Inteligentní agenti

Agentem zde rozumíme cokoliv, co ***vnímá*** své prostředí pomocí sensorů a ***působí*** na to prostředí pomocí ***efektorů***.

Člověk-agent z tohoto hlediska má oči, uši a další orgány jako sensory; dále má ruce, nohy, ústa a další části těla jako efektory.

Robot-agent nahrazuje kamerami a infračervenými detektory sensory. Efektory jsou vlastně tvořeny různými motory.

Softwarový agent pro vnímání a působení na prostředí disponuje bitovými řetězci.

Cílem je návrh (umělých) agentů, kteří inteligentně a výkonně jsou schopni působit na své prostředí.

Činnost agentů

Za ***racionálního agenta*** budeme považovat takového, který provádí správné věci.

V prvním přiblížení je *správná věc* taková věc, která umožňuje agentovi nejlepší úspěch. *Jak* a *kdy* se měří *výkonnost agenta?* Pro různé agenty neexistuje stejný způsob měření úspěšnosti. Měření (a vyhodnocení) míry úspěšnosti není vhodné provádět subjektivně (tedy přímo agentem), např. agent to není schopen udělat, apod. Proto se používá *objektivní* míra úspěšnosti, kdy je agent pozorován např. námi z vnějšku v jeho prostředí.

Např. (libovolný) agent, jehož úkolem je vysávat nečistotu z podlahy, může být jednoduše posuzován z hlediska množství odstraněné špíny během osmi hodin. Komplexnější posouzení může vzít do úvahy faktor spotřeby el. proudu, množství produkovaného hluku, apod.

Další otázkou je *kdy* agentovu výkonnost vyhodnocovat. Na začátku vysávání může některý agent vysávat usilovně a za hodinu polevit, jiný může pracovat rovnoměrně celou pracovní dobu, další např. celou dobu své existence. Tomu je nutno přizpůsobit měření.

Je nutno také rozlišovat mezi racionalitou a vševědoucností (zde agent zná *skutečný* výsledek svých akcí a tomu může přizpůsobit svou činnost—to je však v realitě nemožné docílit). Racionalita je zaměřena na *očekávaný* úspěch na základě toho, co je agentem vnímáno (mnoho věcí nelze předvídat a vnímat).

Racionalita závisí v libovolném čase na těchto čtyřech věcech:

- Na měření výkonnosti, které definuje stupeň úspěchu.
- Na všem, co agent vnímal do daného okamžiku (tzv. sekvence vnímání, což je kompletní historie agentova vnímání).
- Na znalostech agenta o jeho prostředí.
- Na akcích, které agent může provádět.

Definice ***ideálního racionálního agenta***: Pro jakoukoliv sekvenci vnímání, ideální racionální agent učiní vše, co se od něj očekává pro *maximalizaci* míry jeho výkonnosti, a to na základě evidence poskytnuté sekvencí vnímání a znalosti, kterou agent disponuje. □

Např. před přechodem křižovatky by se měl robot rozhlédnout doleva a doprava; bez toho jeho (post mortem zkoumaná) sekvence vnímání neodhalí, že vkročil do silnice bez rozhlédnutí a srazilo jej auto—takový robot ovšem není racionální a jeho akce *přejít křižovatku* má nízkou míru výkonnosti.

Důležitou součástí *racionality* (tj. “rozumnosti”) je získávání užitečné informace.

Otázka: lze považovat normální hodinky za jednoduchého racionálního agenta? Proč ano či ne? (Vnímání, úprava času při přechodu do jiné časové zóny...)

Ideální mapování ze sekvence vnímání do akcí

Pokud tedy závisí chování agenta na jeho sekvenci vnímání do určitého okamžiku, lze každého agenta popsat pomocí tabulky akcí, které vykonává jako odezvu na každou (doposud) možnou sekvenci vnímání (pro mnoho agentů by byl seznam velice dlouhý). Tento seznam budeme nazývat *mapováním ze sekvencí vnímání do akcí*. Mapování popisuje agenta a ideální ***mapování*** ideálního agenta. Specifikace akcí, které má agent provádět jako odezvu na dané sekvence vnímání, dává možnost vzniku ***ideálního agenta***.

Není ovšem nutno vytvářet explicitní tabulku s buňkou pro každou možnou sekvenci vnímání—lze často definovat specifikaci bez vyčerpávajícího výčtu. (Např. výpočet druhé mocniny na kalkulačce nepotřebuje znát hodnotu pro každou možnou kombinaci stlačení tlačítek—mapování lze vytvořit např. metodou Newtona.)

Autonomie

Ideální racionální agent by měl obsahovat nějakou "vestavěnou" znalost. Pokud takovou má a je založen výhradně na ní, pak ovšem postrádá *autonomii* (jeho akce pak nezávisí na vnímání).

Autonomie systému je dána vlastní zkušeností/znalostí agenta. Nelze ovšem např. požadovat kompletní autonomii agenta na slovo "jdi", a je nutno se vyhnout jeho nahodilé činnosti. Agent by měl mít schopnost se učit (získávat znalost). Skutečně autonomní inteligentní agent musí být schopen úspěšné činnosti v širokém rozsahu různých prostředí, za předpokladu, že má dost času k adaptaci.

Struktura inteligentního agenta

Návrh programu pro umělého agenta je úkolem moderní umělé inteligence. Programem zde rozumíme funkci, která implementuje agentovo mapování z vnímání na akce. Program obvykle běží na určitém typu počítače, tj. vyžaduje se určitá architektura: *agent = architektura + program.* Před návrhem a implementací programu je nutno dobře znát možná vnímání a akce. Rovněž je nutno znát očekávanou výkonnost a cíle činnosti agenta, a také v jakém prostředí bude agent aktivní. Příklady typů agentů:

Typ agenta	Vnímání	Akce	Cíle	Prostředí
Medicínský diagnostický systém	Symptomy, nálezy, pacientovy odpovědi	Otázky, testy, léčba	Zdravý pacient, minimální náklady	Pacient, nemocnice
Systém analýzy satelitních snímků	Pixely různé intensity, barva	Tisk kategorie záběru	Správná kategorizace	Obrazy z obíhajícího satelitu
Robot sbírající součástky	Pixely různé intensity	Zvednutí součástky a její uložení do přihrádky	Umístění součástek do správných přihrádek	Pás přepravující součástky
Řízení čističky	Teplota, tlak	Otevření, zavření ventilů; přizpůsobení teploty	Maximalizace čistoty, vydatnosti, bezpečnosti	Čistička
Interaktivní učitel angličtiny	Napsaná slova	Tisk cvičení, nápověda, opravy	Maximalizace studentových výsledků v testech	Soubor studentů

Některá reálná prostředí mohou být ve skutečnosti velmi jednoduchá (robot pro třídění součástek). Oproti tomu někteří softwaroví agenti *(softbot = software robot)* existují ve složitých, neomezených doménách (např. softbot navržený pro létání na leteckém simulátoru pro Boening 747, softbot pro prohlížení on-line zpráv a výběr zajímavých pro zákazníky...).

Programy agentů

Jednoduchá základní kostra vnímá prostředí a generuje akce, např.:

```
function Skeleton-Agent(percept) returns action
    static: memory // agentova paměť světa

    memory ← Update-Memory(memory, percept)
    action ← Choose-Best-Action(memory)
    memory ← Update-Memory(memory, action)
    return action
```

Po každém vyvolání funkce je agentova paměť aktualizována vzhledem k novému vjemu (vstup funkce); je vybrána nejlepší akce a skutečnost o výběru akce je rovněž uložena do paměti. Paměť setrvává mezi jednotlivými vyvoláními funklce.

Stačí pouze hledat odpovědi?

K napsání jednoduchého agentova programu stačí např. vytvořit vyhledávací tabulku:

```
function Table-Driven-Agent(percept) returns action
    static:  percepts // na počátku prázdná sekvence
             table    // tabulka indexovaná sekvencemi
                      // vnímání, na počátku stanovená

    append percept to the end of percepts
    action ← Lookup(percepts, table)
    return action
```

Agent je založen na předem stanovené tabulce. Hlídá si sekvenci vnímání a pouze vyhledá nejlepší akci. V paměti se uchovává celková sekvence vnímání, která slouží jako index do tabulky, která obsahuje příslušné akce pro všechny možné sekvence.

Metoda hledání odpovědí pro vjemy může selhat:

- Tabulka potřebná pro jednoduchého agenta, jehož úkolem je pouze hrát šachy, by potřebovala přibližně 35^{100} buněk.
- Pro návrháře by to znamenalo velice dlouhý čas pro vytvoření tabulky.
- Agent nemá žádnou autonomii, protože výpočet (výběr) nejlepší akce je zcela zabudován. Se změnou prostředí nějakým neočekávaným způsobem by mohl agent zcela zhavarovat.
- I kdybychom agenta vybavili nějakým učícím mechanismem, aby měl nějaký stupeň autonomie, trvalo by nesmírně dlouho se naučit správné hodnoty pro všechny buňky tabulky.

Přesto ukázaná funkce `Table-Driven-Agent` implementuje požadované mapování. Je ovšem nutno pochopit, proč usuzující agent—na rozdíl od pouze vyhledávajícího—může být lepší při vyhnutí se uvedeným čtyřem nedostatkům.

Příklad

Předpokládejme agenta—umělého řidiče taxi. Úloha řízení není uzavřená: neexistuje limit pro možné nové kombinace okolností, které mohou nastat.

Typ Agenta	Vjemy	Akce	Cíle	Prostředí
Řidič taxíku	Kamery, rychloměr, GPS, sonar, mikrofon	Točení volantem, akcelerace, brzdění, mluvení k pasažérovi	Bezpečná, rychlá, předpisová a pohodlná jízda, maximální zisk	Silnice, další doprava, chodci, zákazníci

Polohu, účast ostatních objektů na silnici, rychlost... potřebuje taxi znát; to lze získat z vjemů poskytovaných příslušnými zařízeními (kamery atd.). Znalost projíždění zatáčkami vyžaduje akcelerometr. Stav vozidla je dán senzory v motoru, v elektrické výbavě... Akce jsou více-méně tytéž jako u řidiče-člověka: kontrola motoru plynovým pedálem, kontrola řízení volantem a brzdami. Navíc komunikace s dalšími vozidly, syntezátor hlasu pro mluvení na pasažéry.

Míra výkonnosti: dojetí na správné místo; minimalizace spotřeby a optřebení; minimalizace času a ceny cesty; minimalizace narušení dopravních pravidel a rušení dalších řidičů; maximalizace bezpečnosti a pohodlí pasažérů; maximalizace zisku—je zřejmé, že některé požadavky jsou konfliktní, takže je nutno nalézt kompromisy.

Prostředí: může být na lokálních silnicích, na dálnicích? V oblasti se sněhem nebo bez? Jízda vždy vpravo nebo i vlevo (Británie, Japonsko)? Čím větší omezení, tím snadnější návrh.

Nyní je nutno rozhodnout, jak vytvořit reálný program pro mapování vjemů na akce. Různá hlediska řízení mohou vyžadovat různé typy programů—zde budeme uvažovat čtyři typy:

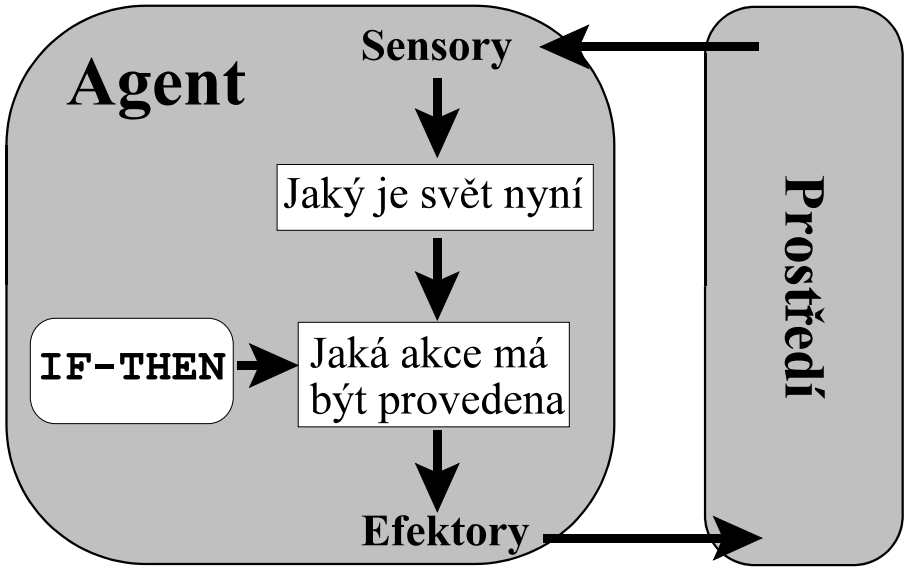
- Jednoduší reflexní agenti.
- Agenti sledující svět.
- Agenti zaměření na cíl.
- Užitkově zaměření agenti.

Jednoduší reflexní agenti

Explicitní vyhledávací tabulky nepřicházejí do úvahy: visuální vstup z jednoduché kamery přichází v množství 50 MB/s (25 obrázků za vteřinu, 1000x1000 pixelů s 8 bity na barvy a 8 bity informace o intensitě). Tabulka by pak musela mít pro jednu hodinu $2^{60x60x50M}$ buněk, což je zjevně přílišný paměťový požadavek.

Je ovšem možné sloučit části tabulky—existují některé obecně se vyskytující vstup/výstupní asociace, např. brzdí-li automobil před námi a jeho brzdová světla se rozsvítí, naše auto by mělo rovněž začít brzdit. To vede k možnosti použití pravidel typu `IF–THEN` (pravidlo typu *podmínka-akce*).

Lidé takovýchto asociací rovněž používají mnoho (některá pravidla se naučí, jiná pocházejí z reflexů). Obrázek ukazuje strukturu jednoduchého reflexního agenta ve schematické formě, a ukazuje, jak pravidla umožňují propojit agentovi vjemy s akcemi:



Jednoduchý reflexní agent hledá pavidlo, jehož podmínka odpovídá současné situaci (určené vjemem) a pak provede akci s tím pravidlem asociovanou:

```
function Simple-Reflex-Agent(percept) returns action
     static:  rules // soubor IF-THEN pravidel

     state ← Interpret-Input (percept)
     rule ← Rule-Match (state, rules)
     action ← Rule-Action[rule]
     return action
```

Agenti sledující svět

Jednoduchý reflexní agent pracuje pouze tehdy, když správné rozhodnutí může být vytvořeno na základě okamžitého vjemu. Pokud auto vpředu je současný model, je možné z jednoduchého obrázku z kamery poznat, zda svítí brzdová světla či ne. Starší modely mohou mít jiná uspořádání a jejich brzdění nemusí být detekováno. Proto musí agent-řidič udržovat určitý druh *vnitřního stavu* pro správný výběr akce. Zde není vnitřní stav příliš rozsáhlý—potřebuje předchozí snímek kamery k určení stavů, kdy se co rozsvítí nebo změní na autě vepředu při jeho brzdění, ukazování změny směru jízdy (blinkr, mechanická ručka...) apod.

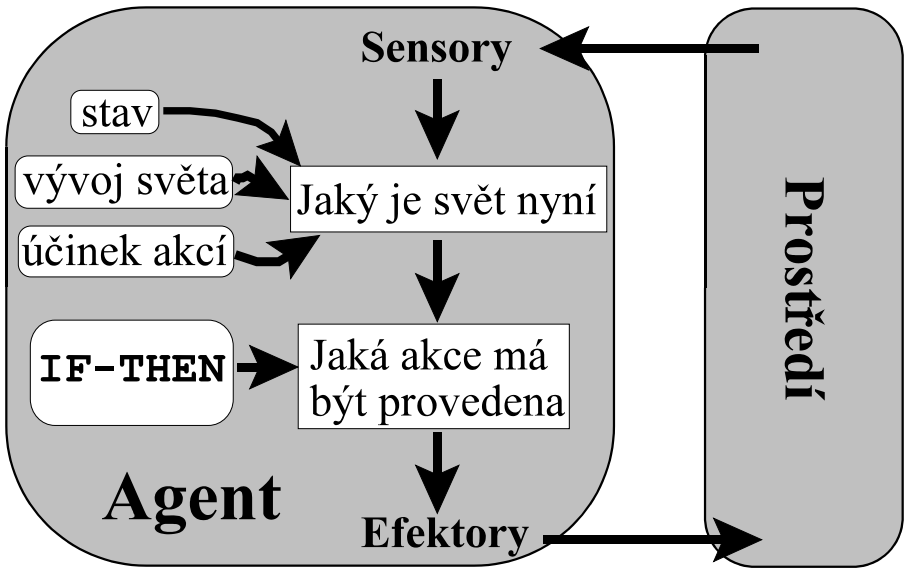
Obdobně je nutno uvážit, že senzory nemusejí vždy poskytnout kompletní informaci o stavu na silnici, v jízdních pruzích apod. V takových případech musí agent udržovat nějakou vnitřní stavovou informaci k odlišení stavů světa, které generují stejný perceptuální vstup, ale jsou výrazně odlišné (tj. jsou zapotřebí výrazně odlišné akce). Např. nejede-li ve vedlejším pruhu auto nebo není-li vidět generuje tentýž vstup, ale ve druhém případě je správnou akcí pokračování v původním pruhu, v prvním případě možnost přejetí do vedlejšího.

Aktualizace vnitřní stavové informace vyžaduje dva druhy znalosti zakódované do agentova programu:

1. Je nutno znát, jak se svět vyvíjí nezávisle na agentovi—např. předjíždějící automobil bude obecně blíže za námi než před okamžikem.

2. Je nutno znát, jak vlastní agentovy akce ovlivňují svět—přejíždí-li agent do pravého pruhu, vzniká po něm přinejmenším dočasně mezera v původním pruhu, nebo že po cca pěti minutách jízdy směrem na sever se nachází cca pět kilometrů severně od místa, kde byl před pěti minutami (věci zdánlivě triviální pro člověka, ale pro stroj-agenta je nutno takto uvažovat).

Schema reflexního agenta s interním stavem:




```
function Reflex-Agent-With-State(percept) returns action
    static:  rules // soubor IF-THEN pravidel
             state // popis stavu současného světa

    state ← Update-State (state, percept)
    rule ← Rule-Match (state, rules)
    action ← Rule-Action[rule]
    state ← Update-State (state, action)
    return action
```

Agenti zaměření na cíl

Znalost okamžitého stavu prostředí nemusí vždy postačovat k rozhodnutí o činnosti. Na křižovatce může taxi jet doleva, doprava, nebo rovně. Správné rozhodnutí zde závisí na tom, jaký je cíl jízdy. Agent tedy potřebuje ještě *infirmaci o cíli* jízdy. Agentův program pak může zkombinovat tuto informaci s informací o výsledcích možných akcí (tatáž informace byla použita k aktualizaci vnitřního stavu u reflexního agenta) k rozhodnutí o výběru správné akce k dosažení cíle.

Někdy je to jednoduché (jedna akce), jindy složité (dlouhá posloupnost zatáčení apod. pro nalezení cesty k dosažení cíle). Umělá inteligence používá metod *vyhledávání* a *plánování* k tomuto účelu.

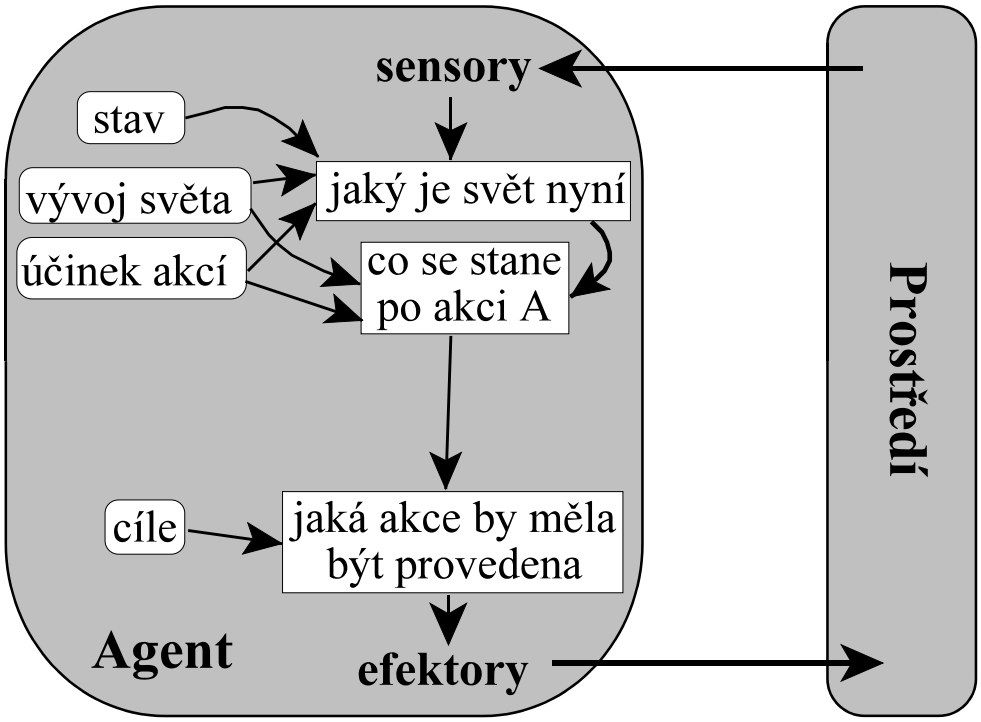
Je dobré si povšimnout, že tento typ rozhodování se fundamentálně liší od používání pravidel typu `IF-THEN` v tom, že zahrnuje úvahy o budoucnosti: “Co se stane když udělám to-a-to?” a “Pomůže mi to nějak?”.

U návrhů reflexních agentů se tato informace explicitně nepoužívá, protože návrhář předem stanovil správné akce pro různé případy. *Reflexní agent* brzdí, když před sebou uvidí svítit brzdová světla. *Cílově zaměřený agent* v principu může uvažovat o tom, že pokud má auto před ním rozsvícená brzdová světla, tak zpomalí. Z toho, jak se obvykle svět vyvíjí, plyne, že jediná akce k zamezení nárazu (dosažení cíle *nesrazit se*) je brzdit.

Cílově zaměřený agent se může zdát méně efektivní, ale je daleko flexibilnější. Začne-li např. pršet, agent může aktualizovat svou znalost o efektivnosti jeho brzd—to může automaticky ovlivnit změny relevantních chování, aby se vygovělo novým podmínkám. U reflexního agenta by bylo nutno přepsat velké množství pravidel typu *podmínka-akce*.

Dále je cílově zaměřený agent také mnohem flexibilnější vzhledem k dosažení různých cílů. Jednoduchou specifikací nového cíle dojezdu lze získat agenta s novým/aktualizovaným chováním. Pravidla reflexního agenta, kdy zatočit a kdy jet přímo, fungují pouze pro jeden cíl dojezdu. Pro dojezd jinam musí být všechna vyměněna.

Cílově zaměřený agent:



Užitkově zaměření agenti

Cíle samy o sobě nepostačují ke generování vysoce kvalitního chování agenta. Např. existuje mnoho sekvencí akcí, které dostanou taxík do jeho určeného cíle—tím je tedy dosaženo *cíle*.

Na druhé straně lze cíl ovšem dosáhnout za různých okolností: ryhcleji, bezpečněji, spolehlivěji, levněji, apod.

Cíle pouze poskytují hrubé rozlišení mezi přínosnými a nepřínosnými stavy.

Obecnější míra výkonnosti by měla poskytnout exaktní srovnání různých stavů světa (nebo *sekvencí stavů*) podle toho, do jaké míry jsou pro agenta přínosné pokud je dosáhne. (Pro přínosnost se používá termínu *agentův užitek*.)

Užitek je tedy funkce, která mapuje určitý stav na reálné číslo, které dále slouží pro určení asociovaného stupně přínosnosti. Kompletní specifikace funkce užitku umožňuje racionální rozhodování ve dvou případech, nastanou-li potíže s cíly:

1. Pokud existují konfliktní cíle, pak pouze některé mohou být dosaženy (např. *rychlost* a *bezpečnost*). Funkce užitku zde stanoví vhodný kompromis.

2. Pokud existuje několik cílů, na jejichž splnění se agent může zaměřit a žádného z nich nelze dosáhnout s jistotou, pak užitečnost poskytuje způsob jak přiřadit váhu pravděpodobnosti úspěchu vzhledem k důležitosti dosažení cílů.

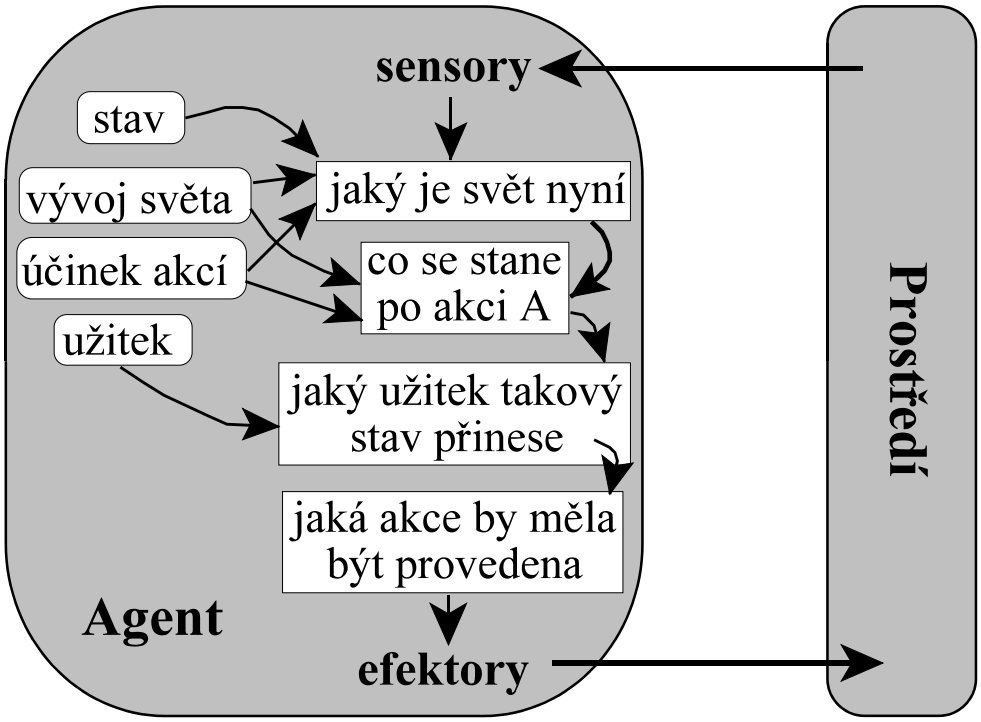
Agent, který vlastní *explicitní* funkci užitku, může tedy vytvářet racionální rozhodnutí, ale musí porovnávat potenciálně dosažitelné užitky, k nimž vedou různé posloupnosti akcí.

Cíle, i když jsou z tohoto hlediska hrubější, umožňují agentovi výběr akce přímo, a to za předpokladu, že akce vyhovují pro dosažení cíle.

Existují také případy, kdy lze převést funkci užitku na soubor cílů takovým způsobem, že cílově zaměřený agent při použití těchto cílů dosáhne stejného rozhodnutí o činnosti jako užitkově zaměřený agent používající danou funkci.

Jiným příkladem užitkově zaměřeného agenta je např. agent hrající nějakou hru (šachy), který musí vytvářet značně “jemná” rozlišování mezi různými pozicemi vznikajícími na šachovnici.

Užitkově zaměřený agent:



Prostředí

Vlastnosti prostředí

Prostředí mají několik charakteristik. Základní odlišnosti:

- **Přístupné a nepřístupné:** zda agentovy sensory umožňují poskytnou kompletní informaci o stavu prostředí. Pokud ano, je prostředí *přístupné*, jinak je *nepřístupné*. Sensory by měly umožnit získat údaje relevantní pro výběr akce. Přístupné prostředí má tu výhodu, že si agent nemusí udržovat vnitřní stavy sledovaného světa.
- **Deterministické a nedeterministické:** je-li následující stav prostředí zcela určen současným stavem a akcemi zvolenými agentem, pak se jedná o *deterministické* prostředí. V principu nemá agent starosti s nejistotou v přístupném a deterministickém prostředí. Zda je prostředí deterministické se obvykle musí určit přímo z *hlediska agenta*.

- **Episodické a neepisodické:** v episodickém prostředí je agentova znalost/zkušenost rozdělena mexi “episody”. Každá episoda se skládá z agentových vnímání následovaných akcemi. Kvalita agentovy akce závisí pouze na dané episodě, protože následné episody na předcházejicích nezávisejí co do jejich vlastní kvality. Episodická prostředí mají výhodu v tom, že agent nemusí myslet dopředu.
- **Statické a dynamické:** pokud se prostředí mění během agentova uvažování, pak je prostředí z jeho hlediska *dynamické*, jinak je *statické*. Statická prostředí mají výhodu v tom, že během svého uvažování nemusí agent sledovat svět, a také se nemusí zabývat časem.
- **Semidynamické:** pokud se prostředí nemění v čase, ale agentova výkonnost na čase záleží, mluvíme o *semidynamickém* prostředí.
- **Diskrétní a kontinuální:** existuje-li omezené množství odlišných a jasně určených vnímání a akcí, pak se jedná o *diskrétní* prostředí (např. šachy jsou diskrétní prostředí—existuje pevné množství možných tahů v každé pozici). Řízení taxíku je *kontinuální*—rychlost a poloha taxíku včetně ostatních vozidel se mění v intervalu spojitých hodnot. Pozn.: Při dostatečně jemné úrovni granularity může dokonce i prostředí taxíku být diskrétní, protože obraz kamery je digitalizován na diskrétní hodnoty pixelů, ale smysluplný program agenta je normálně na abstraktnější úrovni, tj. na úrovni, kde je granularita považována za spojitou záležitost.

Zjevně nejobtížnější případ nastane, když je prostředí nepřístupné, neepisodické a dynamické; obdobně je to i v případě prostředí dynamického. Rovněž se ukazuje, že většina reálných situací je tak složitá, že to, zda je prostředí doopravdy deterministické, je záležitost sporná a pro praktické účely se zachází se situací jako s nedeterministickou.

Následující tabulka ukazuje *typy prostředí* pro některá známější prostředí.

Prostředí	Přístupné	Determi-nistické	Episodické	Statické	Diskrétní
Šachy s hodinami	ano	ano	ne	semi	ano
Šachy bez hodin	ano	ano	ne	ano	ano
Poker	ne	ne	ne	ano	ano
Backgam-mon	ano	ne	ne	ano	ano
Řízení taxi	ne	ne	ne	ne	ne
Medicín-ská diag-nostika	ne	ne	ne	ne	ne
Analýza obrazů	ano	ano	ano	semi	ne
Robot tří-dící sou-částky	ne	ne	ano	ne	ne
Řízení čističky	ne	ne	ne	ne	ne
Interaktiv-ní učitel angličtiny	ne	ne	ne	ne	ano

Hodnoty v tabulce mohou ovšem záviset na tom, jak je *vytvořen pojem* (tzv. konceptualizace) prostředí a agentů.

Např. *poker* je deterministický, pokud si agent může udržovat údaje o sledování pořadí karet v balíčku, jinak bude poker nedeterministický. Obdobně může dojít k tomu, že prostředí je episodické na vyšší úrovni, než se odehrávají agentovy *jednotlivé* akce.

Např. *šachový turnaj* se skládá z posloupnosti her (partií). Každá partie je episoda, protože přínos tahů, provedených agentem v jedné partii, pro celkovou úroveň agentovy výkonnosti není ovlivněn tahy z příští partie. Na druhé straně je nutno uvažovat fakt, že během jedné partie jsou jednotlivé tahy spolu určitým způsobem provázány, takže agent je musí promýšlet v nějakém počtu dopředu.

Programy pro prostředí

Obecný program prostředí ilustruje základní vztah mezi agenty a prostředími:

```
procedure Run-Environment(state, Update-Fn,agents,
                          termination)
     inputs:   state // počáteční stav prostředí
               Update-Fn // funkce pro modifikaci prostředí
               agents // soubor agentů
               termination // predikát k otestování na závěr

     repeat
          for each agent in agents do
               Percept[agent] ← Get-Percept(agent, state)
          end
          for each agent in agents do
               Action[agent] ← Program[agent]Percept([agent])
          end
     state ← Update-Fn (actions, agents, state)
     until termination(state)
```

Uvedený základní program pracuje v principu tak, že *předává* každému agentovi jeho *vjemy* (percepce), od každého agenta *získává* jeho *akci*, a pak *aktualizuje prostředí*.

Simulátory prostředí mohou být založeny na uvedené programové kostře:

Simulátor dostane jako vstup jednoho či více agentů a zařídí, aby opakovaně každý agent obdržel správné vjemy; poté získá akce agentů. Následně simulátor aktualizuje prostředí na základě akcí, případně na základě vlivu dalších dynamických procesů (které nejsou agenty, např. déšť) v prostředí.

Prostředí je tedy definováno *počátečním stavem* a *aktualizační funkcí*.

(Pon.: každý agent pracující v simulovaném prostředí by samozřejmě měl pracovat i v reálném prostředí, které poskytuje tytéž druhy vjemů a přijímá tytéž druhy akcí.)

Procedura `Run-Environment` musí korektním způsobem testovat agenty v prostředí. Pro některé druhy agentů (např. takové, kteří se účastní dialogu v přirozeném jazyce) může postačovat pouhé sledování jejich chování.

Pro získání detailnějších informací o agentově výkonnosti se začlenuje nějaký kód pro měření výkonnosti.

Může to být např. funkce `Run-Eval-Environment`: měří výkonnost každého agenta a vrací seznam výsledných skóre. Proměnná *`score`* obsahuje sledování skóre pro každého agenta:

```
function Run-Eval-Environment(state, Update-Fn, agents,
         termination, Performance-Fn) returns scores
     local variables: scores // vektor daný počtem agentů
                             // na počátku vše 0

     repeat
          for each agent in agents do
               Percept[agent] ← Get-Percept(agent, state)
          end
          for each agent in agents do
               Action[agent] ← Program[agent]Percept([agent])
          end
     state ← Update-Fn (actions, agents, state)
     scores ←  Performance-Fn(scores, agents, state)
     until termination(state)
     return scores
```

Měření výkonnosti obecně může záviset na celkové posloupnosti stavů prostředí generovaných během činnosti programu. Obvykle se však používá jednoduchá akumulace (součet, výpočet průměru, nebo vzetí maxima). Např. uvážíme-li měření výkonnosti agenta vysávajícího podlahy a použijeme-li jako míru celkové množství odstraněné špíny, pak `scores` jednoduše udržuje hodnotu o tom, kolik špíny již bylo doposud vysáto.

`Run-Eval-Environment` vrací míru výkonnosti pro jedno prostředí, definované jedním počátečním stavem a konkrétní aktualizační funkcí.

Agent je však obvykle navržen pro práci ve **třídě prostředí,** čímž se rozumí celý soubor různých prostředí. Např. se může jednat o šachový program, který hraje proti širokému souboru lidských a strojových oponentů. Pokud bychom takový program navrhli proti jedinému protihráči, pak by mohlo dojít k tomu, že se sice využijí konkrétní slabosti daného protivníka, ale nemuseli bychom získat program hrající obecně dobře.

Z toho plyne zžejmý požadavek: pro měření výkonnosti agenta je zapotřebí generátor prostředí, který vybírá konkrétní prostředí (s určitou pravděpodobností) v němž je agent testován. Nás pak zajímá průměrná agentova výkonnost přes celou třídu prostředí.

Je nutno ovšem při realizaci simulátoru prostředí dbát na rozdíl mezi *stavovou proměnnou simulátoru prostředí* a *stavovou proměnnou v agentovi*: agent nesmí vidět stavovou proměnnou simulátoru prostředí, jeho stavová proměnná může pouze nabývat hodnot na základě *agentových* vjemů, bez kompletního přístupu k informacím.

Příklad prostředí a agentů ve světě *vysávání špíny*

Uvedený svět lze popsat takto:

- **Vjemy:** každý vysavačový agent dostává tříprvkový vektor vjemů pro každé kolo vysávání. První element (z dotykového sensoru) dává 1 pokud stroj do něčeho vrazí, jinak 0. Druhý element (fotosensor na spodku stroje) dává 1 pokud je zjištěno smetí, jinak 0. Třetí element dává 1 pokud je agent ve výchozím místě (“doma”), jinak 0.

- **Akce:** pět akcí je k dispozici—jdi vpřed, otoč se doprava o 90°, otoč se doleva o 90°, vysaj smetí, a vypni se.
- **Cíle:** Cílem pro každého agenta je odstranit smetí a vrátit se domů. Exaktnější zadání: 100 bodů za každý vysátý kus smetí, -1 bod za každou akci, -1000 bodů pokud se agent vypne jinde než doma.
- **Prostředí:** prostředí je dáno mřížkou se čtverci. Některé čtverce obsahují překážky (zdi, nábytek...), jiné obsahují volný prostor.Některé volné čtverce obsahují smetí. Každá akce "jdi vpřed" je přesun na další čtverec, pokud tam není překážka—v tom případě agent zůstane stát v původním čtverci, ale dotykový senzor je nadále aktivní. "Vysaj smetí" vždy zlikviduje (odstraní) detekovaný kus smetí. "Vypni se" vždy ukončí simulaci.

Složitost prostředí lze měnit ve třech směrech:

- **Tvar místnosti:** v nejjednodušším případě má místnost $n \times n$ čtverců (pro nějaké pevné n). Obtížnější případ nastane změnou na tvar obdélníka, tvar písmene `L`, nebo nepravidelný tvar, případně série místností spojených chodbami.
- **Nábytek:** umístění nábytku vytváří složitější problém. Z hlediska agenta-vysavače nemusí vjem rozlišovat mezi zdí a kusem nábytku, obojí aktivuje sensor tak, že vydá hodnotu 1.
- **Rozmístění smetí:** nejjednodušší je smetí rozmístit rovnoměrně v místnosti. Realističtější je více smetí umístit v určitých lokacích (např. dlouhá a velmi používaná cesta do další místnosti, nebo kolem židle u jídelního stolu...