Predstavenie hry Sokoban ako plánovací problém

Reprezentácia znalostí a riešenie problému pomocou programu Minisat

17. novembra 2017

Róbert Ruska

1 Úvod

Sokoban (skladník) je logická hra, v ktorej hráč posúva debne v bludisku a snaží sa ich umiestniť na vyznačené pozície. Hra bola vymyslená v roku 1980 pánom Hiroyukim Imabayshim, v rámci súťaže Takarazuckých skladov o vymyslenie motivačnej hry pre zamestnancov. Zverejnená bola v roku 1982. Cieľom projektu je vytvoriť nástroj na vyriešenie ľubovoľného bludiska tak, aby riešenie bolo čo najkratšie.

Hru Sokoban môžeme skúmať z hľadiska výpočtovej zložitosti. Bolo dokázané, že problém riešenia hry má NP-ťažkú výpočtovú zložitosť. Tento problém je zaujímavý aj z hľadiska umelej inteligencie, pretože problém môžeme riešiť automatickým plánovaním, ktorý musí vykonávať robot, ktorý presúva krabice v sklade. Preto tento problém v tomto projekte budeme riešiť plánovaním a pomocou programu, na riešenie NP-ťažkých problémov, pod menom Minisat.[1]

2 Analýza hry

2.1 Popis hry

Hra sa hrá na hracej ploche, ktorá pozostáva z štvorcov, kde každý štvorec reprezentuje buď stenu, alebo podlahu. Niektoré štvorce sú označené ako cieľové miesta na ukladanie debien. Počet cieľových miest je presne toľko ako debien.

Hráč je umiestnený na doske a môže sa pohybovať vodorovne alebo zvisle na prázdne štvorce. Hráč sa tiež môže posunúť na miesto debny, ktorá sa posunie o jedno miesto ďalej v tom istom smere, ako sa posunul hráč. Toto sa, ale môže uskutočniť len vtedy, ak za debnou je voľné miesto, respektíve nie je stena, ani iná debna. Debne sa nesmú tlačiť do iných debien a nesmú byť ťahané. Na jednom políčku hracej plochy môže byť buď hráč, alebo debna. Vo verzii hry, ktorý tento projekt rieši, nie je možné vrátiť krok naspäť.

2.2 Cieľ

Cieľom hry je dostať všetky debny na cieľové miesta. Debny môžu byť na cieľových miestach v hocijakom poradí. Hráč musí vykonať správnu, ideálne najkratšiu postupnosť krokov, aby všetky debny presunul do cieľových miest, pritom musí dávať pozor, aby debny netlačil na také miesta, kde nie sú v cieli a zároveň sa nebudú dať už ďalej posúvať. V tomto prípade už by bola hra neriešiteľná.

3 Formalizácia

Na formalizáciu je potrebné nejakým spôsobom zapísať hraciu plochu. Zapisovanie faktov a akcií v jednotlivých krokoch (stavoch) bude tiež potrebné. Akcie a fakty budeme nazývať predikáty, pretože ich zapisovanie sa veľmi podobá zapisovaniu v predikátovej logike. Jednotlivé políčka budeme označovať ako kombináciu čísiel, ktoré predstavujú riadok a stĺpec. Akcie a fakty musia byť v CNF.

3.1 Hracia plocha

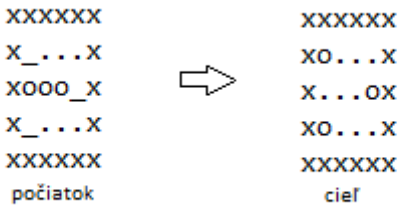
Hracia plocha je daná ako textový súbor, kde jednotlivé znaky reprezentujú štvorce hracej plochy. Hracia plocha musí byť ohraničená stenami.

Jednotlivé znaky a ich reprezentácia na mape:

- 'x' - stena
- '.' - podlaha
- '_' - cieľ
- 'o' - debna

Súbor na konci obsahuje dve čísla, ktoré reprezentujú počiatočnú pozíciu hráča. Prvé číslo reprezentuje riadok a druhé stĺpec v hracej ploche.

Na dole uvedenom obrázku vidíme počiatočný stav(forma zapisovanie v súbore) a cieľový stav:



3.2 Použité predikáty

Na zapisovanie jednotlivých faktov a akcií používame rôzne predikáty:

- $at(p, 1, 2, 6)$ - hráč na riadku 1 v stĺpci 2 v stave 6
- $at(b, 1, 3, 6)$ - debna na riadku 1 v stĺpci 3 v stave 6

- $move(p, 1, 2, 1, 3, 7)$ - presun hráča z riadku 1 a zo stĺpca 2 do riadku 1 a do stĺpca 3 v stave 7
- $move(p, 1, 3, 1, 4, 7)$ - presun debnu z riadku 1 a zo stĺpca 3 do riadku 1 a do stĺpca 4 v stave 7

Vyjadrovanie toho, keď hráč alebo debna nie je na danom políčku, alebo keď sa nejaký posun neuskutočnil, použijeme negáciu. Negáciu vyjadrujeme v tvare: znak '$-$' + predikát, napr: $-at(b, 1, 4, 6)$.

3.3 Počiatočný stav

V počiatočnom stave určíme, na ktorých políčkach sa používateľ a debny nachádzajú alebo nenachádzajú. Zo vstupného súboru vieme zistiť všetky potrebné informácie. Pre jednotlivé políčka nastavujeme teda štyri predikáty:

- $at(p, i, j, 0)$ - pre políčko, kde sa používateľ nachádza
- $-at(p, i, j, 0)$ - pre všetky políčka, kde sa používateľ nenachádza
- $at(b, i, j, 0)$ - pre všetky políčka, kde sa nachádzajú debny
- $-at(b, i, j, 0)$ - pre všetky políčka, kde sa nenachádzajú debny

3.4 Cieľový stav

V cieľovom stave určíme, na ktorých políčkach sa používateľ a debny nachádzajú alebo nenachádzajú. Zo vstupného súboru vieme zistiť, kde sa debny majú nachádzať na konci hry. Koncový stav prestavuje posledný krok hráča na hracej ploche. Pozícia hráča je nerelevantná, takže ju nemusíme zapisovať.
Pre jednotlivé políčka nastavujeme teda štyri predikáty(predpokladajme, že koncový stav je k-ty):

- $at(b, i, j, k)$ - pre všetky políčka, kde sa nachádzajú debny
- $-at(b, i, j, k)$ - pre všetky políčka, kde sa nenachádzajú debny

3.5 Akcie

Budeme mať dve akcie, presun hráča a presun debny. Tieto akcie sa môžu vykonávať v každom stave zo všetkých políčok, pre každý smer, kam sa hráč alebo debna vie presunúť(t.j. nie je stena). Pre každý z presunov v danom stave a smere musíme zapísať všetky predpoklady a všetky efekty, ktoré musia platiť pre daný posun. Použijeme implikáciu a konjunkciu.

Predpokladajme, že robíme k-ty krok:

Presun hráča:

$move(p, i, j, i+1, j, k) \implies at(p, i, j, k-1) \wedge -at(p, i+1, j, k-1) \wedge -at(p, i, j, k) \wedge at(p, i+1, j, k)$

Presun debny:

$move(b, i, j, i+1, j, k) \implies at(b, i, j, k-1) \wedge -at(b, i+1, j, k-1) \wedge -at(b, i, j, k) \wedge at(b, i+1, j, k) \wedge move(p, i-1, j, i, k)$

Pri presune debien musí platiť okrem základných znalostí, že sa musí presunúť aj hráč v tom istom smere, tesne za debnou.

Napr. slovne:

Presun hráča z pozície i,j na i+1,j v nasledujúcom stave, keď platí, že bol v predošlom stave v i,j, nebol v predošlom stave v i+1,j a v nasledujúcom stave bude na i+1,j a nebude na i,j.

3.6 Exkluzivita

Pri exkluzivite určujeme exkluzivitu predikátov, teda že v istom stave sa môže vykonať iba istý počet. Hráč môže byť v jednom stave iba na jednom mieste. Hráč a debna nesmú byť na tom istom mieste. Hráč sa môže posunúť iba jedným smerom naraz v jednom stave. Debna sa môže posunúť v jednom stave iba jedným smerom.

Napríklad:
Hráč môže byť v jednom stave iba na jednom mieste:

$$-at(p, 3, 2, k) \vee -at(p, 2, 2, k)$$
$$-at(p, 3, 2, k) \vee -at(p, 2, 3, k)$$

Hráč a debna nesmú byť na tom istom mieste:

$$-at(b, 2, 2, k) \vee -at(p, 2, 2, k)$$
$$-at(b, 2, 3, k) \vee -at(p, 2, 3, k)$$

Hráč sa môže posunúť iba jedným smerom naraz v jednom stave:

$$-move(p, 1, 1, 2, 1, k) \vee -move(p, 1, 1, 1, 2, k)$$
$$-move(p, 1, 1, 1, 2, k) \vee -move(p, 1, 1, 2, 1, k)$$

Debna sa môže posunúť v jednom stave iba jedným smerom:

$$-move(b, 2, 2, 2, 3, k) \vee -move(b, 2, 2, 2, 1, k)$$
$$-move(b, 2, 3, 1, 3, k) \vee -move(b, 2, 3, 3, 3, k)$$

3.7 Frame Problem

Riešenie klasických frame problémov, t.j keď nejaký fakt platil a už neplatí, tak sa museli vykonať tieto posuny.

Napríklad:

Keď sa zmenila pozícia hráča, tak sa mohol vykonať jeden z týchto presunov hráča pre každý stav:

$-at(p,3,2,1) \vee -at(p,3,2,2) \implies move(p,2,2,3,2,2) \vee move(p,3,1,3,2,2)...$
pre všetky možné smery

Keď sa zmenila pozícia debny, tak sa mohol vykonať jeden z týchto presunov debien pre každý stav:

$-at(b,3,2,1) \vee -at(b,3,2,2) \implies move(b,2,2,3,2,2) \vee move(b,3,1,3,2,2)...$
pre všetky možné smery

Keď sa zmenila pozícia debny, tak sa mohol vykonať jeden z týchto presunov hráča pre každý stav:

$-at(b,1,3,1) \vee -at(b,1,3,2) \implies move(p,3,3,2,3,2) \vee move(p,1,1,1,2,2)...$
pre všetky možné smery

Keď sa hráč presunul v nasledujúcom stave na miesto, kde nie je debna, všetky debny ostanú na mieste a nevykonalo sa žiadny posun debny:

$move(p,3,3,2,3,2) \wedge -at(b,2,3,1) \implies (at(b,5,6,1) => at(b,5,6,2))$

3.8 Background knowledge

V tejto časti sú zapísané takzvané informácie v pozadí. Informácie, ktoré sa vzťahujú na tento problém a tým pádom ju pomáhajú riešiť.

V každom kroku sa musí vykonať krok hráča:

$move(p,1,2,1,3,1) \vee move(p,1,2,1,3,1) \vee move(p,2,1,1,1,1) \vee move(p,5,2,4,2,1)...$

Keď sa hrač presúva v nasledujúcom stave na pozíciu, na ktorej je debna, musí sa presunúť v tom istom stave aj debna:

$$move(p,1,2,1,3,1) \wedge at(b,1,3,0) => move(b,1,3,1,4,1)$$

Hráč a debna si nemôžu vymeniť pozície:

$$at(b,1,2,1) \wedge at(p,1,3,1) => -at(b,3,1,2) \wedge -at(b,1,2,2)$$

Debna sa nemôže vrátiť do predchádzajúcej pozície:

$$-move(b, 1, 2, 1, 3, 1) \vee -move(b, 1, 3, 1, 2, 2)$$

4 Automatizovanie a Vizualizácia

4.1 Automatizovanie

Príloha obsahuje bash script, ktorý slúži na automatické vypočítanie riešenia na minimálny počet krokov. Po spustení skriptu stačí zadať cestu k súboru a skript nám hneď začne vypočítavať riešenia pre kroky od 1 až do prvého úspešného riešenia. Toto riešenie zapisuje do súboru "result.txtä zároveň zapisuje iba pravdivé predikáty do súboru "finalresult.txt". Tento súbor vlastne obsahuje postupnosti krokov, ktoré treba vykonať na vyriešenie problému.

4.2 Vizualizácia

Po skončení automatického vypočítavania si môžeme pustiť vizualizáciu riešenia. Vizualizáciu môžeme spustiť aj samostatne, stačí po spustení zadať mapu a výsledok v súbore "finalresult.txt".

5 Záver

V tomto projekte sme dosiahli pomocou plánovania vyriešenie hry Sokoban. Na riešenie sme použili program Minisat, pre ktorý sme generovali vstup podľa hore uvedených faktov a akcií. Generovanie pre Minisat bolo vyriešené v jazyku Python. Vizualizácia je napísaná v jazyku Java.

6 Zdroje

1. https://en.wikipedia.org/wiki/Sokoban [2017.11.16]