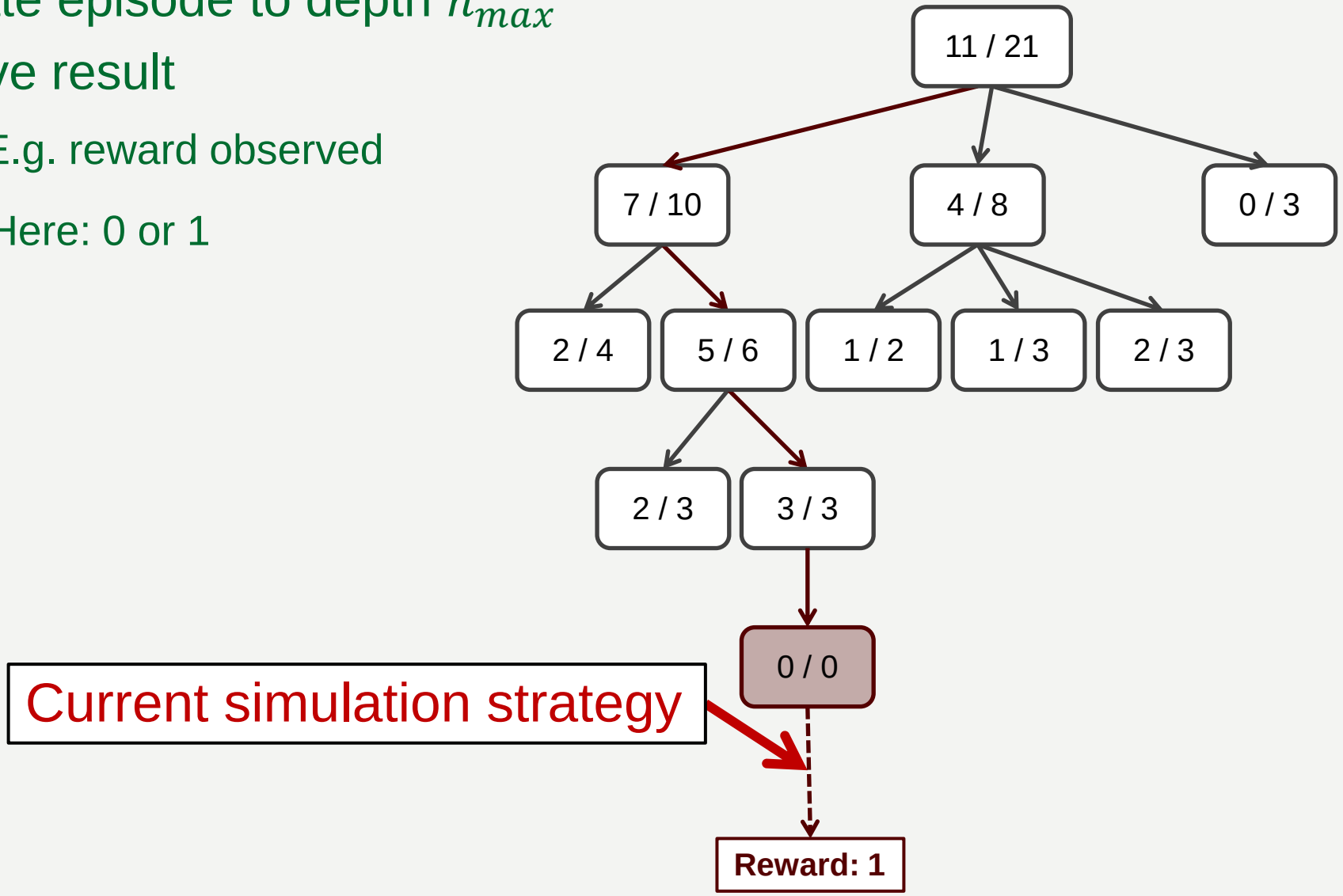
Strategy Outside the Tree

Simulate episode to depth h_{max}

Observe result

- E.g. reward observed

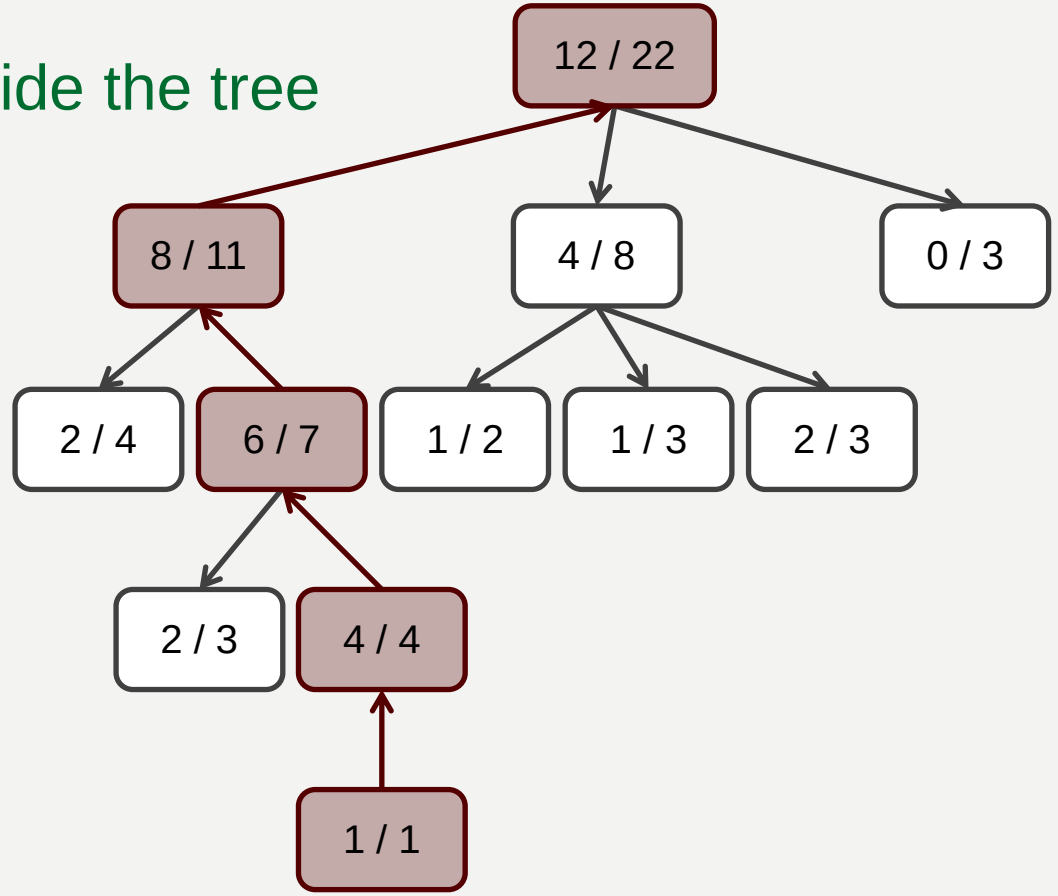
  Here: 0 or 1



Update Strategy

Update the statistics

This changes the strategy inside the tree



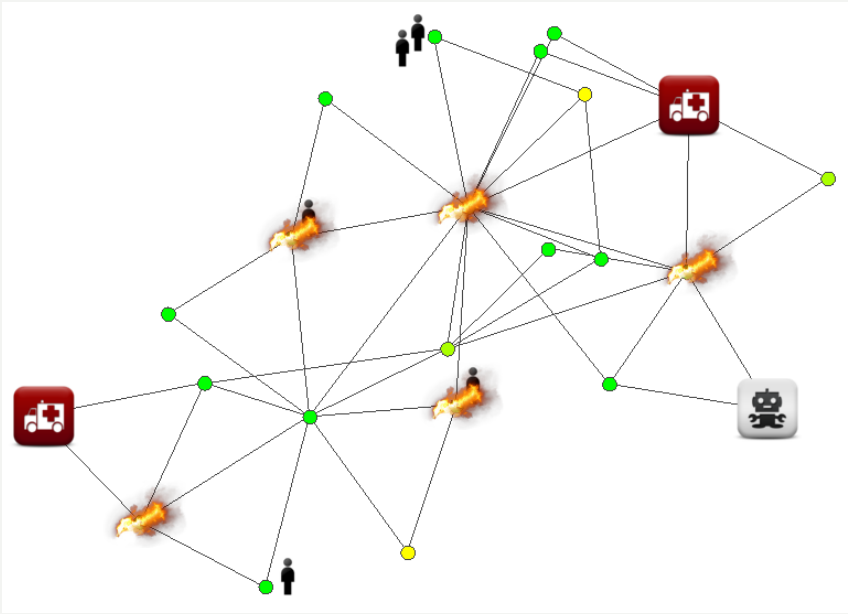
Search and Rescue Case Study: Experiment

Experiment

- Topology randomly generated with

  20 nodes, 6-7 connections/nodes
- Action failure probability = 5%
- Planning depth = 20
- Reward for victims at safe positions:

$$R(s) = 100 \cdot \Sigma_{v \in victims(s)} \#(\text{v.position.safe})$$



Validation (I)

Provided reward

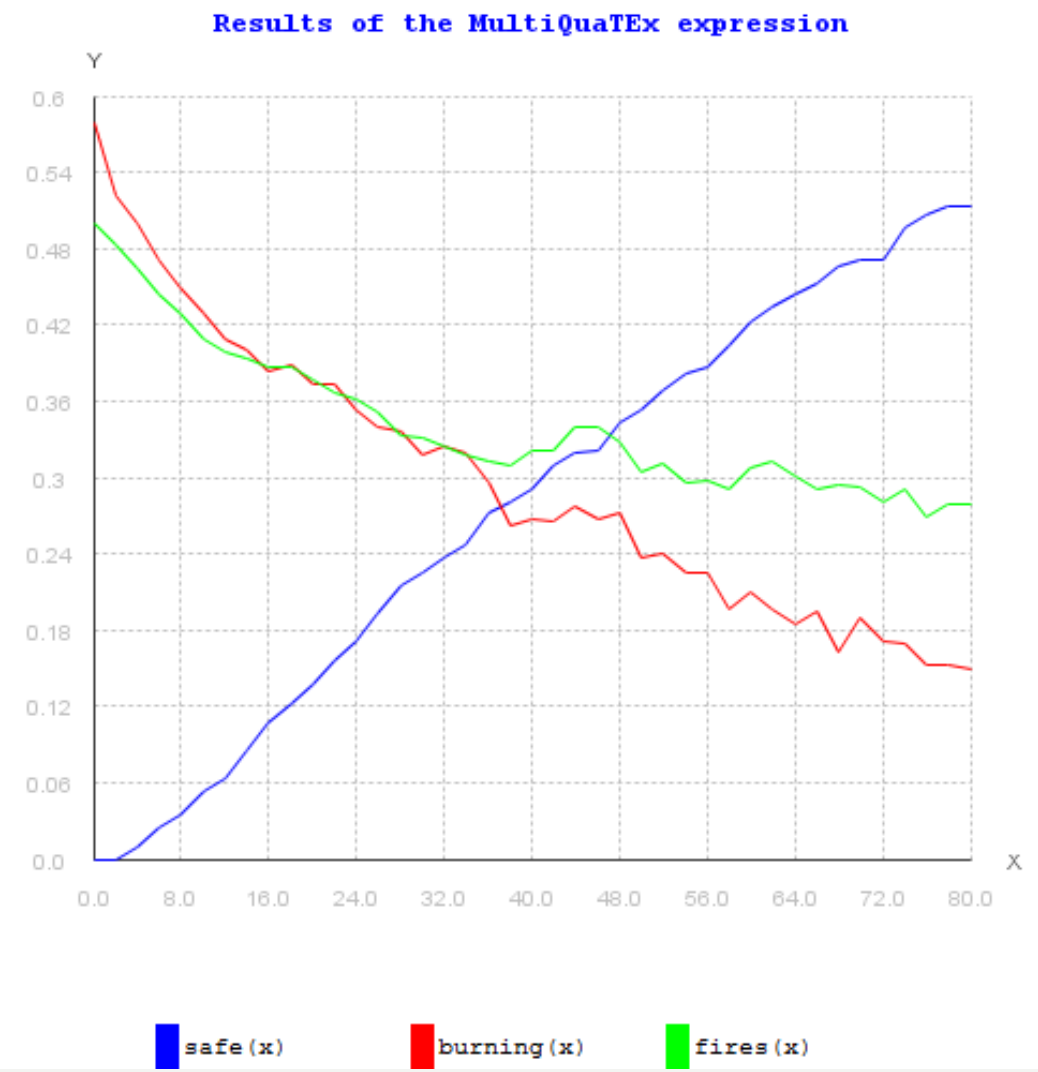
- Victim at ambulance: +100

System synthesizes sensible behaviour

Results in 0.95 confidence interval

- Checked with MultiVeStA

Autonomy



Measured (in %): Victims at ambulance (blue), in a fire (red)
Positions on fire (green)

[Stefano Sebastio and Andrea Vandin. MultiVeStA: statistical model checking for discrete event simulators. ValueTools '13. 2013. 310-315.]

Validation (II)

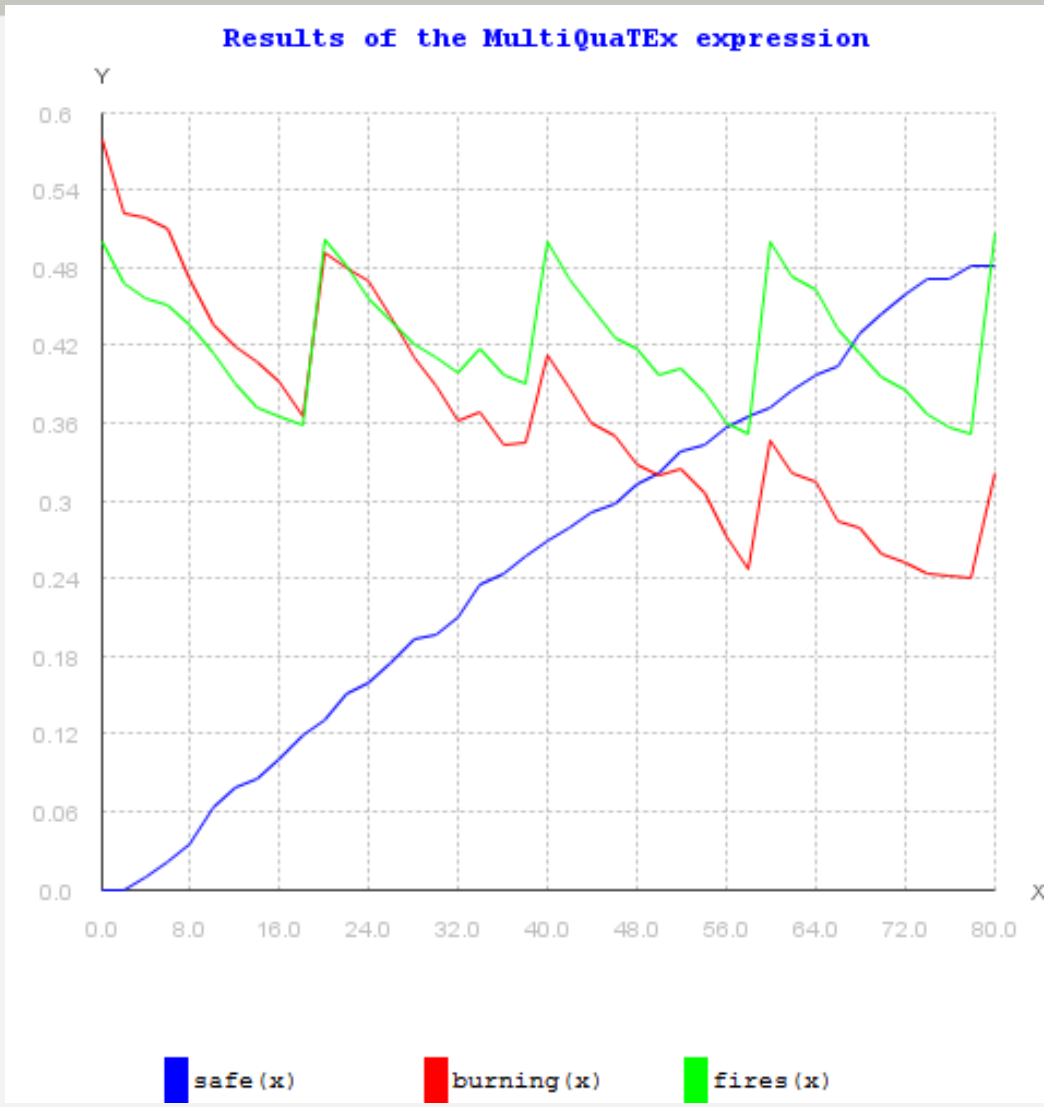
Expose system to **unexpected events**

- At steps 20, 40, 60, 80
- All carried victims are dropped
- New fires break out
- Events NOT simulated by planner
- New situation incorporated by planner

System shows sensible reactions

Results in 0.95 confidence interval

Robustness



Measured (in %): Victims at ambulance (blue), in a fire (red)
Positions on fire (green)

Validation (III)

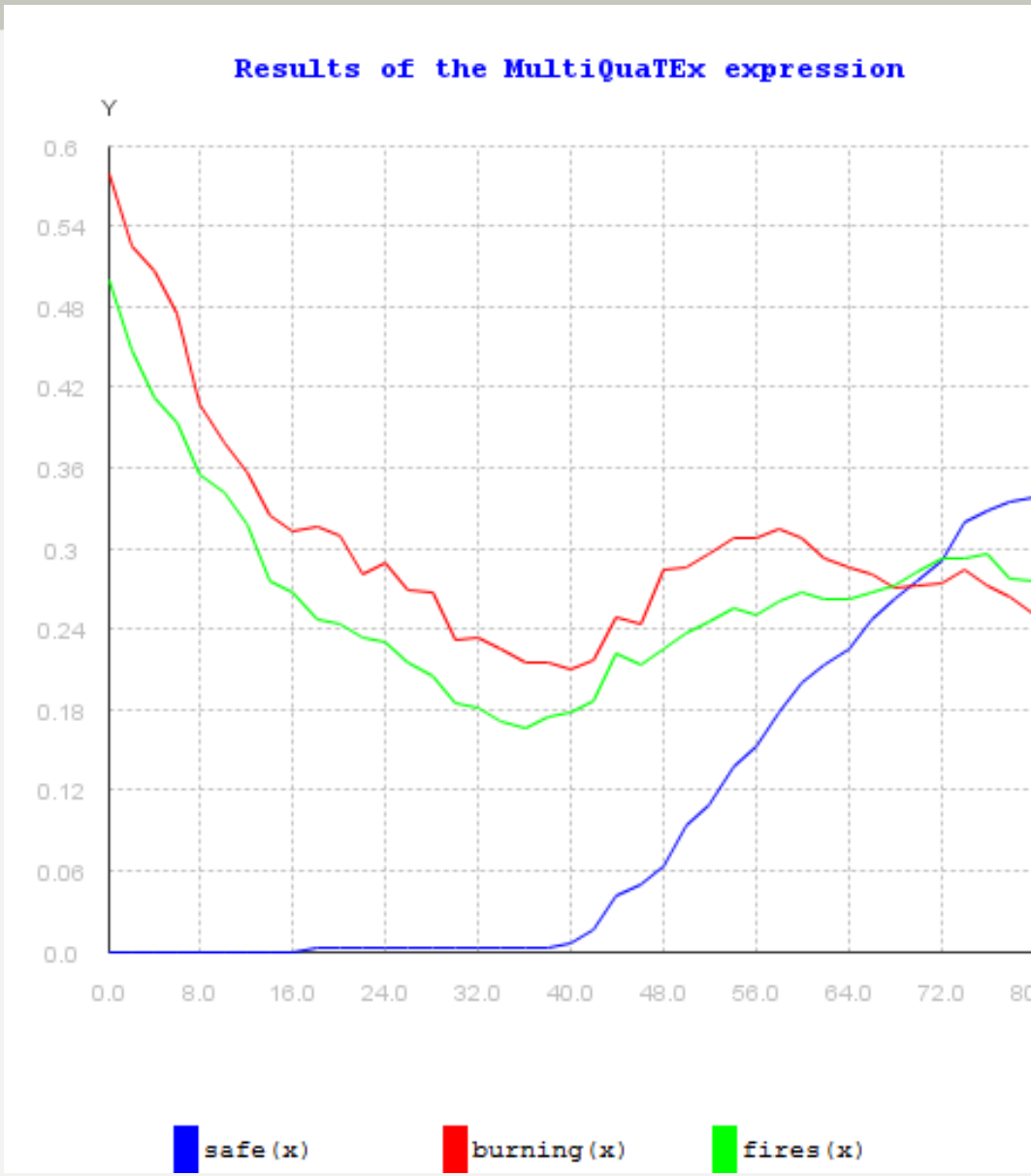
Change system goals while operating

- Change of reward function
  - Steps 0-40: Reward for victims not in a fire
  - Steps 40-80: Reward for victims at ambulance
- Change NOT simulated by planner
- But planner incorporates new situation

System adapts behaviour wrt. goals

Results in 0.95 confidence interval

Flexibility



Measured (in %): Victims at ambulance (blue), in a fire (red)
Positions on fire (green)

4. Safe Learning

PSyCo: Policy SYnthesis with safety COnstraints

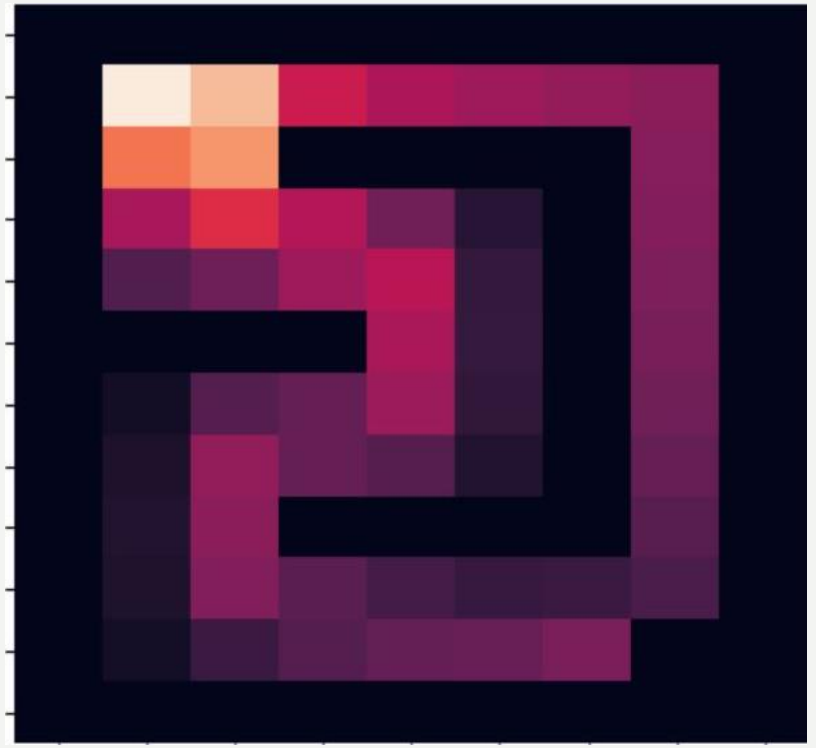
Case study Robot in Grid Requirements

Grid world with two passages

- broad long passage and short narrow passage

Robot agent

- can move into 4 directions but slips with prob 0.05



Requirements

- **Achieve goal**
  - Go to bottom right corner:

    F atBottomCorner
- **Safety („maintain“) goal**
  - Don‘t collide with walls:

    $p(G \neg collision) > p_{sat}$ *iff*

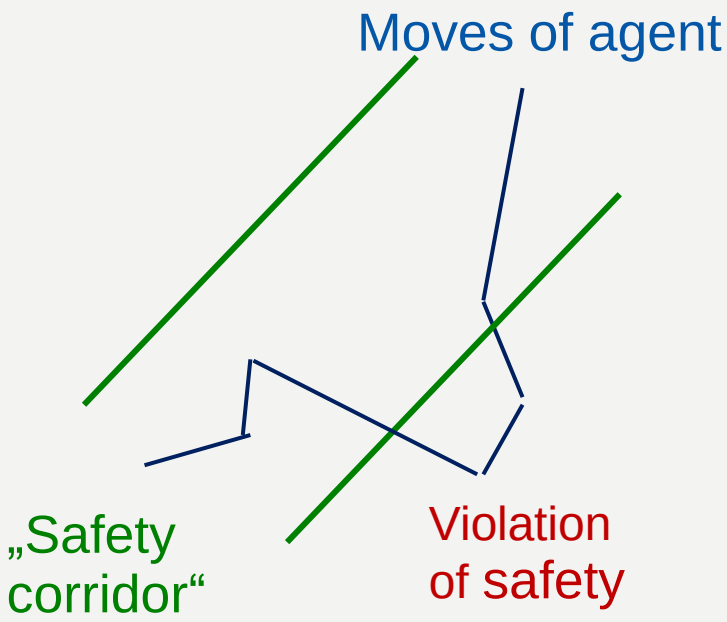
    $\mathbf{p(F\ collision) <= p_{sat}}$

PSyCo: Modelling by Policy Synthesis

Idea: Generate reinforcement learning algorithm from requirements so that the safety requirements are (mostly) respected

- **Achieve goals**
  - **Optimise goal satisfaction** by **Q-learning** as long as safety requirements hold
- **Safety goals**
  - **Control goal satisfaction**

    in case of violation return into the **"safety corridor"**



Modelling: Formally

Reinforcement learning

- **Learn** in parallel **Q-function** of achieve goal and **safety distribution**
- Then the combined policy
  - **maximises expected future reward when being safe**

    and
  - **minimises future expected constraint satisfaction probability when violating constraints**
- **Formally:**

Negated (!) safety property

$$\text{Let } A_{\text{safe}}(s,\pi) = \{a | p(F\phi | s,a,\pi) \leq p_{\text{sat}}\}.$$

$$\pi(s) = \begin{cases} \arg\max_{a \in A_{\text{safe}}(s,\pi)} Q(s,a) & \textbf{if } A_{\text{safe}}(s,\pi) \neq \emptyset \\ \arg\min_a p(F\phi | s,a,\pi) & \textbf{otherwise} \end{cases}$$

Learning safety-aware policy

- **Learn** in parallel
  Q-function of achieve goal and **safety distribution**

- **Q-value iteration*:**

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A(s')} Q(s',a'))$$

where *r* reward, α learning rate, γ discount factor

- **Safety distribution:**

Initialize $\forall s, a, \pi : p(F\phi|s,a,\pi) = 0.$

$$p(F\phi|s,a,\pi) \leftarrow \begin{cases} (1-\alpha)p(F\phi|s,a,\pi) + \alpha 1 & \textbf{if } s' \models \phi \\ (1-\alpha)p(F\phi|s,a,\pi) + \alpha p(F\phi|s',\pi(s'),\pi) & \textbf{otherwise} \end{cases}$$

[**C.Watkins: Learning from Delayed Rewards. Ph.D. thesis,Cambridge University, 1989*]

Robot in Grid World Experiment

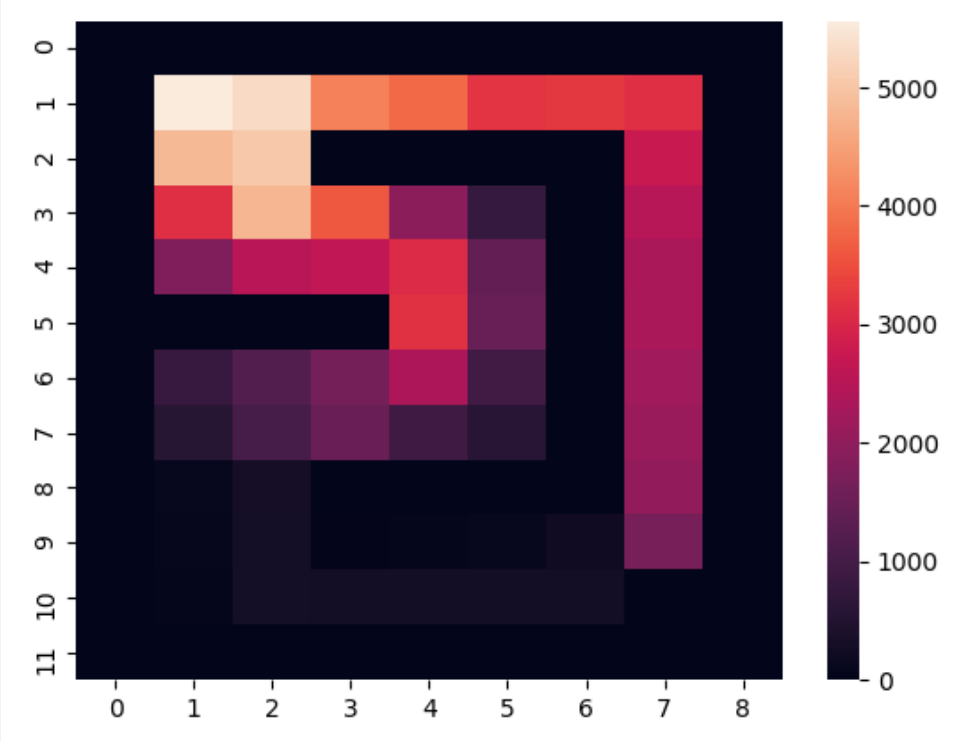
Experiment

- Vary p_{sat} from 0.1 to 0.5
- Learning rate $\alpha = 0.1$, discount factor $\gamma = 0.9$,
- Reward r = 1.0 if robot at bottom right corner

Training

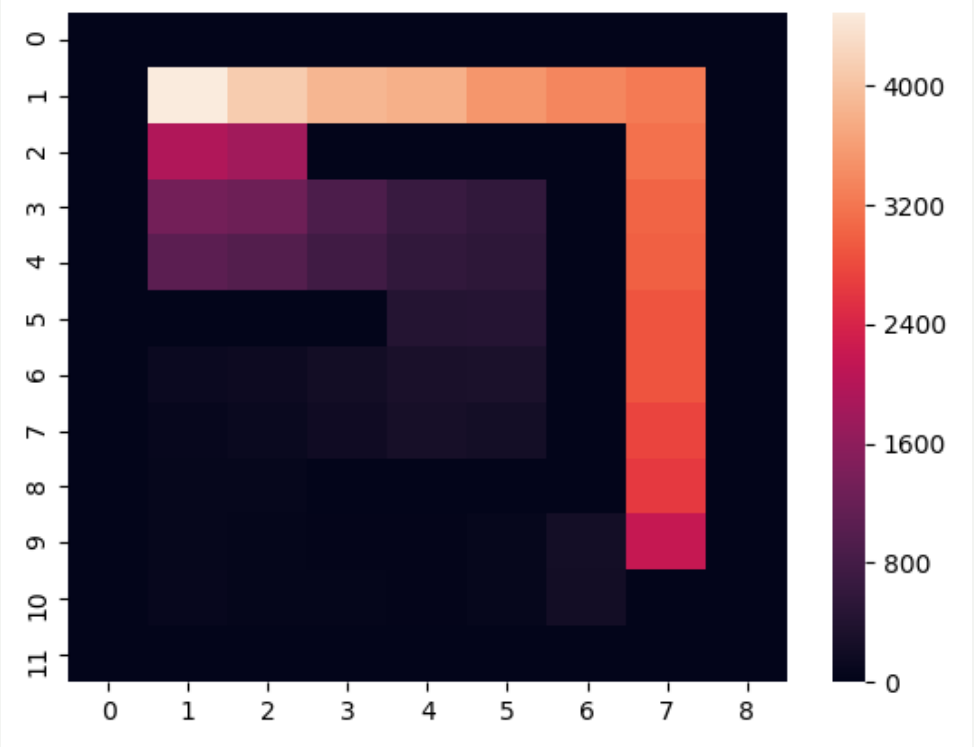
Training: 100.000 episodes (each up to 100 steps)

State visitation count



$p_{sat} = 0.3$

Mostly the long passage is chosen to avoid too many collisions



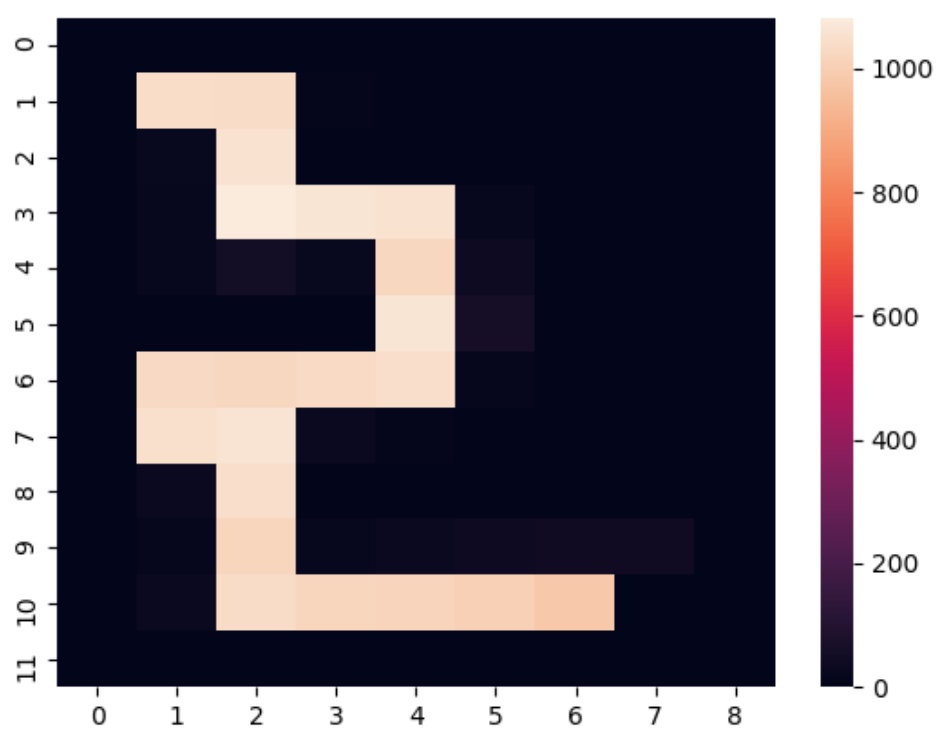
$p_{sat} = 0.5$

Mostly the short passage is chosen due to lax collision requirements

Validation for p_{sat} >= 0.3

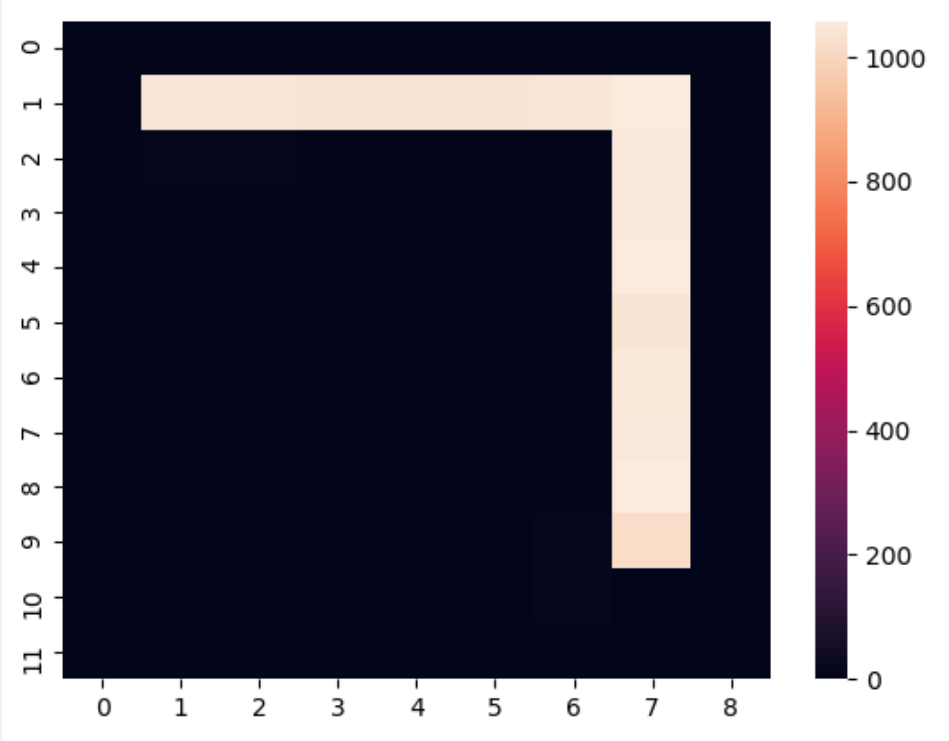
Validation: 1000 episodes using learned policy

State visitation count



p_{sat} = 0.3

Learned policy chooses long passage



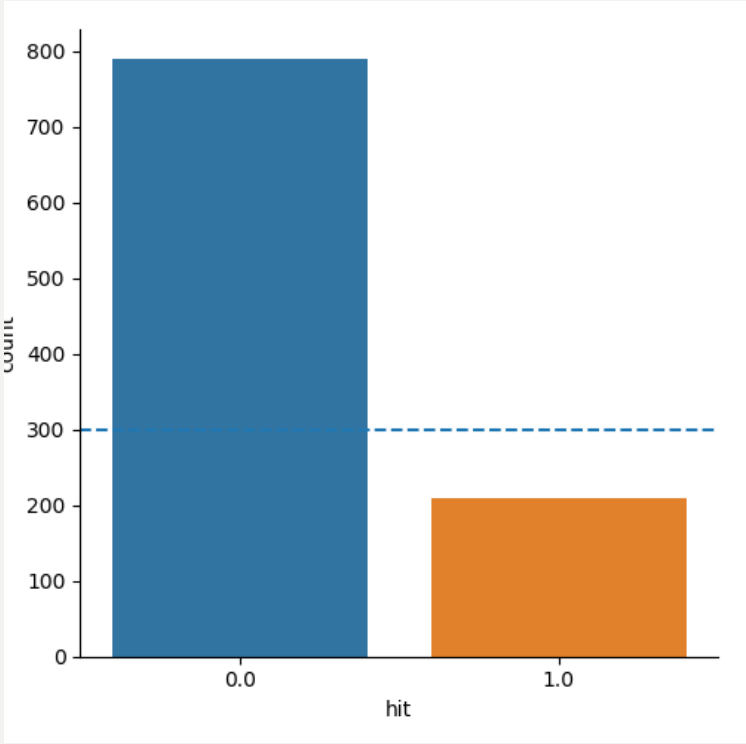
p_{sat} = 0.5

Learned policy chooses short passage

Validation for p_{sat} >= 0.3

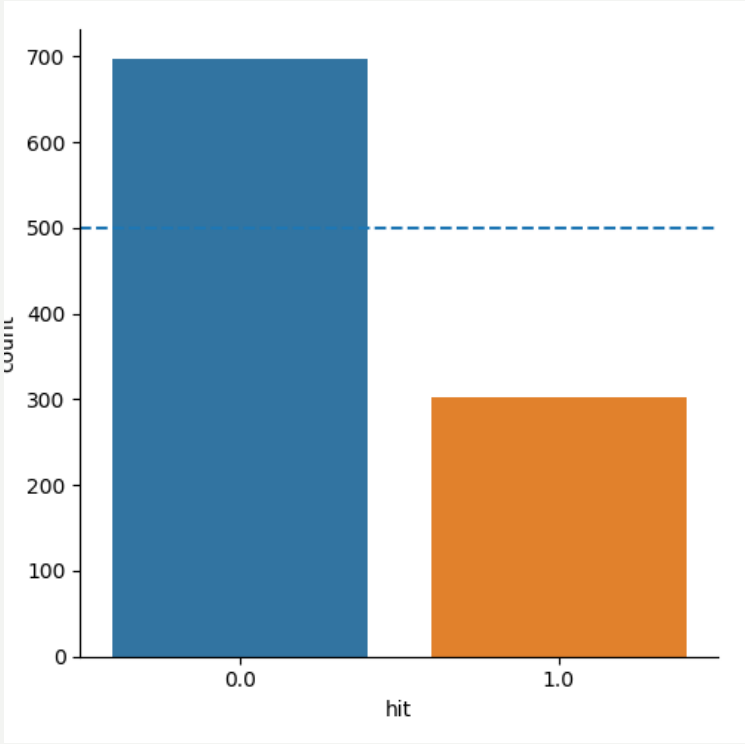
Validation: 1000 episodes with learned policy

 # collision free episodes vs. # episodes with collisions:



p_{sat} = 0.3

80% no collision on long passage



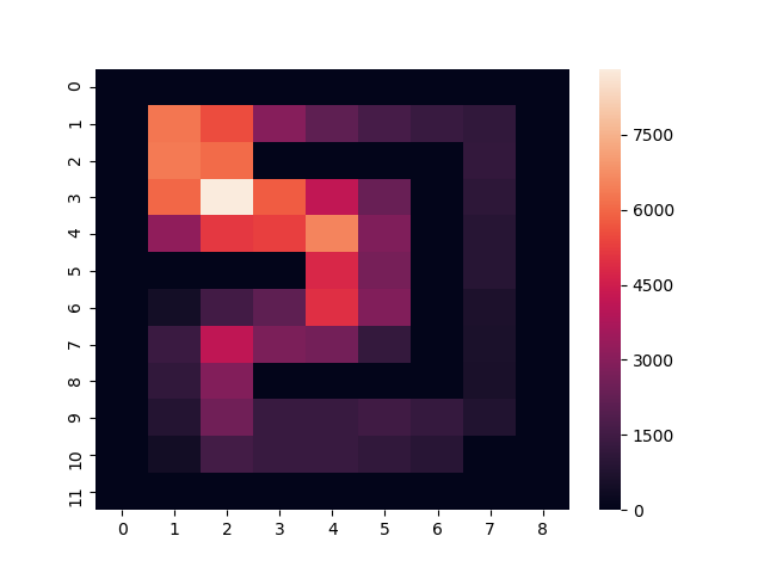
p_{sat} = 0.5

70% no collision on short passage

Validation fails for p_{sat} <= 0.2

Validation: 1000 episodes using learned policy

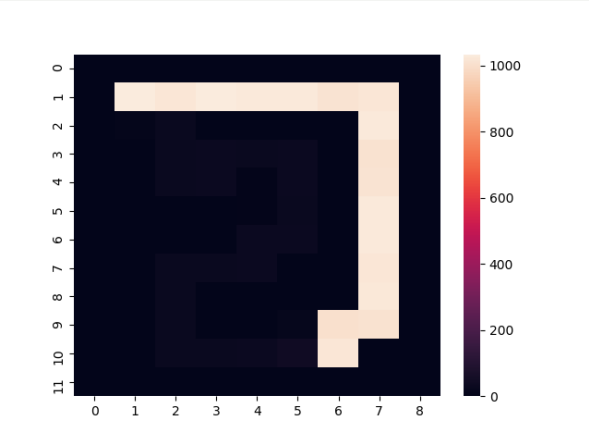
Validation fails for strong collision freeness (p_{sat} = 0.1, 0.2)!

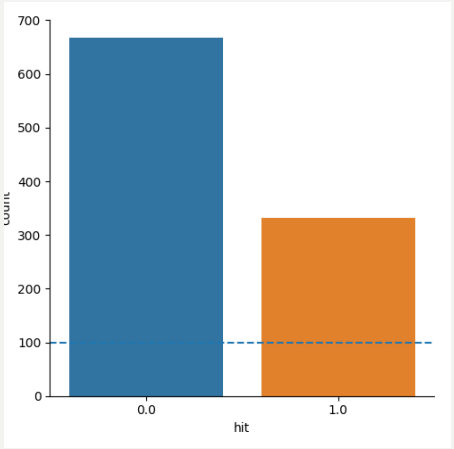


p_{sat} = 0.1

State visitation count

long passage preferred





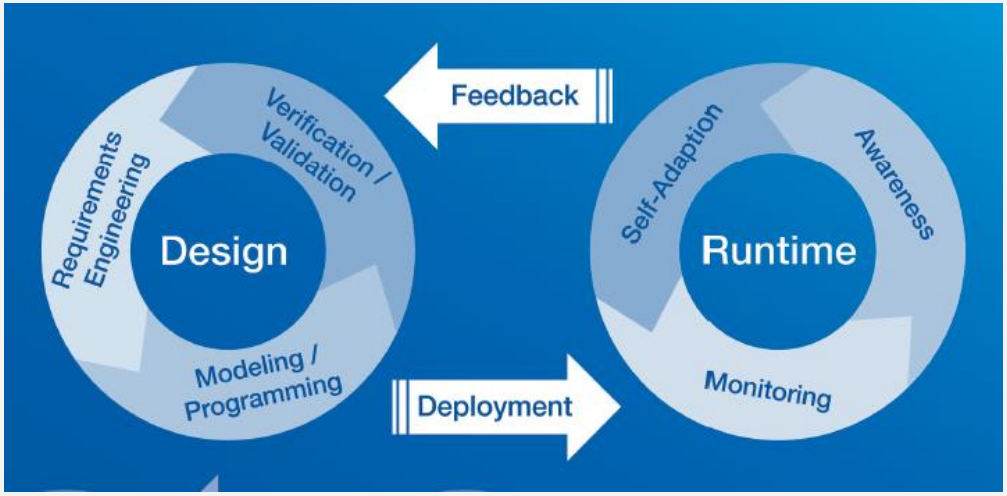
p_{sat} = 0.1

but learned policy follows short passage

and has >30% collisions

➢ **Further design cycle is necessary!**

Concluding Remarks



	„Safe Learning" PSyCo	**Online simulation planning**
Adaptation:	„slow“ at design time	„fast“ at runtime
Execution:	FAST	SLOW

- **Many challenges**
  - **How to engineer complex safe adaptive multi-agent systems?**
  - **In adaptive environments?**