Multi-agentní hledání cest
Jak fungují autonomní sklady

prof. RNDr. Roman Barták, Ph.D.
Matematicko-fyzikální fakulta Univerzity Karlovy
Katedra teoretické informatiky a matematické logiky

Někde v Evropě


TARZAN
42

Někde v Americe

Někde v Africe (či Asii)

Někde na vašem počítači


5100
0050
74/17

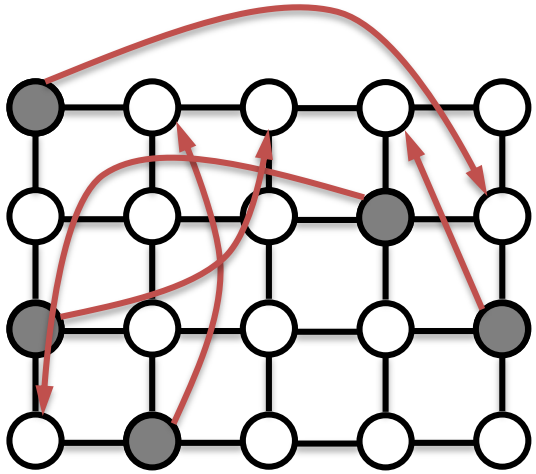
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

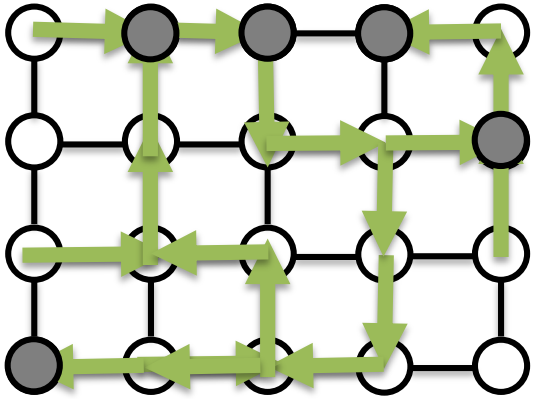
MAPF

Multi-Agent Path Finding

Hledání cest pro mnoho agentů

Co je to **multi-agent path finding** (MAPF)?



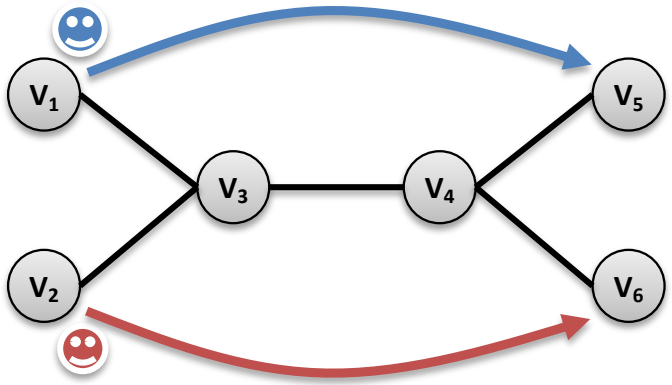


Agent se v jednom kroku může buď posunout do sousedního vrcholu (**move**) nebo zůstat stát (**wait**).

MAPF zadání

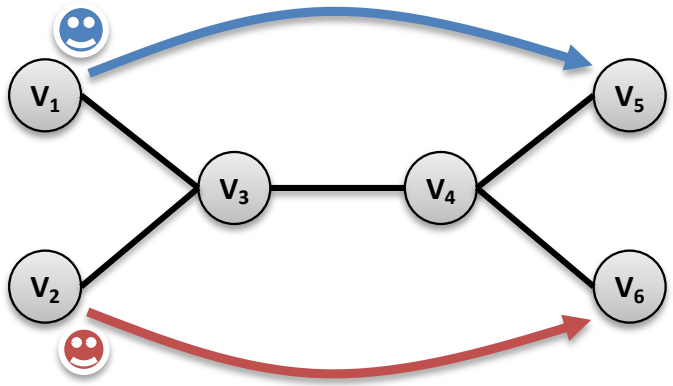
Máme dáno:

- **graf** (orientovaný či neorientovaný)
- množinu **agentů**, kde má každý agent určený počáteční a cílový vrchol v grafu



MAPF řešení

Hledáme **plán**/cestu v grafu pro každého agenta tak, aby se v žádném čase dva agenti nepotkali ve stejném vrcholu grafu (**cesty nekolidují**).

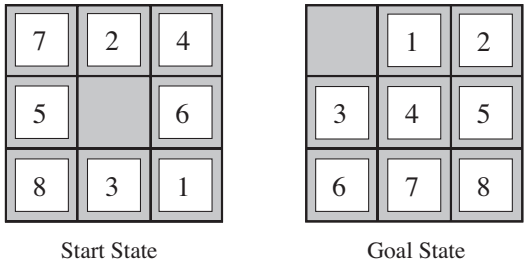


čas	agent 1	agent 2
0	v_1	v_2
1	wait v_1	move v_3
2	move v_3	move v_4
3	move v_4	move v_6
4	move v_5	wait v_6

Řešíme MAPF

Hledáme bezkolizní cesty pro mnoho agentů

Řešení problémů prohledáváním



Správně formulovaný problém se skládá z:

- **počátečního stavu**
- **přechodové funkce**:
  (stav, akce) → stav
  - stavový prostor (graf) je tak definován implicitně (takže lze popsat i obrovské stavové prostory)
- **cílové podmínky**

Řešením problému je **posloupnost akcí**, která vede z počátečního stavu do stavu splňujícího cílovou podmínku.

Prohledávací algoritmy zvažují (prohledávají) různé posloupnosti akcí.

Základní struktura prohledávacího algoritmu:

1. **dej** počáteční stav do seznamu otevřených vrcholů
2. **vyber vrchol** ze seznamu otevřených vrcholů použitím nějaké **prohledávací strategie**
3. **expanduj (zavři) tento vrchol** (přidej všechny následující stavy do seznamu otevřených vrcholů)
4. **opakuj** dokud vybraný vrchol nesplňuje **cílovou podmínku** (nebo je seznam otevřených vrcholů prázdný)

Prohledávací strategie

Trochu značení:

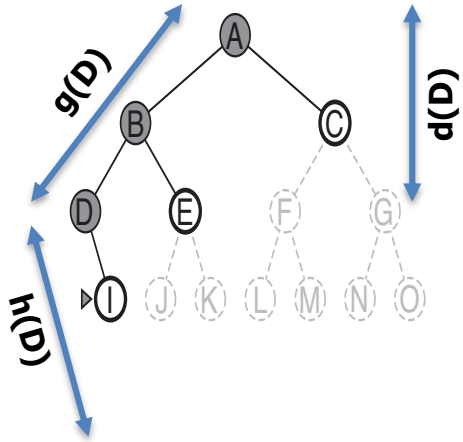
- **f(n)** = ohodnocení vrcholu **n**;
  pro expanzi se vybírá vrchol s nejmenší hodnotou f
- **d(n)** = hloubka vrcholu **n** v prohledávacím stromu
- **g(n)** = délka cesty z počátečního stavu do vrcholu **n**
- **h(n)** = odhad délky cesty z vrcholu **n** do cíle (heuristika)

Neinformované prohledávání:

- prohledávání do šířky: f(n) = d(n)
- prohledávání do hloubky: f(n) = –d(n)
- Dijkstrův algoritmus: f(n) = g(n)

Informované (heuristické) prohledávání:

- hladové prohledávání: f(n) = h(n)
- **algoritmus A***: f(n) = g(n) + h(n)



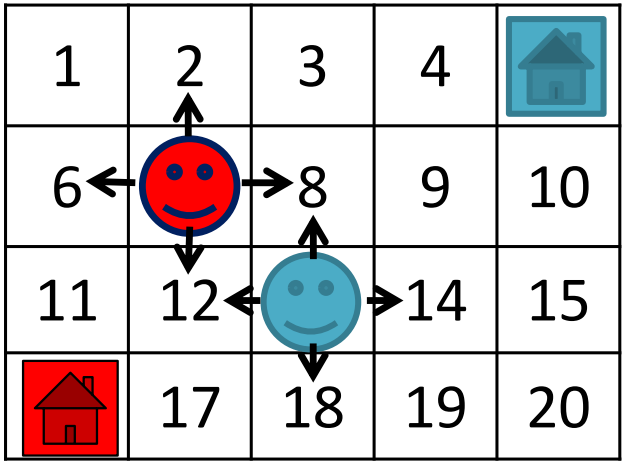
MAPF jako prohledávací problém

stav = umístění všech agentů

akce (přechod) = všichni agenti udělají jeden krok

cílová podmínka = všichni agenti jsou na svých destinacích

Výzva:



počet agentů	počet stavů	větvení
1	20	5
2	20^2	5^2
k	20^k	5^k

5^{20}=

95 367 431 640 625

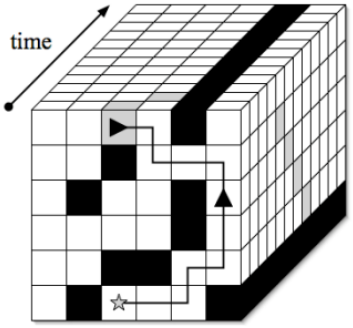
[Silver, AIIDE 2005]

Prioritizované plánování

Myšlenka: plánujeme každého agenta zvlášť

Jak vyřešíme konflikty?

- při hledání cesty uvážíme již nalezené cesty pro jiné agenty

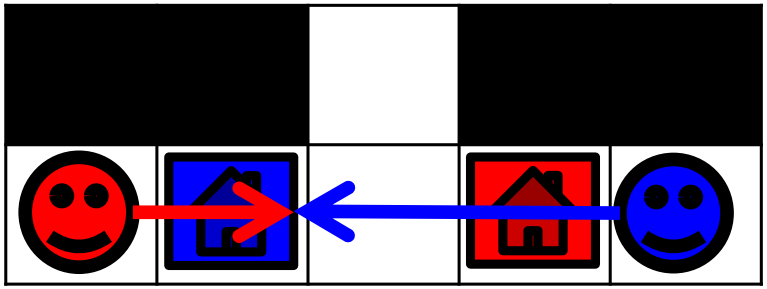


Algoritmus:

- **uspořádej** (nějak) agenty
- **najdi cestu** pro prvního agenta
- **najdi cesty** pro další agenty (jednu po druhé) v časově-prostorovém grafu (zde jsou zaneseny již nalezené cesty a tedy obsazené vrcholy)

Vlastnosti:

- polynomiální ve velikosti grafu a maximálním času
- korektní
- neúplný!!



[Sharon et al, AAAI 2012]

Conflict-based Search

Co když zkusíme řešit konflikty, když nastanou?

Dvě úrovně hledání:

- hledání cest pro jednotlivé agenty
- hledání způsobu vyřešení konfliktů

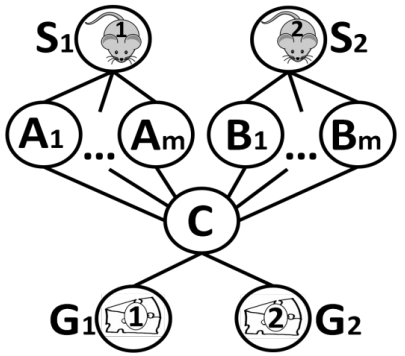
Myšlenka:

- **najdi** cestu pro každého agenta samostatně (nezávisle na ostatních agentech)
- pokud jsou nějací dva agenti někde v konfliktu, **přeplánuj** jednoho z agentů tak, aby se konfliktu vyhnul (agenti si pamatují zakázaná místa)
- **opakuj**, dokud jsou nějaké konflikty

[Sharon et al, AAAI 2012]

CBS: Prohledávací strom

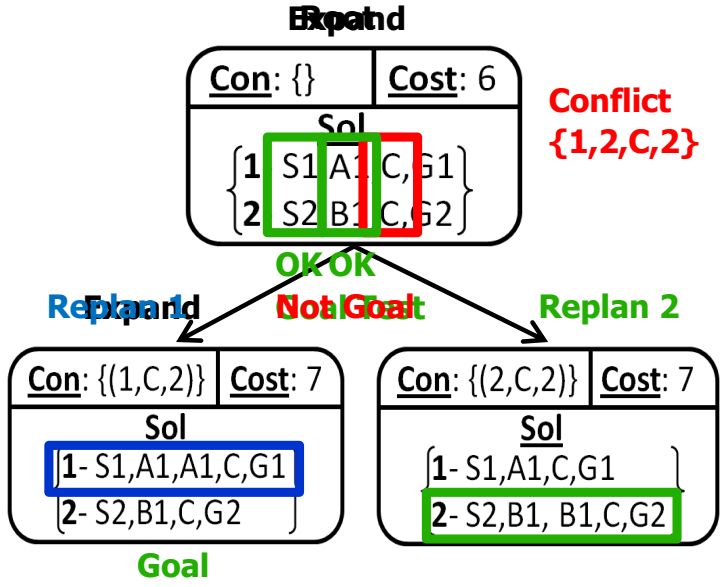
Počáteční stav:
- každý agent má svojí cestu do cíle a žádné zakázané pozice (podmínky)



Další stavy:
- každý agent má cestu do cíle konzistentní s podmínkami
- množina podmínek zakazujících agentům některé pozice

Přechod:
- přidání podmínky řešící nějaký konflikt
- odpovídající přeplánování cest

Cílová podmínka:
- všechny cesty jsou nekolizní



Řešení problémů kompilací

Klasický přístup
k řešení problémů

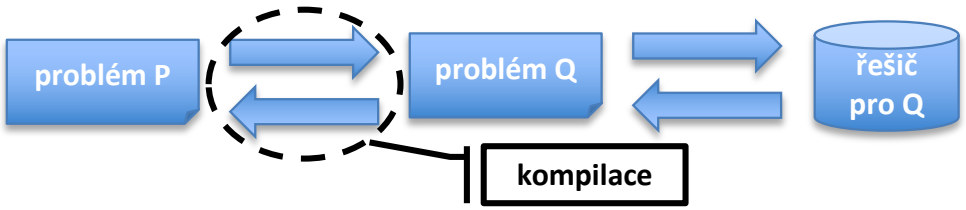
Kompilační (redukční) přístup k řešení problémů

Jaké je myšlenka?

využijme znalosti (techniky) někoho jiného, kdo umí řešit nějaký vlastní problém

Jak ji realizovat?

překladem (kompilací) problému P na jiný problém Q



Proč je to užitečné?

pokud někdo zlepší řešení problému Q dostaneme také lepší řešení problému Q zadarmo

Jiná označení kompilačních přístupů:

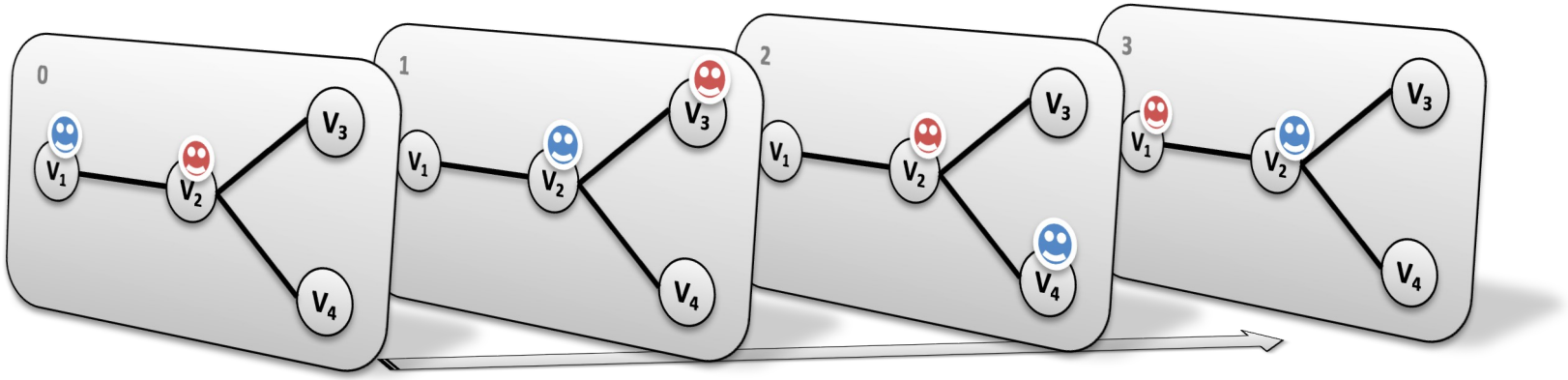
redukční techniky, re-formulační techniky

Hledáme plány

Délku plánu sice předem neznáme, ale můžeme zkusit hledat **plán dané délky** a když neuspějeme, tak délku prostě prodloužíme.

Plán tak můžeme popsat jako posloupnost vrstev/stavů:

- vlastnosti každé vrstvy formálně popíšeme formulí
- a podobně popíšeme přechody mezi vrstvami



Matematický model pro MAPF

Vrstvy popíšeme Booleovskými proměnnými (hodnoty 0 a1)

B_{tav} : v čase *t* je agent *a* ve vrcholu *v*

Podmínky:

- Každý agent je v každém čase právě v jednom vrcholu.

$$\Sigma_{v=1}^{n} B_{tav} = 1 \text{ for } t = 0, \ldots, m, \text{ and } a = 1, \ldots, k.$$

- V žádném vrcholu není více než jeden agent.

$$\Sigma_{a=1}^{k} B_{tav} \leq 1 \text{ for } t = 0, \ldots, m, \text{ and } v = 1, \ldots, n.$$

- Pokud je agent *a* v čase *t* ve vrcholu *v,* potom v čase *t+1* musí být v některém ze sousedů vrcholu (včetně *v*).

$$B_{tav} = 1 \Rightarrow \Sigma_{u \in neibs(v)} (B_{(t+1)au}) \geq 1$$

Jak řešíme MAPF v Picatu?

```
import sat.

path(N,As) =>
    K = len(As),
    lower_upper_bounds(As,LB,UB),
    between(LB,UB,M),
    B = new_array(M+1,K,N),
    B :: 0..1,

    % Initialize the first and last states
    foreach (A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M+1,A,FV] = 1
    end,

    % Each agent occupies exactly one vertex
    foreach (T in 1..M+1, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #= 1
    end,

    % No two agents occupy the same vertex
    foreach (T in 1..M+1, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % Every transition is valid
    foreach (T in 1..M, A in 1..K, V in 1..N)
        neibs(V,Neibs),
        B[T,A,V] #=>
        sum([B[T+1,A,U] : U in Neibs]) #>= 1
    end,

    solve(B),
    output_plan(B).
```

Postupné přidávání kroků

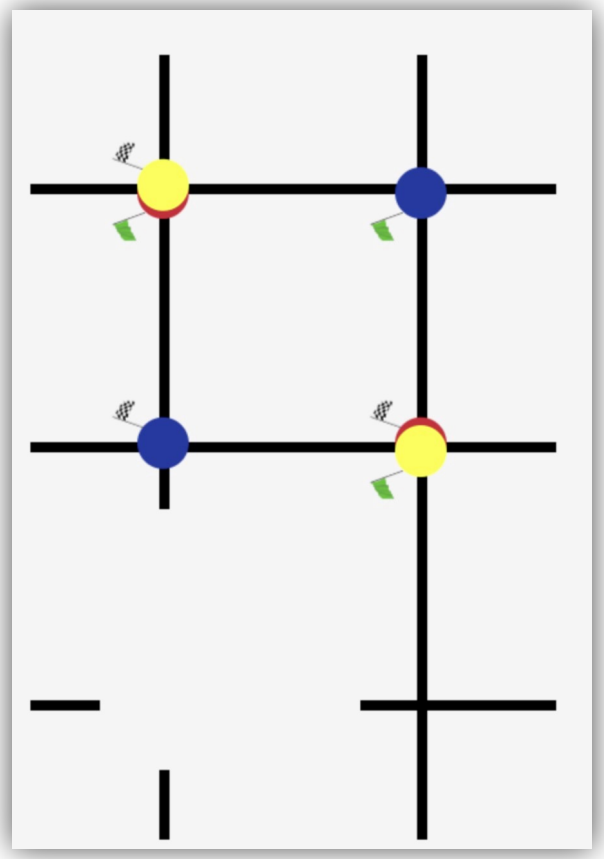
Nastavení startu a cíle každého agenta

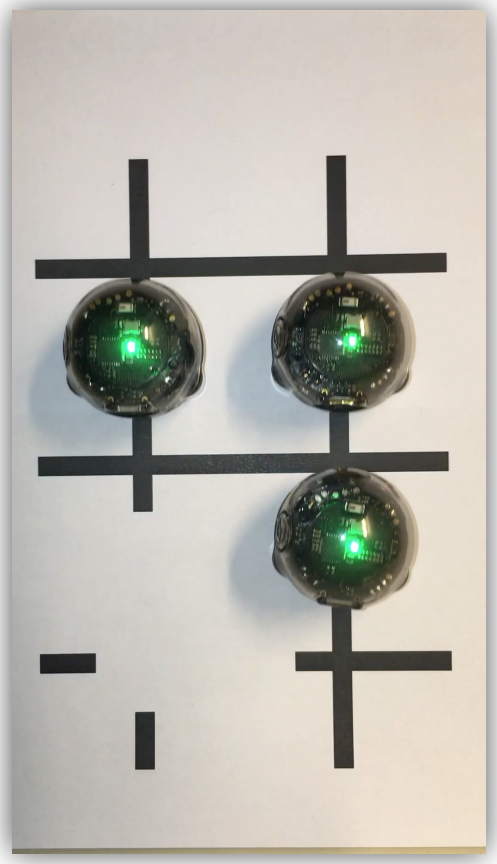
Agent je v každém kroku v jednom vrcholu

Dva agenti se nepotkají ve stejném vrcholu

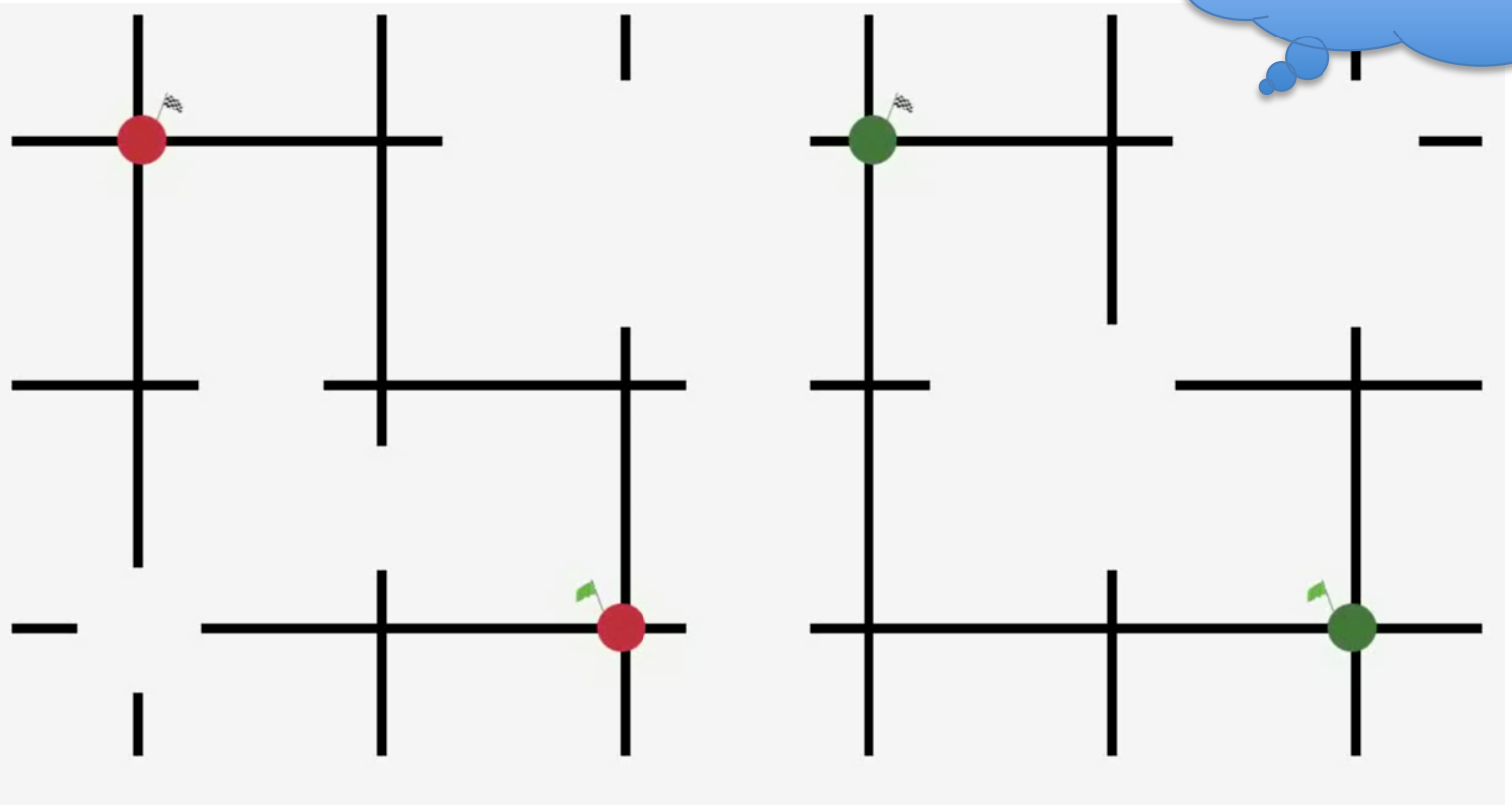
Agent se přesouvá jen do okolních vrcholů

A co na to říkají roboti?

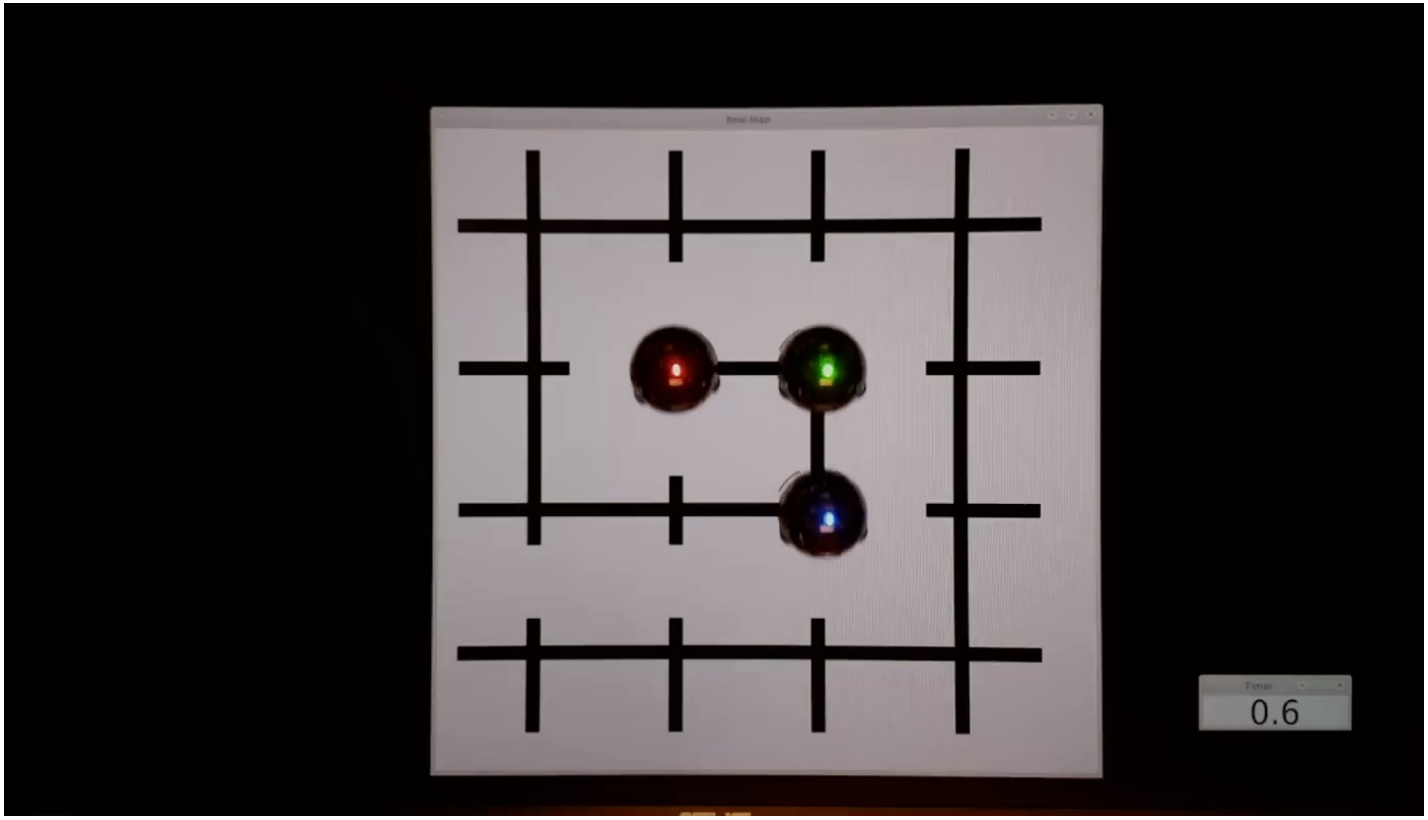




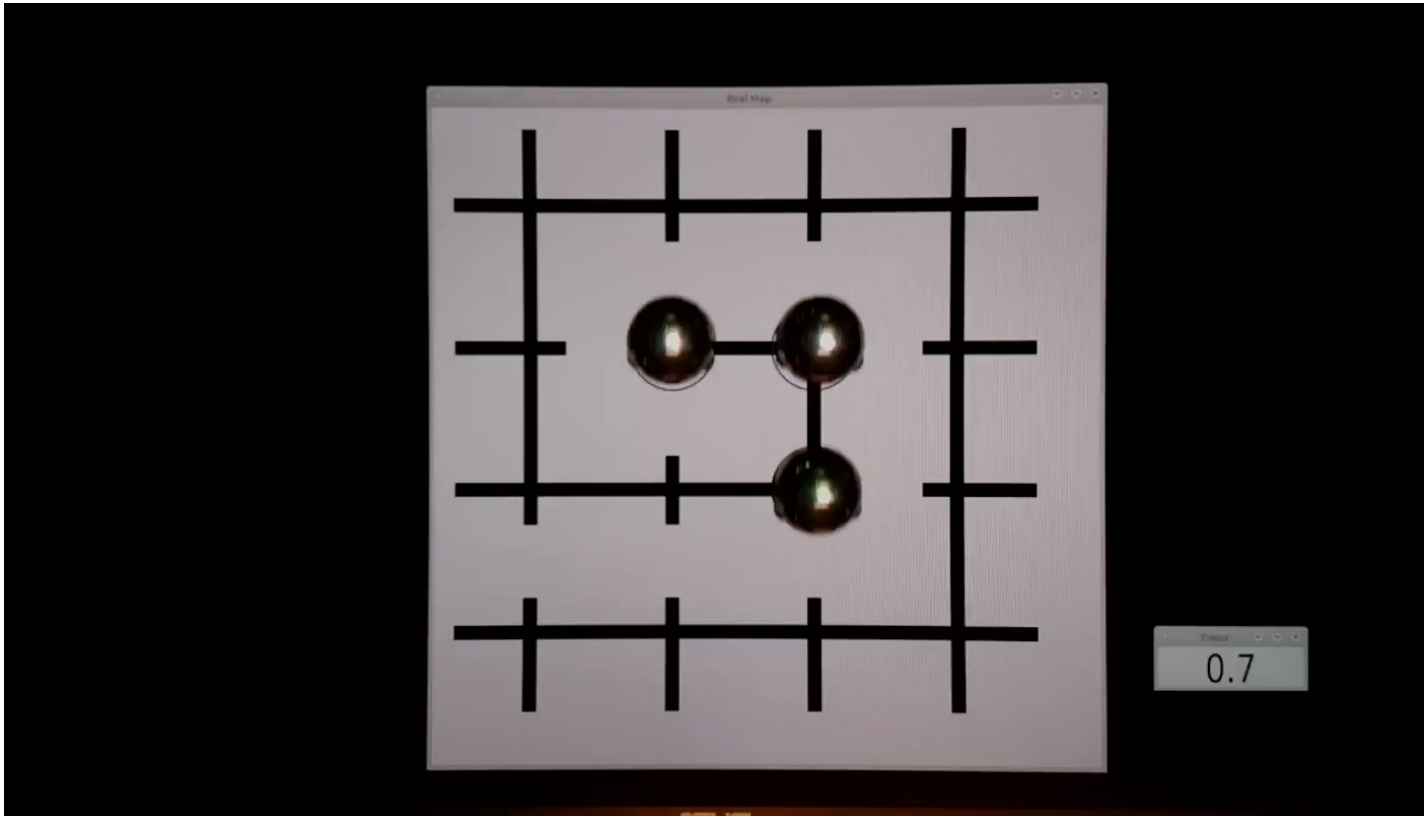
Kde se stala chyba?



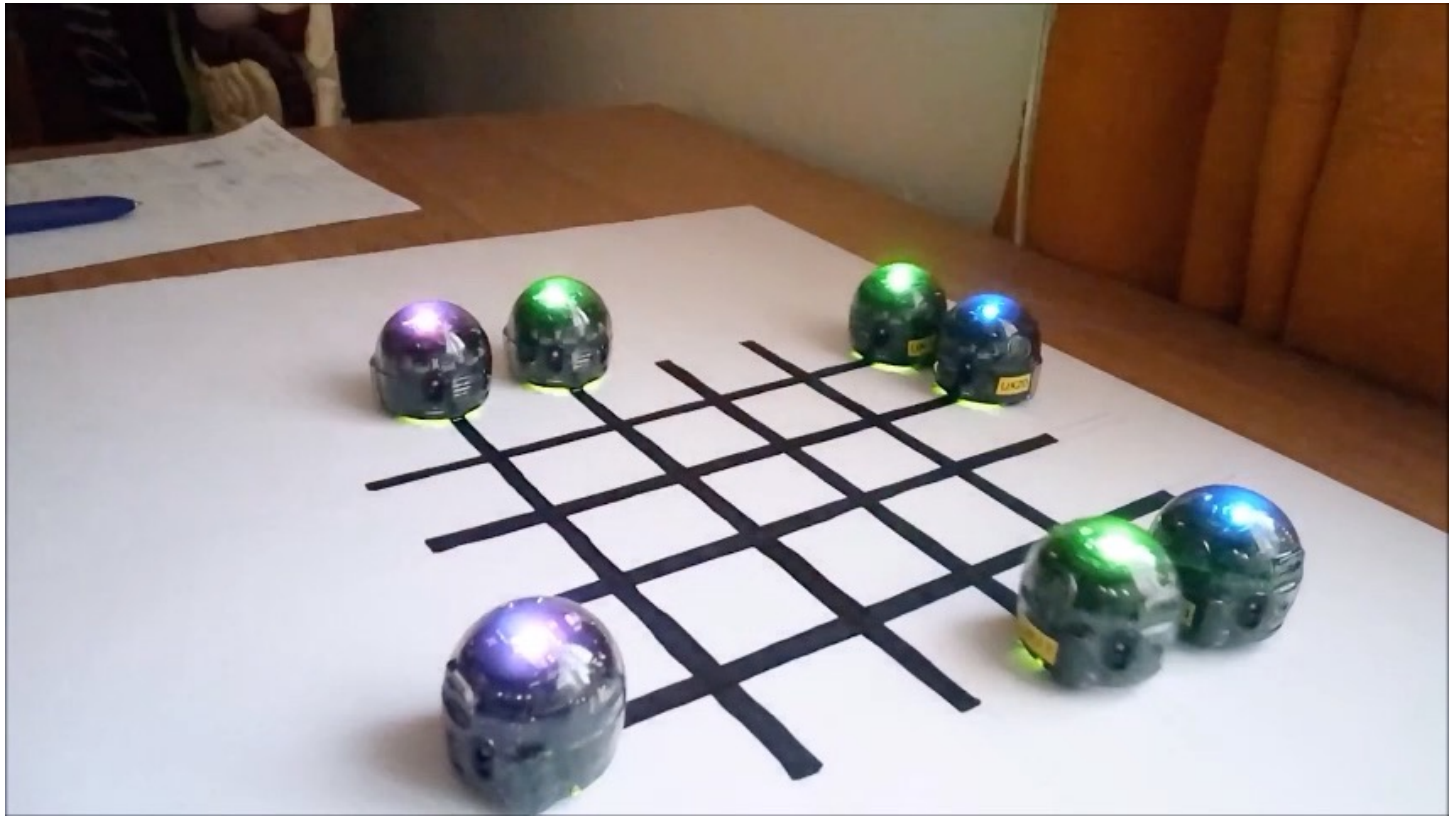
S novým modelem



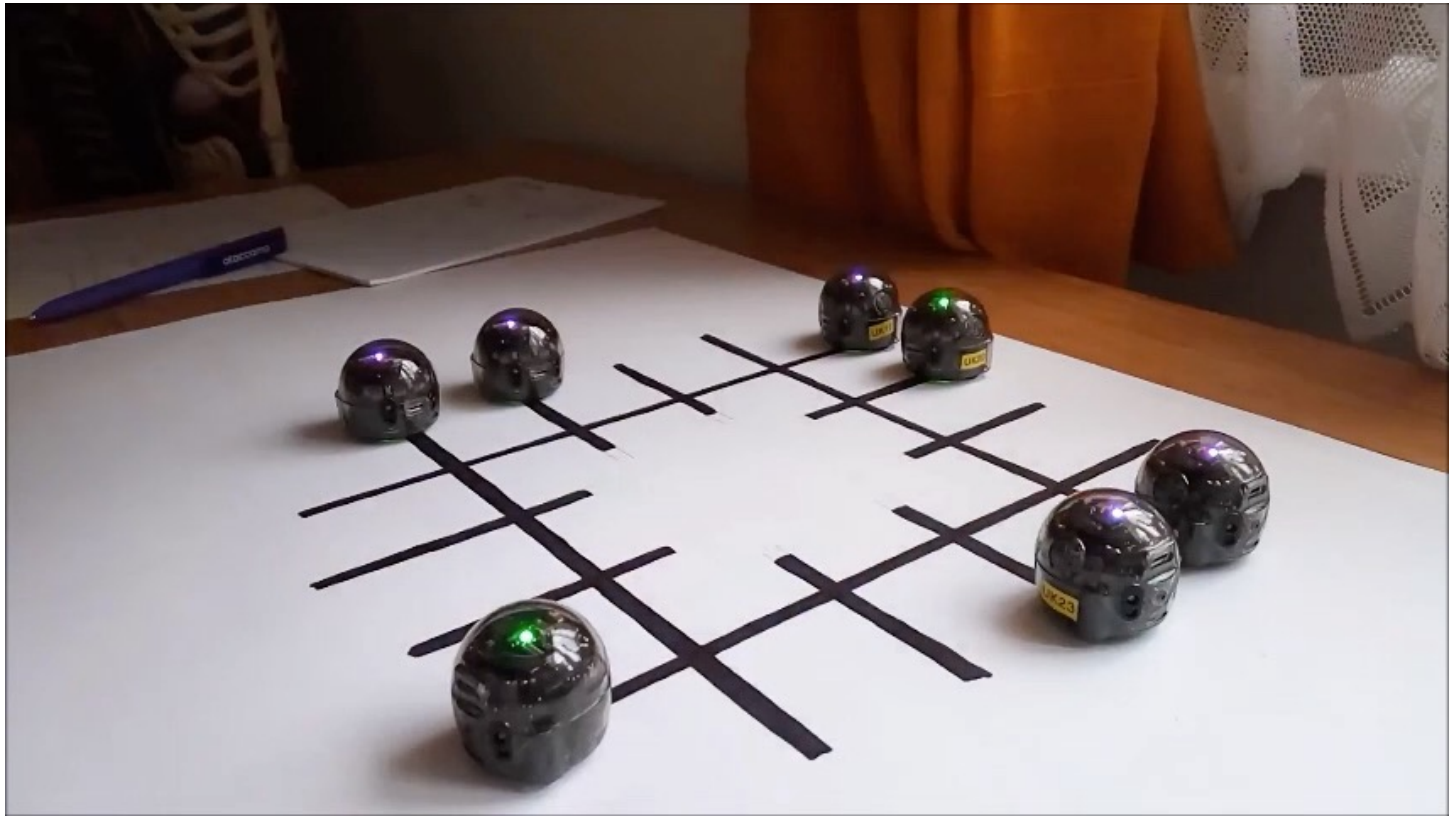
Někdy si roboti dělají, co chtějí



A zvládnete křižovatku?



Klidně i kruhový objezd



Roman Barták
Univerzita Karlova, Matematicko-fyzikální fakulta
bartak@ktiml.mff.cuni.cz

Multi-agentní hledání cest
Jak fungují autonomní sklady

prof. RNDr. Roman Barták, Ph.D.
Matematicko-fyzikální fakulta Univerzity Karlovy
Katedra teoretické informatiky a matematické logiky

tutoriály nejen o MAPF