VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

HRA SOKOBAN A UMĚLÁ INTELIGENCE

SOKOBAN GAME AND ARTIFICIAL INTELLIGENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Petr Žlebek

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. Jiří Dvořák, CSc.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Petr Žlebek**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**RNDr. Jiří Dvořák, CSc.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Hra Sokoban a umělá inteligence

Stručná charakteristika problematiky úkolu:

Diplomová práce se věnuje problematice hry Sokoban a metod umělé inteligence. Sokoban je japonská hra pro jednoho hráče, ve které se hráč pohybuje v bludišti a musí posouvat předměty (kameny) na cílové pozice. Bludiště je 2D mapa zobrazující pozici hráče a rozmístění zdí, kamenů a cílových pozic. Kameny je nutné tlačit, nikoli táhnout a v každém okamžiku je možné posouvat pouze jeden kámen. Hra má řadu variant a byla řešena různými metodami umělé inteligence.

Cíle diplomové práce:

1. Analyzovat hru Sokoban a možnosti jejího řešení pomocí metod umělé inteligence.
2. Implementovat návrh umělého hráče hry Sokoban využívajícího vybrané metody umělé inteligence.
3. Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

YANNAKAKIS, G. N., TOGELIUS, J. Artificial Intelligence and Games. New York, Springer, 2018.

SETIANI, R. T., INDRIYONO, B. V. Implementation of A* Algorithm for Solving Sokoban Logic Games. Journal of Applied Intelligent System, vol. 4, no. 2, 2019, pp. 104-111.

PEREIRA, A. G., RITT, M., BURIOL, L. S. Optimal Sokoban Solving using Pattern Databases with Specific Domain Knowledge. Artificial Intelligence, vol. 227, 2015, pp. 52-70.

BENTO, D. S., PEREIRA, A. G., LELIS, L. H. S. Procedural Generation of Initial States of Sokoban. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), pp. 4651-4657, 2019.

KANTHARAO, V. et al. Expert System to Build the Cognitive Model of a Student Using the Sokoban Game and for Career Assessment. International Journal of Advanced Science and Technology, vol. 29, no. 5, 2020, pp. 9331-9342.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato práce je zaměřena na řešení hry Sokoban metodami umělé inteligence. Teoretická část popisuje hru Sokoban, problematiku stavového prostoru a princip vybraných prohledávacích algoritmů. V rámci praktické části byly v jazyce Python implementovány popsané algoritmy a bylo vytvořeno grafické uživatelské rozhraní. V závěrečné části byly provedeny srovnávací experimenty.

ABSTRACT

The thesis is focused on solving the Sokoban game using artificial intelligence algorithms. The first part of the thesis describes the Sokoban game, state space and selected state space search methods. In the second part selected methods were implemented and graphic user interface was created in the Python environment. Comparative experiments were executed in the final part.

KLÍČOVÁ SLOVA

Sokoban, hledání nejkratší cesty grafem, neinformované prohledávání, informované prohledávání, Single Player Monte Carlo Tree Search

KEYWORDS

Sokoban game, shortest path problem, uninformed search, heuristic search, Single Player Monte Carlo Tree Search

2021

BIBLIOGRAFICKÁ CITACE

ŽLEBEK, Petr. *Hra Sokoban a umělá inteligence*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2021, 74 s. Diplomová práce. Vedoucí práce: RNDr. Jiří Dvořák, CSc.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2021 .....

Petr Žlebek

PODĚKOVÁNÍ

Děkuji tímto svému vedoucímu diplomové práce RNDr. Jiřímu Dvořákovi, CSc. za cenné rady a připomínky.

Rád bych poděkoval svým rodičům za materiální i duševní podporu během celého náročného období vzdělávání se. Obrovské díky patří také mé přítelkyni Lucii, která mi byla oporou v průběhu psaní i studia.

OBSAH

1 ÚVOD

Hra Sokoban je hlavolam, který i přes svá jednoduchá pravidla dokáže potrápit lidský mozek. K tomu, aby člověk úlohu hry Sokoban vyřešil, musí prokázat schopnost kreativně a logicky uvažovat. Musí tedy využít svou *inteligenci*.

Záhy po vydání první verze se hra Sokoban stala objektem zkoumání vědců i nadšenců zabývajících se umělou inteligencí. Za necelých 40 let tak vznikla celá řada umělých řešitelů hry Sokoban, které se svým charakterem řadí mezi problémy prohledávání stavového prostoru.

Cílem této diplomové práce je shrnout současný stav poznání oblasti řešení hry Sokoban metodami umělé inteligence, popsat a implementovat vlastní umělé hráče řešící úlohy hry Sokoban a provést s nimi srovnávací experimenty.

Práce je rozdělena na teoretickou a praktickou část. Teoretická část nejprve popisuje historii, pravidla, cíle a speciální herní situace hry Sokoban a zmiňuje některé významné řešitele hry využívající metody umělé inteligence. Dále jsou shrnuty poznatky z problematiky stavového prostoru, které jsou nejdříve definovány formálně, a pak aplikovány na konkrétní příklady ze hry Sokoban. V této části jsou také popsány ty algoritmy prohledávání stavového prostoru, které byly vybrány k implementaci v praktické části.

Praktická část práce obsahuje podrobný popis vytvořeného grafického uživatelského rozhraní. Je v ní dále nastíněn princip nejvýznamnějších doplňků určených k urychlení a zpřesnění umělých řešitelů. Detailně je popsána implementovaná heuristická funkce. Závěr praktické části práce tvoří popis uskutečněných experimentů a vyhodnocení jejich výsledků.

2 TEORETICKÁ ČÁST

V této kapitole je nejprve čtenář uveden do problematiky hry Sokoban a obeznámen s několika vybranými možnostmi jejího řešení pomocí metod umělé inteligence. Dále jsou zde shrnuty poznatky z oblasti stavového prostoru. Kapitola je zakončena popisem několika vybraných algoritmů prohledávání stavového prostoru.

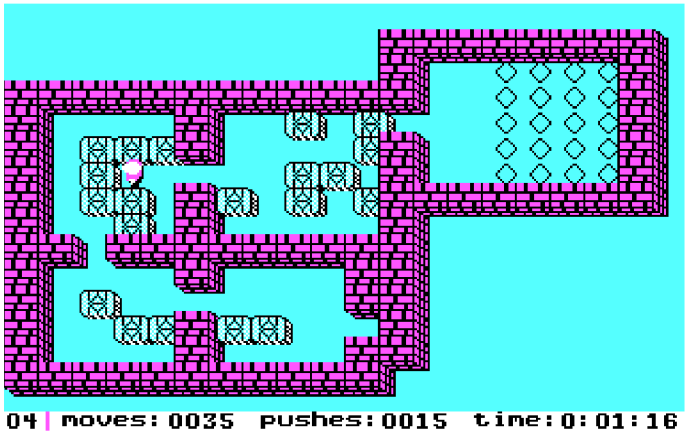
2.1 Hra Sokoban

V následujícím je čtenáři stručně představena historie hry, její pravidla a její cíl. Je zde zmíněna problematika složitosti řešení hry Sokoban. Jsou také jmenovány a krátce popsány některé významné řešitele této hry využívající metody umělé inteligence.

2.1.1 Historie

Sokoban je logická hra pro jednoho hráče. Jejím autorem je Hiroyuki Imabayashi. Hru vytvořil v 80. letech 20. století v Japonsku[1] jako svůj projekt pro soutěž o nejlepší počítačovou hru. Hra v soutěži zvítězila.

První komerční verze hry byla publikována společností Thinking Rabbit v roce 1982. V následujících letech byly publikovány další dvě rozšíření obsahující nové úrovně. Do roku 1987 bylo v Japonsku prodáno na 400 000 nosičů hry [1], čímž zaujala americkou společnost Spectrum HoloByte. Ta se stala distributorem v USA a v roce 1988 vydala verzi[2] hry Sokoban pro platformu IBM Personal Computer. Snímek z této hry je na obrázku 1. V dnešní době je hra v modernějším provedení dostupná na všech běžných platformách. [3, 4]



Obr. 1: Snímek ze hry Soko-ban vydané v roce 1988 společností Spectrum HoloByte.

[1] Slovo „sokoban“ se dá z japonštiny přeložit jako „skladník“.

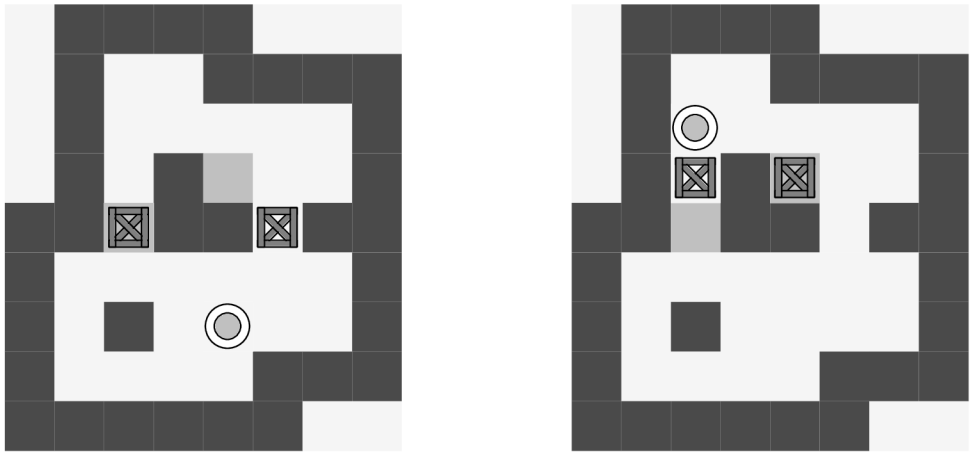
[2] Verzi hry vydanou společností Spectrum HoloByte lze hrát on-line na této adrese. [2]

2.1.2 Hrací plocha a její komponenty

Hrací plochu představuje mřížka tvořena čtvercovými poli. Čtvercová pole nabývají jedné z následujících sedmi hodnot: zeď, podlaha, cíl, agent, krabice, agent v cíli a krabice v cíli (viz obrázek 2).

Obr. 2: Grafické znázornění jednotlivých komponent.

Instance úlohy je definována počtem řádků, počtem sloupců, pozicemi zdí a pozicemi cílů. Tyto komponenty hrací plochy jsou *neměnné* po celou dobu řešení úlohy. Naopak agent a krabice jsou komponenty *pohyblivé*. Ukázka téže úlohy ve dvou různých fázích se nachází na obrázku 3.



Obr. 3: Tatáž úloha lišící se pozicemi pohyblivých komponent.

Na hrací ploše se musí vyskytovat právě jeden agent. Počet krabic musí být stejný jako počet cílů, přičemž je žádoucí, aby úloha obsahovala alespoň jednu krabici (resp. cíl). Zpravidla platí, že čím více krabic (resp. cílů) úloha obsahuje, tím obtížnější je úlohu vyřešit. Počet řádků, počet sloupců a počty ostatních komponent nejsou nijak omezeny.

2.1.3 Cíl hry a pravidla

Hra Sokoban si svou popularitu získala mimo jiné svými snadnými a intuitivními pravidly. Agent se smí po hrací ploše pohybovat o jedno políčko ve vertikálním nebo horizontálním směru. Povolenými tahy hráče jsou:

- přesun agenta na pozici, která neobsahuje zeď ani krabici,

- potlačení krabice do nové pozice (neobsazené jinou krabicí ani zdí) a přesun agenta na původní pozici krabice.

Agent tedy nesmí přeskakovat krabice ani zdi a nesmí ani krabice táhnout. V jednu chvíli smí tlačit nanejvýš jednu krabici.

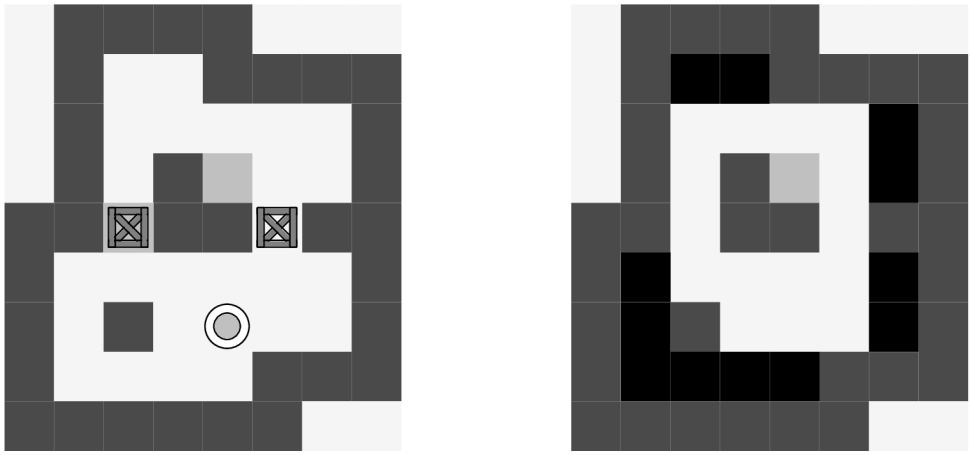
Cílem hry je v co nejmenším počtu tahů přemístit krabice tak, aby se všechny nacházely v cílech.

2.1.4 Uváznutí

Tahy hráče mohou hrací plochu uvést do stavu, který se nazývá uváznutí (angl. „deadlock“). Hrací plocha se nachází ve stavu uváznutí právě tehdy, kdy už není možné úlohu úspěšně vyřešit, tj. přemístit všechny krabice do cílových pozic. Uváznutí se dělí na dva základní typy - triviální a netriviální.

Triviální uváznutí

Triviální uváznutí (angl. „simple deadlock“) je vymezeno množinou pozic vedoucích k triviálnímu uváznutí. Tato množina obsahuje takové pozice na hrací ploše, pro které bezpodmínečně platí, že dostane-li se do této pozice jedna z krabic, úloha se stává neřešitelnou. Tyto pozice jsou závislé pouze na neměnných komponentech hrací plochy (počet řádků, počet sloupců, zdi a cíle). U každé úlohy tedy stačí detekovat tuto množinu pouze jedenkrát. Obrázek 4 obsahuje vyznačené pozice vedoucí k triviálnímu uváznutí.

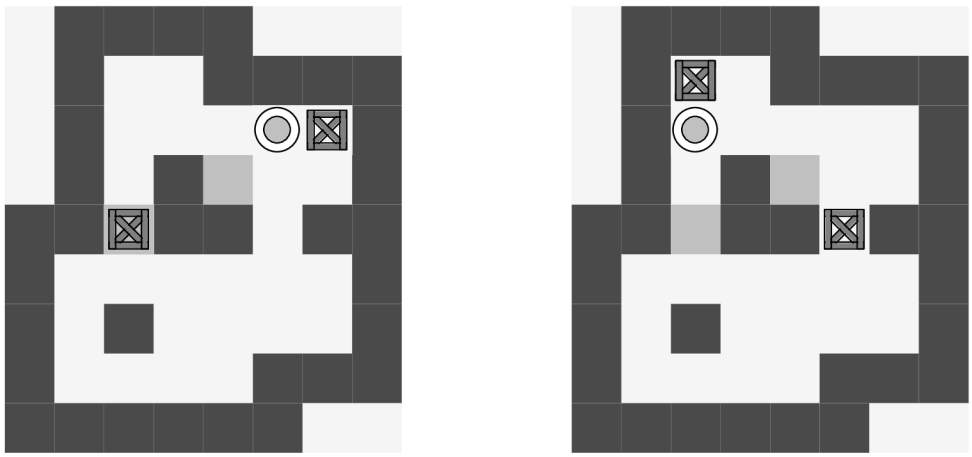


Obr. 4: Pozice vedoucí k triviálnímu uváznutí (označeny černě) pro konkrétní úlohu.

Hrací plocha je ve stavu triviálního uváznutí právě tehdy, když se pozice alespoň jedné z krabic nachází v množině pozic vedoucích k triviálnímu uváznutí. Obrázek 5 zobrazuje dva konkrétní příklady triviálního uváznutí.

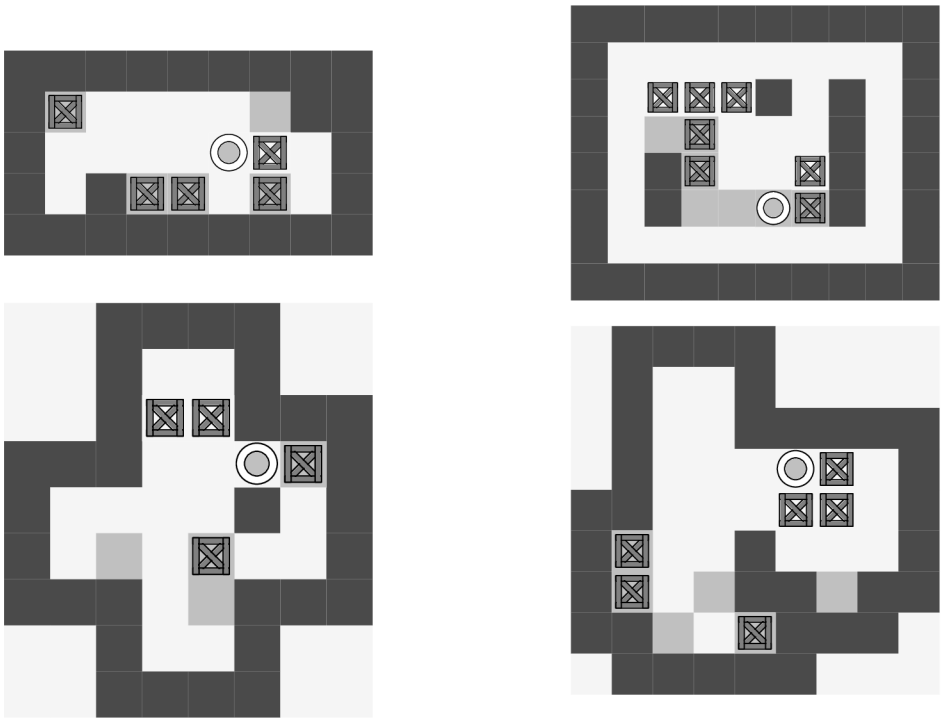
Netriviální uváznutí

Netriviální uváznutí (angl. „non-simple deadlock“) jsou velmi obtížně detekovatelná. Je totiž třeba brát v potaz vzájemné interakce mezi jednotlivými krabicemi, případně



Obr. 5: Příklady triviálního uváznutí.

mezi krabicemi a agentem [5]. Je zřejmé, že netriviální uváznutí nelze jednoznačně definovat, protože závisí na neměnných i pohyblivých komponentech hrací plochy. Na obrázku 6 je ukázáno několik konkrétních příkladů netriviálního uváznutí.



Obr. 6: Příklady netriviálních uváznutí.

2.1.5 Složitost

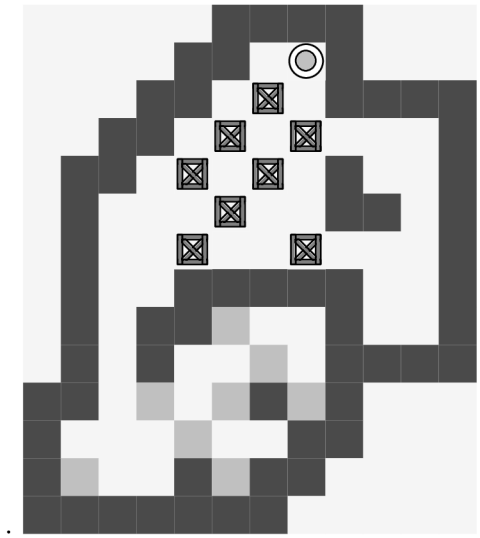
Hra Sokoban je kvůli své složitosti tématem pro oblast umělé inteligence. Stejně jako jiné logické hry je také cílem výzkumů oboru teorie složitosti. Teorie složitosti se zaměřuje na klasifikaci problémů dle časové nebo paměťové náročnosti jejich řešení.

Z hlediska teorie složitosti se jedná o NP-složitý problém [6]. Bylo také dokázáno, že hra Sokoban je PSpace-úplná [7].

Složitost hry je ovlivněna několika okolnostmi. Faktor větvení je závislý na počtu krabic. Každou krabici je možné posunout až do čtyř různých nových pozic. Hloubka řešení, tj. počet tahů nutný k nalezení řešení, bývá často velmi vysoká. Na složitosti přidává problému také možnost uváznutí (viz 2.1.4) [8]. Horní mez velikosti stavového prostoru d hry Sokoban je v práci Junghanns et. al. [9] stanovena dle následujícího vzorce:

$$d = \binom{r_a - q}{t} \cdot r_a, \tag{1}$$

kde r_a je celkový počet dosažitelných polí agentem, q je počet prvků množiny triviálních uváznutí a t je počet cílů, resp. krabic. Velikost stavového prostoru pro úlohu na obrázku 7 je řádově 10^{10}.



Obr. 7: Ukázka úlohy s osmi cíli, resp. krabicemi.

Při řešení téměř všech instancí úloh hry Sokoban je nutné občas uskutečnit tah, který hráče v danou chvíli na první pohled vzdaluje od úspěšného řešení. Je to však nezbytné například pro uvolnění cesty pro jinou krabici, která by se tak nedostala k cílové pozici. Také kvůli této skutečnosti je hra Sokoban výzvou nejen pro umělé hráče, ale i pro hráče lidské.

2.1.6 Významné řešitele využívající metody umělé inteligence

Existuje celá řada vědeckých prací věnující se řešení hry Sokoban pomocí metod umělé inteligence. V následujícím je popsáno několik významných umělých hráčů, kteří sloužili jako inspirace při psaní této práce.

Jedním z nejstarších a nejvýznamnějších řešitelů je „Rolling Stone“[3] vytvořený vědci z University of Alberta, Canada [5]. Kostrou algoritmu „Rolling Stone“ je algoritmus A* s iterativním prohlubováním. Tým pod vedením Andrease Junghannse se hře Sokoban věnoval několik let a svůj algoritmus postupně optimalizoval. V navazujících publikacích [10, 9, 11, 12] se objevovala vylepšení jako například použití transpozičních tabulek, vykonávání většího množství akcí najednou či využívání tabulek vzorových uváznutí [13]. Všechny tyto inovace vedly ke zmenšení prohledávaného stavového prostoru a tím tak k rychlejšímu nalezení řešení. „Rolling Stone“ a k němu vydané práce jsou dodnes hlubokou studnicí vědomostí využívaných při tvorbě nových řešitelů, ať už podobných, nebo využívajících jiných metod.

Dalším poměrně úspěšným řešitelem je algoritmus, jehož autorem je André G. Pereira et. al. [14, 15]. Při řešení využívá databáze vzorů (angl. *pattern databases*) pro zdokonalení výpočtu heuristické funkce algoritmu A*.

Mezi kvalitní algoritmy se řadí YASC (Yet Another Sokoban Clone). Jedná se o řešitel, který byl vytvořen k počítačové hře Sokoban pro Windows [16].

V porovnání [17] výkonnosti různých umělých hráčů se jako jeden z nejlepších jeví algoritmus FESS (FEature Space Search) [18]. Algoritmus využívá celou řadu doplňků urychlujících řešení a upřesňujících hodnotu heuristické funkce stavů.

2.2 Stavový prostor

Důležitým rysem všech inteligentních systémů, mezi které patří například i umělý hráč řešící hru Sokoban, je schopnost vytvářet modely prostředí pro následnou práci s nimi. Úkolem inteligentního systému je hledat vhodnou posloupnost akcí, jejichž aplikací lze přejít od jednoho předem definovaného modelu k jinému. Modely se označují jako stavy a množina všech stavů tvoří **stavový prostor**. Posloupnost akcí pak označujeme jako plán, resp. řešení úlohy nad stavovým prostorem. [19]

V dalším jsou formálně definovány pojmy stavový prostor, stav, přechody mezi stavy, úloha nad stavovým prostorem a řešení úlohy. Každá definice je vždy aplikována na případ hry Sokoban a je demonstrována na jednoduchém příkladu ze hry. Je také popsána souvislost mezi stavovým prostorem a teorií grafů.

2.2.1 Definice stavového prostoru

Formálně lze stavový prostor definovat jako uspořádanou dvojici $S = (D, \Phi)$, kde D je konečná množina stavů a Φ je konečná množina operátorů reprezentujících přechody mezi stavy. Každý operátor $\phi \in \Phi$ je parciálním zobrazením z množiny stavů do množiny stavů $\phi : D \mapsto D$. [20]

[3] V jiných textech mohou být krabice na rozdíl od této práce nazývány kameny. Sousloví „Rolling stone“ znamená v češtině „valící se kámen“.

2.2.2 Stav

Stav obecného stavového prostoru může být popsán například symbolem abecedy, číslem, n-ticí, textovým řetězcem, tabulkou, množinou atd. Volba zápisu stavu závisí na charakteru řešeného problému. Je žádoucí volit zápis tak, aby kladl co nejnižší nároky na paměť a na výpočetní náročnost přechodů mezi stavy a aby jím byly stavy jednoznačně určeny. Volba zápisu stavu může mít velký vliv na výkon umělého řešitele.

Stav hry Sokoban

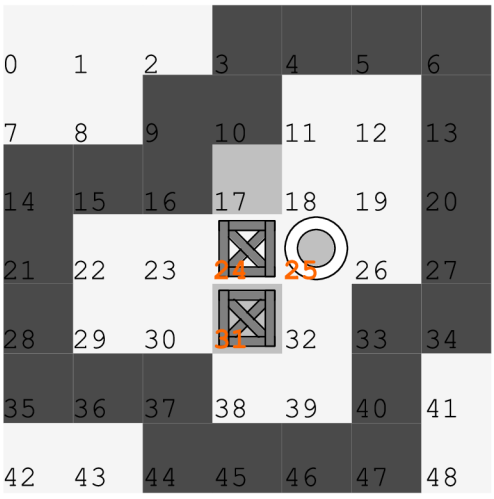
Stavem hry Sokoban je dvojice $s_i = (a_i, B_i) \in D$, kde a_i je pozice agenta a B_i je množina pozicí krabic.

Není nutné, aby zápis stavu obsahoval informaci o neměnných komponentech hrací plochy. Je ale žádoucí tyto hodnoty zaznamenávat. Neměnné komponenty hrací plochy tvoří čtveřici $Q = (r, c, W, T)$, kde r je počet řádků hracího pole, c je počet sloupců hracího pole, W je množina pozicí zdí a T je množina pozicí cílů. Čtveřici $Q = (r, c, W, T)$ můžeme označit jako vlastnosti úlohy.

Vlastnosti úlohy Q a stav hry s_i Sokoban dohromady umožňují vytvořit obrazovou podobu stavu. Ta je důležitá pro komunikaci s uživatelem v rámci grafického uživatelského rozhraní hry.

Zaznamenávání pozic

Pozice je výraz využívaný v souvislosti s agentem, krabicemi, cíli a zdmi. V publikacích věnujících se řešení hry Sokoban je zvykem využívat tzv. *řádkové indexování pozic*. Pozice komponentu hrací plochy je zaznamenána jako $z \in \mathbb{N}$ splňující podmínku $0 \leq z < r \cdot c$. Na obrázku 8 je s využitím řádkového indexování určena například pozice agenta jako $a_i = 25$. Je zřejmé, že tento způsob zápisu pozice má nižší nároky na paměť než zápis formou dvou souřadnic v mřížce hrací plochy.



Obr. 8: Ukázka řádkového indexování hrací plochy hry Sokoban.

2.2.3 Operátor a přechod mezi stavy

Z výše uvedené formální definice stavového prostoru plyne

$$s_1 = \phi(s_0), \tag{2}$$

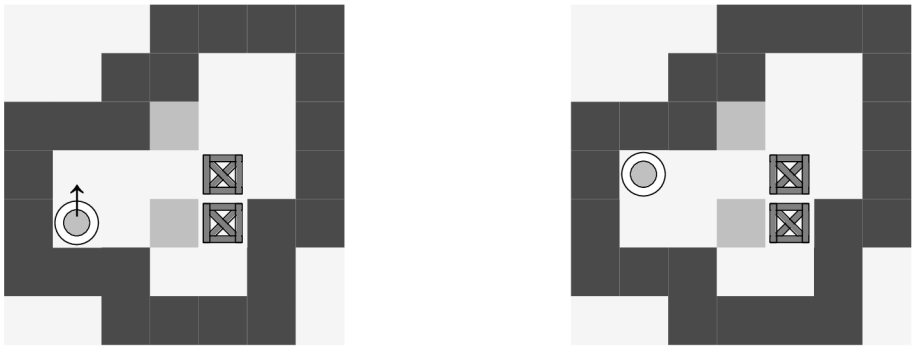
kde $s_0, s_1 \in D$ a $\phi \in \Phi$. Operátor ϕ reprezentuje přechod ze stavu s_0 do stavu s_1.

Množina operátorů a faktor větvení hry Sokoban

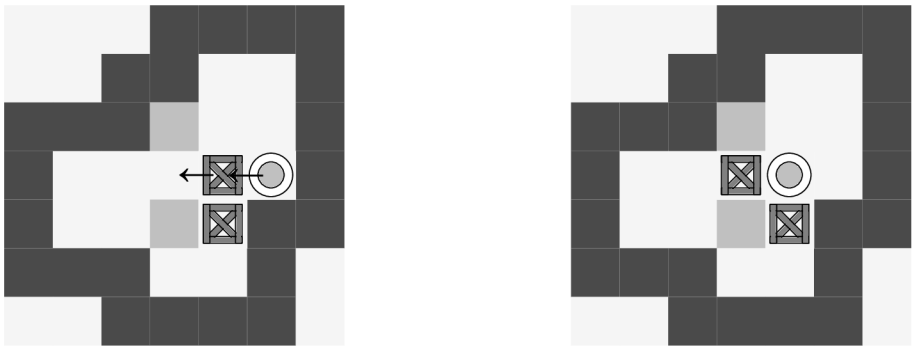
Na množinu operátorů Φ lze ve hře Sokoban pohlížet dvěma způsoby. Intuitivně

$$\Phi = \{\phi_u, \phi_r, \phi_d, \phi_l\}, \tag{3}$$

kde například ϕ_u symbolizuje přesun agenta na hracím poli nahoru[4] o jednu pozici, resp. potlačení krabice a přesun agenta nahoru o jednu pozici. Znázornění těchto přechodů je na obrázcích 9 a 10.



Obr. 9: Přechod jako přesun agenta.



Obr. 10: Přechod jako přesun agenta s posunem krabice.

Připomeňme ale, že cílem hry Sokoban je pohyby agenta přemístit krabice tak, aby se všechny nacházely v cílech. Je tedy zřejmé, že relevantními kroky pro dosažení tohoto cíle jsou pouze takové akce, kdy agent tlačí krabici.

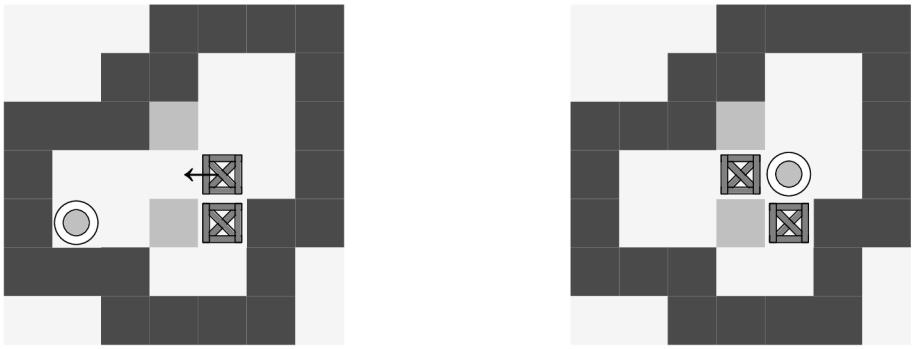
[4] Indexy u, r, d a l jsou zkratkami anglických slov „up“, „right“, „down“ a „left“.

Dle vzoru jiných publikací je tedy množina operátorů definována jako

$$\Phi = \{\phi_u^{b_i^1}, \phi_r^{b_i^1}, \phi_d^{b_i^1}, \phi_l^{b_i^1}, \ldots, \phi_u^{b_i^t}, \phi_r^{b_i^t}, \phi_d^{b_i^t}, \phi_l^{b_i^t}\}, \tag{4}$$

kde $b_i^1, \ldots, b_i^t \in B_i$ a $t = |T| = |B_i|$ je počet cílů (resp. krabic) dané instance úlohy.

Například ϕ_l^{25} symbolizuje potlačení krabice nacházející se na hracím poli na pozici 25 agentem směrem vlevo. Znázornění tohoto přechodu je na obrázku 11.



Obr. 11: Přechod jako přímé posunutí krabice.

Při použití množiny operátorů (3) je faktor větvení $b = 4$. Při použití množiny operátorů (4) je faktor větvení $b = 4 \cdot t$ a je tedy větší než při (3). Je tak ale snížena velikost stavového prostoru.

Přechod mezi stavy hry Sokoban

Přechod mezi stavy pomocí operátorů definovaných v (4) se skládá ze dvou kroků:

1. Přesun agenta na pozici vedle krabice, kterou agent posune.
2. Samotný posun krabice agentem o jedno políčko patřičným směrem.

První krok přechodu je úlohou hledání nejkratší cesty z pozice agenta do potřebné pozice, přičemž všechny krabice jsou považovány za zdi. Pro proveditelnost daného přechodu je nutné, aby potřebná pozice byla pro agenta při vzájemném rozmístění krabic dosažitelná. Pro ověřování dosažitelnosti definujme množinu R_i, která obsahuje všechny dosažitelné pozice agentem ve stavu $s_i = (a_i, B_i) \in D$. Na obrázku 12 jsou vyznačeny všechny prvky této množiny v konkrétním stavu úlohy.

Obr. 12: Zvýraznění všech agentem dosažitelných pozic.

Přechod mezi stavy a podmínky proveditelnosti hry Sokoban formálně

Přechod ze stavu $s_0 = (a_0, B_0)$ pomocí operátoru $\phi_{dir}^{b_0^k}$, kde $dir \in \{u, r, d, l\}$ a $b_0^k \in B_0$ lze vyjádřit podle (2) tímto způsobem

$$s_1 = \phi_{dir}^{b_0^k}(s_0) = \phi_{dir}^{b_0^k}((a_0, B_0)) = (a_0 + m_{dir}, \{b_0^1, b_0^2, \ldots, b_0^k + m_{dir}, \ldots, b_0^t\}), \tag{5}$$

kde při použití řádkového indexování platí $m_u = -c, m_r = 1, m_d = c, m_l = -1$.

Řekneme, že přechod $\phi_{dir}^{b_0^k}$ je ve stavu $s_0 = (a_0, B_0)$ proveditelný právě tehdy, když jsou splněny všechny následující podmínky.

- Pozice těsně vedle krabice nutná pro potlačení je agentem dosažitelná:

$$b_0^k - m_{dir} \in R_0. \tag{6}$$

- Nová pozice krabice neleží mimo hrací plochu. V případě potlačení vlevo (resp. vpravo) je vzhledem k řádkovému indexování nutné ověřit, že tlačená krabice není v prvním (resp. posledním) sloupci hrací plochy:

$$0 \leq b_0^k + m_{dir} < c \cdot r, \tag{7}$$

$$dir = l \rightarrow b_0^k \ mod \ c \neq 0, \tag{8}$$

$$dir = r \rightarrow b_0^k \ mod \ c \neq c - 1. \tag{9}$$

- Nová pozice krabice není obsazena jinou krabicí nebo zdí:

$$b_0^k + m_{dir} \notin W \cup B_0. \tag{10}$$

2.2.4 Úloha nad stavovým prostorem a její řešení

Úloha U nad stavovým prostorem $S = (D, \Phi)$ je dvojice $U = (s_0, C)$, kde $s_0 \in D$ je počáteční stav a $C \subseteq D$ je množina cílových stavů. [20]

Plánem P pro danou úlohu U je taková posloupnost operátorů $P = (\phi_1, \phi_2, \ldots, \phi_n)$, ke které lze přiřadit posloupnost stavů $(s_0, s_1, \ldots, s_n)$, pro kterou platí:

$$s_1 = \phi_1(s_0), s_2 = \phi_2(s_1), \ldots, s_n = \phi_n(s_{n-1}), s_n \in C. \tag{11}$$

Plán P je též nazýván **řešením** úlohy U. [19]

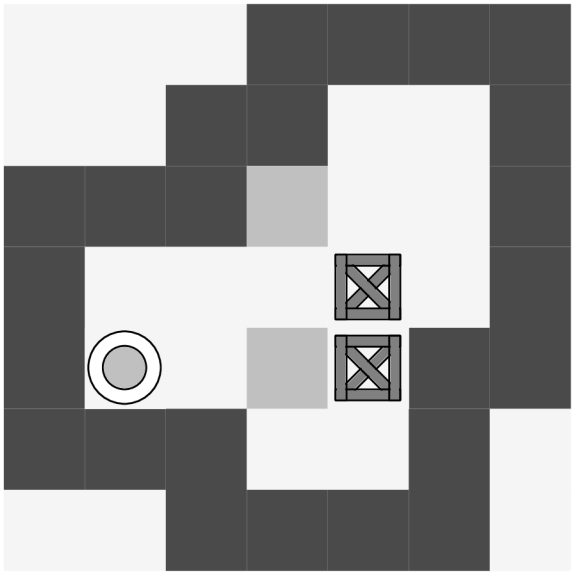
Úloha a řešení hry Sokoban

Úloha nad stavovým prostorem hry Sokoban je definována počátečním stavem $s_0 \in D$ a vlastnostmi úlohy $Q = (r, c, W, T)$. Z vlastností Q je odvozena množina cílových stavů C tak, že množina C obsahuje všechny stavy $s_c = (a_c, B_c) \in D$, pro které platí

$$B_c = T. \tag{12}$$

Jinak řečeno, prvky množiny cílových stavů jsou všechny stavy, jejichž množina pozicí krabic je rovna množině pozicí cílů.

Například úloha nad stavovým prostorem hry Sokoban na obrázku 13 je definována počátečním stavem $s_0 = (a_0, B_0) = (29, \{25, 32\})$ a vlastnostmi úlohy $Q = (r, c, W, T) = (7, 7, W, \{17, 31\})$. Množina pozicí zdí W obsahuje 24 prvků, a tak byla její konkrétní podoba vynechána.



Obr. 13: Ukázka obrazové podoby úlohy nad stavovým prostorem hry Sokoban.

Řešením hry Sokoban je pak plán P splňující podmínky (11). Je standardem umělých řešitelů hry Sokoban označovat přesun agenta malými písmeny u, r, d, l v případě, že je měněna pouze pozice agenta a velkými písmeny U, R, D, L v případě, že agent tlačí krabici. Řešením hry Sokoban je tedy posloupnost znaků z abecedy $\{u, r, d, l, U, R, D, L\}$. Například pro úlohu na obrázku 13 je řešením posloupnost $P = rururrdLLddrUUldlluRdrruruulDlDurrdLdLUlldR$.

2.2.5 Grafová reprezentace stavového prostoru

Orientovaný graf G je dvojice $G = (V, E)$, kde V je množina uzlů grafu a E je množina hran grafu. Prvky množiny $E \subseteq V^2 = V \times V$ jsou uspořádané dvojice $e = (v_1, v_2)$, kde $v_1, v_2 \in V$. Platí, že v_1 je počáteční uzel hrany a v_2 je koncový uzel hrany. [21]

Stavový prostor $S = (D, \Phi)$ může být reprezentován orientovaným grafem $G = (V, E)$. Terminologicky bývá ztotožňována množina stavů D s množinou uzlů V a množina přechodů Φ s množinou hran E.

Tato reprezentace umožňuje využívat obecných poznatků teorie grafů. Pro hledání řešení úloh nad stavovým prostorem tedy lze využít metody prohledávání grafu, kterým je věnována následující podkapitola. [20]

2.3 Metody prohledávání a popis vybraných algoritmů

Většina problémů, kterými se zabývá umělá inteligence, se dá reprezentovat grafem, resp. jeho speciálním typem - stromem. [21] Řešení problémů pak spočívá v hledání cesty od kořene stromu (počáteční stav úlohy) až po listy stromu (množina koncových stavů). Všechny níže uvedené algoritmy patří mezi **metody prohledávání stromu**. Jednotlivé algoritmy se (mimo jiné) liší postupem, kterým se strom větví a schopností najít buďto optimální, nebo pouze suboptimální řešení. [22]

Klasické metody prohledávání se dělí na **neinformované** a **informované** podle toho, zda využívají znalosti o úloze. Zvláštní skupinu metod prohledávání tvoří tzv. **heuristické algoritmy**. Ty ve většině případů také využívají znalosti o úloze, ale kostra algoritmu je jiná než u klasických metod. [20]

Obecně může být algoritmus prohledávání interpretován jako funkce, do které vstupují následující parametry:

- počáteční stav $s_0 \in D$,
- množina koncových stavů $C \subseteq D - \{s_0\}$,
- metoda *expandState*(s_i), která má na vstupu libovolný stav $s_i \in D$ a na výstupu všechny jeho potomky.

Na výstupu je pak buďto nalezená cesta z počátečního stavu do jednoho z koncových, nebo informace o neúspěchu algoritmu.

Klasické metody prohledávání se vyznačují tím, že při řešení využívají datové struktury *OPEN* a *CLOSED*. Struktura *OPEN* obsahuje ty stavy, které mají být expandovány a do *CLOSED* jsou postupně přidávány expandované stavy. Klasické metody jsou obecně implementovány dle následujícího pseudokódu.

Algoritmus 1: Obecná metoda prohledávání stromu

```
Function findShortestPath(s_0, C, expandState):
    addToOpen(s_0);
    while openIsNotEmpty() do
        s_c = chooseToExpand();
        addToClosed(s_c);
        E = expandState(s_c);
        foreach s_e ∈ E do
            if s_e ∈ C then
                return tracePath(s_e);
            end
            handleExpanded(s_e);
        end
    end
    return False;
```

Jednotlivé algoritmy se od sebe odlišují implementací metod *chooseToExpand* a *handleExpanded*. Ty ovlivňují to, který stav je vybrán z *OPEN* k expanzi a jak je naloženo s expandovanými stavy. Všechny níže uvedené algoritmy byly vybrány k implementaci umělých hráčů v praktické části této práce.

2.3.1 Neinformované metody prohledávání

Rozdílem mezi různými neinformovanými metodami prohledávání je způsob, kterým je vybírán stav k expanzi. Nakládání s expandovanými stavy je pro následující dvě metody stejné. Do struktury *OPEN* jsou přidány pouze ty stavy, které se zatím nenachází v *OPEN* ani *CLOSED*.

Algoritmus Breadth First Search

Algoritmus Breadth First Search (prohledávání do šířky) je velmi systematickou strategií. Metoda prohledává nejprve všechny cesty délky 1, potom všechny cesty délky 2 a tak dále. Z toho plyne, že je-li nalezeno řešení, pak je optimální. [23]

Při prohledávání do šířky je k expanzi vybrán ten stav, který byl do struktury *OPEN* přidán nejdříve. Struktura *OPEN* má tedy charakter fronty. [24]

Algoritmus Depth First Search

Na rozdíl od prohledávání do šířky má algoritmus Depth First Search (prohledávání do hloubky) nižší požadavky na paměť. Expandován je vždy jeden z doposud neexpandovaných uzlů s největší hloubkou. Řešení nalezené touto strategií je obecně suboptimální. [23]

Při použití této metody je k expanzi vybrán ten stav, který byl do struktury *OPEN* přidán jako poslední. Struktura *OPEN* má tedy charakter zásobníku. [24]

Srovnání neinformovaných metod

V tabulce 1 je porovnána časová a paměťová náročnost výše popsaných metod a také úplnost[5] a optimálnost nalezeného řešení.

Tab. 1: Tabulka srovnání neinformovaných metod prohledávání grafu.

algoritmus	čas	paměť	úplnost	optimálnost
Breadth First Search	$O(b^{l+1})$	$O(b^m)$	ano	ano
Depth First Search	$O(b^m)$	$O(b \cdot m)$	ano	ne

b - konečný faktor větvení, l - délka nejkratší cesty a m - konečná maximální hloubka stromu.

Z tabulky 1 vyplývá, že obě uvedené neinformované metody prohledávání mohou vzhledem k velikosti stavového prostoru být velmi neefektivní. Je zbytečně prohledávána velká část stavového prostoru, která nevede k cílovým stavům. Tomu lze předejít využitím informovaných metod prohledávání. [20]

[5] Úplnost je schopnost algoritmu vždy nalézt řešení, pokud existuje.

2.3.2 Informované metody prohledávání

Informované metody prohledávání využívají při postupování stromem znalost o daném problému. Tato znalost je reprezentována hodnotící funkcí $f : D \to \mathbb{R}^+$. Funkci f můžeme rozepsat jako

$$f(s_i) = g(s_i) + h(s_i). \tag{13}$$

Funkce $g : D \to \mathbb{R}^+$ je nejkratší dosud nalezená vzdálenost z počátečního stavu s_0 do stavu s_i. Jestliže je stav s_i potomkem stavu s_{i-1}, pak platí

$$g(s_i) = g(s_{i-1}) + a_i, \tag{14}$$

kde a_i je délka cesty mezi stavem s_i a jeho rodičem s_{i-1}. Pro počáteční stav platí

$$g(s_0) = 0. \tag{15}$$

Funkce $h : D \to \mathbb{R}^+$ je tzv. **heuristická funkce**. Heuristická funkce $h(s_i)$ je přípustná právě tehdy, když je nezáporným dolním odhadem skutečné délky optimální cesty mezi stavem s_i a jedním z koncových stavů $s_c \in C$. Pro funkci neexistuje žádný obecný předpis a je (obzvláště u rozsáhlých úloh) velmi obtížné určit, zda je přípustná. To je největší slabinou informovaných metod. Funkce $h(s_i)$ je ohodnocením stavu s_i, a proto je také využívána jako *fitness funkce* heuristických algoritmů.

Čím více odpovídá hodnota heuristické funkce realitě, tím menší část stavového prostoru je při průchodu stromem prohledávána. V případě, že heuristická funkce odhaduje vzdálenost do koncového stavu úplně přesně, expanduje algoritmus pouze stavy nacházející se na optimální cestě. [20]

Informované metody prohledávání vybírají k expanzi ze struktury *OPEN* stav s nejnižší hodnotou funkce f. Proto se tato skupina algoritmů označuje jako **Best First Search** algoritmy. Jednotlivé metody se od sebe liší způsobem výpočtu funkce f. Nakládání s expandovanými stavy funguje u všech níže uvedených informovaných strategií dle následujícího pseudokódu.

Algoritmus 2: Metoda *handleExpanded* informovaných algoritmů.

```
Function handleExpanded(s_i):
    f = countValueF(s_i);
    if isInClosedOrOpen(s_i) then
        f' = getCurrentValueF(s_i);
        if f < f' then
            updateInOpen(s_i);
        end
    else
        addToOpen(s_i);
    end
```

Algoritmus Uniform-Cost Search

Algoritmus Uniform-Cost Search (algoritmus stejnoměrných cen) se neřadí mezi heuristické metody, ale do informovaných metod patří, protože využívá informace o délkách, resp. cenách přechodů mezi stavy. Platí tedy, že $f(s_i) = g(s_i)$. Takto definovaná hodnotící funkce zaručuje nalezení optimálního řešení.

Algoritmus Greedy Best First Search

Algoritmus Greedy Best First Search se též nazývá „hladový" algoritmus. Toto označení je příhodné, protože do výpočtu hodnotící funkce f vstupuje pouze heuristická funkce h. To způsobuje, že je expandován vždy ten stav, který je nejblíže k cíli. Platí tedy, že $f(s_i) = h(s_i)$. Nalezení optimálního řešení není obecně garantováno.

Algoritmus A*

Algoritmus A* patří mezi nejpoužívanější informovanou metodu prohledávání stavového prostoru. Hodnotící funkce f je ovlivněna jak funkcí g, tak heuristikou h. Platí tedy vztah (13).

Nalezení optimálního řešení je zaručeno právě tehdy, když je heuristická funkce h přípustná.

2.3.3 Metaheuristický přístup

Při řešení složitých problémů se stává, že klasické metody prohledávání, včetně těch popsaných výše, selhávají. Buďto nedokážou najít optimální řešení, nebo nedokážou v požadovaném čase najít ani dostatečně vyhovující suboptiomální řešení. Vzniká zde tak prostor pro heuristické metody, které dostatečně vyhovující řešení nalézt dokážou. Přístupy, které nejsou vázány na konkrétní typ úlohy, jsou tzv. *metaheuristiky*. [20]

Metaheuristik existuje celá řada. Některé z nich jsou inspirovány přírodními procesy jako jsou evoluční princip (genetické algoritmy) nebo hejnová inteligence (algoritmus mravenčí kolonie, rojové algoritmy). Jiné zase napodobují procesy technologické (algoritmus simulovaného žíhání). [25]

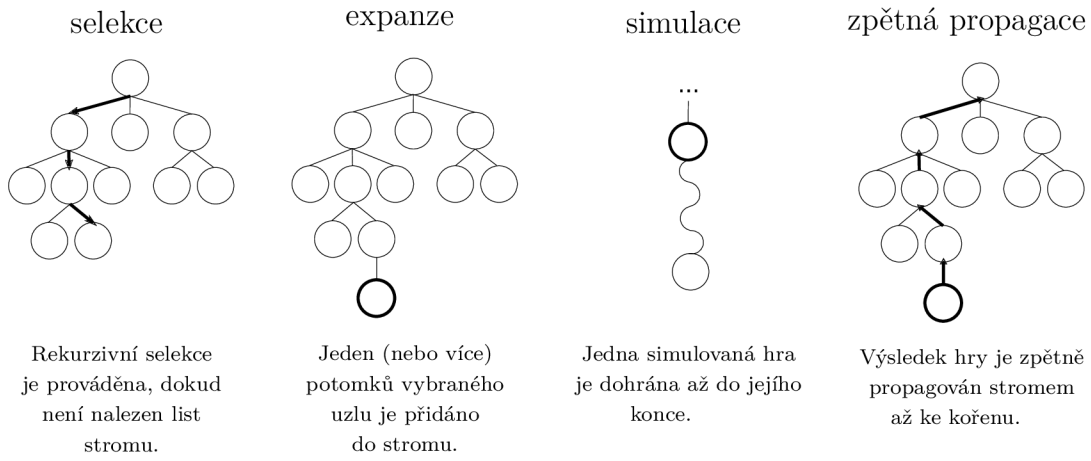
Metaheuristické algoritmy obecně nezaručují ani nalezení nějakého řešení úlohy. Dá se na ně ale pohlížet tak, že umožňují jakýsi kompromis mezi časem vynaloženým na hledání řešení a spokojením se s přijatelným suboptimálním řešením. Důležitým aspektem všech metaheuristik je hledání rovnováhy mezi objevováním nových řešení a rozvíjením těch doposud nejúspěšnějších (anglicky *exploration vs. exploitation*).

Monte Carlo Tree Search

Monte Carlo Tree Search (zkráceně MCTS) je algoritmus prohledávání stromu založený na metodě minimax. MCTS se nejčastěji používá při hrách dvou (a více) hráčů. Nejslavnější aplikací algoritmu je bezpochyby AlphaGo. Umělý hráč hrající

hru Go je kombinací MCTS s hlubokým učením neuronové sítě. [26] Algoritmus je také často využíván na poli počítačových her, zejména v realtimových strategiích (RTS) [27].

Vzhledem k tomu, že se MCTS využívá především při hraní her více hráčů, bývá jeho rozhodovací čas omezen. Algoritmus je tvořen čtyřmi procedurami, které se opakují ve smyčce tak dlouho, jak jen to daná aplikace a její rozhodovací čas dovoluje. Jednotlivé procedury se nazývají *selekce*, *expanze*, *simulace* a *zpětná propagace* a v dalším je naznačen jejich princip. Schémata jednotlivých procedur jsou na obrázku 14.



Obr. 14: Schéma algoritmu Monte Carlo Tree Search

1. *Selekce.* V této fázi je rozhodnuto o tom, který uzel grafu bude expandován. Proces selekce začíná u kořene stromu a končí v okamžiku, kdy je vybrán uzel, který ještě nebyl expandován. Při rozhodování o tom, do kterého uzlu přejít, je využita funkce *Upper Confidence Bound*. Může to být například UCB_1, která má následující tvar:

$$UCB_1(i) = \frac{x_i}{v_i} + c_e \cdot \sqrt{\frac{ln(q_j)}{q_i}} \tag{16}$$

   kde i je uzel stromu a j je jeho rodič, x_i je součet všech zpětně propagovaných odměn potomky uzlu, q_i a q_j odpovídají počtu dřívějších výběrů uzlů i a j ve fázi selekce a c_e je explorační koeficient. Každý uzel stromu je touto funkcí ohodnocen. Vybrán je vždy ten potomek jehož hodnota UCB_1 je nejvyšší.
2. *Expanze.* V této fázi je využita metoda *ExpandState(s_i)*. Do stromu jsou přidány expandované stavy. Je žádoucí přidávat do stromu pouze ty stavy, které v něm ještě nejsou, aby bylo předcházeno cyklům. Pro další fázi je náhodně vybrán jeden z nově přidaných uzlů.
3. *Simulace.* Fáze simulace je hlavní části algoritmu MCTS. Probíhá v ní náhodná simulace akcí zpravidla až do doby, než se algoritmus dostane ke koncovému

stavu hry, tedy ke stavu, ve kterém hráč buďto vyhrál, remizoval, nebo prohrál. Tato informace je vstupem pro poslední fázi algoritmu.

4. *Zpětná propagace.* Závěrečnou fází smyčky je procedura zpětné propagace výsledku náhodné simulace. Informaci o vítězství, remíze, nebo prohře se říká odměna a v tomto pořadí nabývá hodnot $x \in \{1, 0, -1\}$. Odměna je zpětně propagována stromem od koncového uzlu až ke kořenu stromu.

V okamžiku, kdy vyprší rozhodovací čas je vybrán jeden z potomků kořenu stromu dle strategie selekce. Vybraný tah hráče je vykonán a čeká se na tah soupeře. Detailněji je fungování algoritmu popsáno například v [28, 22].

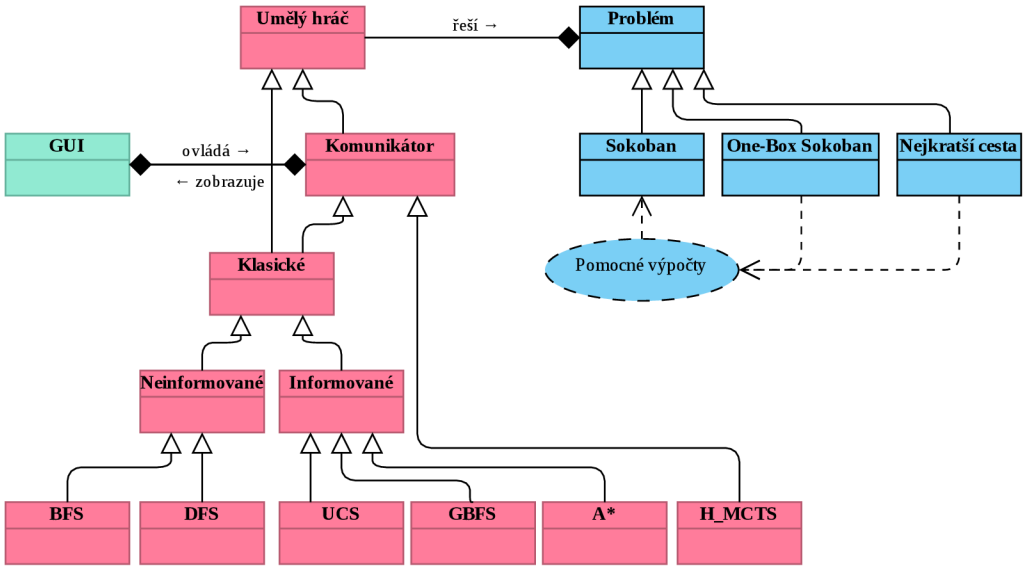
Single Player Monte Carlo Tree Search

Přestože je Monte Carlo Tree Search primárně určený pro hraní her dvou hráčů, existují i implementace pro hry jednoho hráče. Ty jsou nazvány Single Player Monte Carlo Tree Search, zkráceně SP-MCTS. Například Schadd se ve své práci [29] věnuje implementací MCTS na řešení hry SameGame. Cílem hry je zjednodušeně řečeno nahrát co nejvyšší skóre. Nelze tedy při implementaci fáze zpětné propagace MCTS postupovat standardně. Nejsou totiž definovány pojmy výhra, remíza a prohra. V závěrečné fázi klasického MCTS je tedy zpětně propagována právě hodnota skóre, které se umělému hráči v dané simulaci podařilo zahrát.

Algoritmem SP-MCTS je inspirován poslední z umělých hráčů řešících hru Sokoban implementovaných v praktické části práce. Byl vytvořen jakýsi hybridní algoritmus kombinující klasický MCTS a SP-MCTS s využitím heuristické funkce. Klíčové části algoritmu a rozdíly oproti klasickému algoritmu MCTS jsou v praktické části práce popsány.

3 IMPLEMENTACE UMĚLÝCH HRÁČŮ HRY SOKOBAN

Hlavním cílem práce je implementace vlastních umělých hráčů řešících hru Sokoban. K tomu byl s využitím programovacího jazyka Python [30] a principů objektově orientovaného programování [31] vytvořen systém, jehož schéma s vyznačením nejdůležitějších prvků je na obrázku 15. Systém lze rozdělit na tři hlavní části: grafické uživatelské rozhraní (zelené), stavové třídy (modré) a třídy umělých hráčů (červené). Jednotlivé části systému jsou vzájemně propojeny.



Obr. 15: Zjednodušené schéma systému.

Systém byl vytvořen tak, aby byl v co největší míře modulární. Je možné do něj snadno doplnit další umělé hráče. Je také možné využít kostru systému k řešení jiných problémů než je hra Sokoban. K tomu by bylo potřeba vytvořit třídu daného problému a v závislosti na aplikaci taky nové grafické uživatelské rozhraní.

Zdrojové kódy všech částí systému s vysvětlujícími komentáři jsou uloženy v archivu, který je přílohou této práce. Součástí archivu je také spustitelný soubor grafického uživatelského rozhraní pro operační systém Windows ve formátu *.exe* s názvem *Sokoban_Solver.exe*.

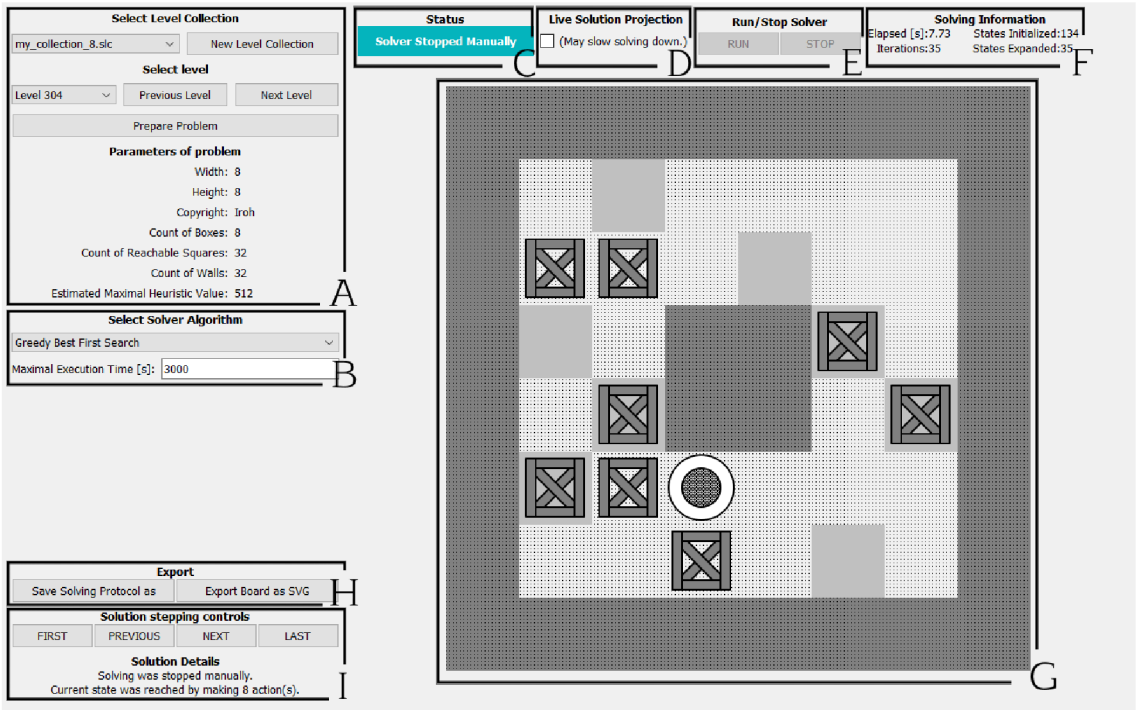
3.1 Grafické uživatelské rozhraní

S využitím knihovny PyQt5 [32] bylo vytvořeno grafické uživatelské rozhraní, jehož hlavní okno má titulek *Sokoban Solver*. Rozhraní umožňuje uživateli pohodlným

způsobem zacházet s implementovanými umělými hráči, volit úlohy k řešení, sledovat průběh řešení a v neposlední řadě výsledky exportovat.

3.1.1 Popis částí rozhraní

Rozhraní lze rozdělit na několik částí, které jsou v následujícím důkladně popsány. Vyznačení jednotlivých oblastí je na obrázku 16. Detaily některých částí jsou pak zobrazeny také na samostatných obrázcích u popisu příslušné části.



A: Volba instance úlohy a její vlastnosti, B: volba umělého hráče a nastavení jeho parametrů, C: zobrazení statusu rozhraní, D: nastavení řešitele, E: ovládací tlačítka řešitele, F: živé informace řešitele, G: hrací plocha hry Sokoban, H: souborové exporty rozhraní, I: procházení nalezeným řešením.

Obr. 16: Snímek grafického uživatelského rozhraní s vyznačením jednotlivých částí.

Volba instance úlohy a její vlastnosti

Oblast je složena ze tří podoblastí vymezených centrovaným nadpisem.

První podoblast obsahuje *Combo Box*, kterým uživatel volí kolekci Sokoban úloh z těch, které jsou ve hře zabudovány. Tlačítko „New Level Collection“ otevře souborový dialog. Uživatel tak může do hry zabudovat vlastní kolekci úloh hry Sokoban ve formátu *.slc*[1], což je standardní formát používaný pro ukládání kolekcí úloh hry Sokoban s XML strukturou.

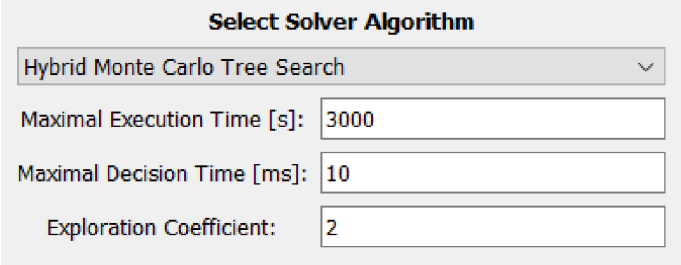
[1] „slc“ je zkratkou pro *Sokoban Level Collection*.

Druhá podoblast obsahuje *Combo Box*, kterým uživatel volí konkrétní úlohu ze zvolené kolekce. Tlačítky „Previous Level“ a „Next Level“ může uživatel mezi jednotlivými úlohami v kolekci přepínat. Tlačítko „Prepare Problem“ je uživatelem stisknuto ve chvíli, kdy už vybral úlohu k řešení. Je jím spuštěna procedura, která vytvoří instanci problému prohledávání stavového prostoru. Úloha je tímto připravena k řešení umělým hráčem.

Třetí podoblast pak jen zobrazuje parametry zvolené úlohy. Je-li úloha nepřipravena, zobrazují se pouze základní parametry úlohy dostupné z kolekce úloh. „Width“ a „Height“ odpovídají počtu sloupců a řádků hracího pole a „Copyright“ je většinou jméno autora úlohy. V okamžiku, kdy je úloha připravena k řešení, zobrazí se další parametry úlohy, které mohou uživateli pomoci vytvořit si představu o její obtížnosti. „Count of Boxes“, „Count of Reachable Squares“ a „Count of Walls“ odpovídají počtům krabic, agentem dosažitelných polí a zdí na hracím poli. Parametr „Estimated Maximal Heuristic Value“ je odhadovaná maximální hodnota heuristické funkce, tedy velmi hrubý horní odhad počtu kroků k nalezení řešení úlohy.

Volba umělého hráče a nastavení jeho parametrů

Oblast obsahuje *combo box*, kterým si uživatel volí umělého hráče a *text boxy*, kterými nastavuje jeho parametry. U všech umělých hráčů je potřeba nastavit maximální dovolený čas řešení v sekundách. Detail této oblasti pro příklad řešitele Hybridní Monte Carlo Tree Search je na obrázku 17.



Obr. 17: Volba umělého hráče a nastavení parametrů pro algoritmus H-MCTS.

Hrací plocha hry Sokoban

V této oblasti je zobrazována obrazová podoba stavu. Jednotlivé komponenty jsou popsány v teoretické části této práce na obrázku 2.

Nastavení řešitele

Uživatel má možnost zaškrtnout *check box* „Live Solution Projection“ a bude tak moct sledovat průběh řešení v reálném čase s frekvencí 1 sekunda. Na hrací ploše je vždy zobrazen poslední expandovaný stav úlohy. Zaškrtnutí této volby může zejména u méně výkonných počítačů vést ke zpomalení výpočtu.

Ovládací tlačítka řešitele

Oblast obsahuje tlačítka „RUN“ a „STOP“, pomocí kterých je spuštěn, resp. zastaven algoritmus řešení úlohy.

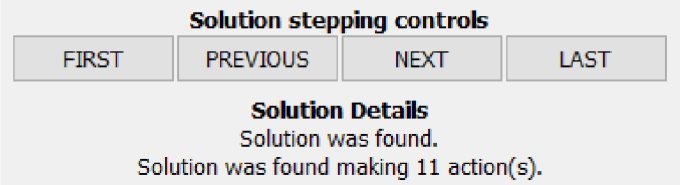
Živé informace řešitele

V této oblasti je možné v reálném čase sledovat aktuální výkon umělého hráče. Konkrétně pak uplynulý čas řešení v sekundách („Elapsed [s]“), počet iterací hlavního cyklu řešitele („Iterations“), počet inicializovaných stavů („States Initialized“) a počet expandovaných stavů („States Expanded“). Hodnoty jsou aktualizovány s frekvencí 1 sekunda.

Procházení nalezeným řešením

V horní polovině oblasti se nachází tlačítka „FIRST“, „PREVIOUS“, „NEXT“ a „LAST“. Jednotlivá ovládací tlačítka umožňují uživateli procházet nalezené řešení přesunem na první, předchozí, další a poslední stav.

Ve spodní polovině oblasti je zobrazena informace o počtu kroků mezi prvním a posledním stavem, které lze procházet. Detail snímku zobrazující celou tuto oblast je na obrázku 18.



Obr. 18: Detail oblasti umožnující procházení nalezeným řešením úlohy.

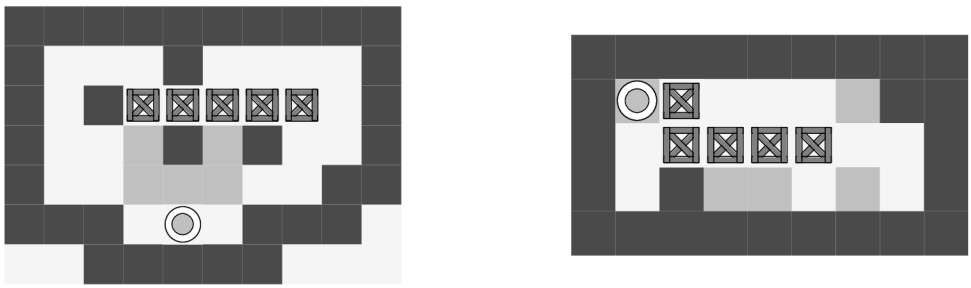
Souborové exporty rozhraní

Rozhraní umožňuje krom procházení nalezeným řešením také možnost dvou souborových výstupů:

- export protokolu řešení pomocí tlačítka „Save Solving Protocol as“, jehož stisknutím je otevřen souborový dialog,
- export grafické podoby hrací plochy v kterékoli fázi řešení pomocí tlačítka „Export Board as SVG“.

Export protokolu řešení umožňuje uživateli vygenerovat textový soubor, ve kterém jsou strukturovaně zaneseny všechny důležité informace o průběhu řešení. Ukázka protokolu řešení je přílohou A této práce.

Uživatel má možnost exportovat ve vektorovém formátu *.svg* grafickou podobu hrací plochy hry Sokoban v kterékoli fázi řešení, ke které je možné se dostat pomocí ovládacích tlačítek řešitele. Ukázky exportu grafické podoby hrací plochy se nachází na obrázku 19 a byly také použity v teoretické části práce.



Obr. 19: Ukázky exportu hrací plochy ve formátu *.svg*.

Zobrazení statusu rozhraní

V této oblasti je zobrazován status, ve kterém se zrovna rozhraní nachází. Zobrazení aktuálního statusu slouží k lepší orientaci uživatele. Statusům rozhraní a tomu, co je uživateli v dané chvíli povoleno, je věnována celá následující podkapitola.

3.1.2 Status rozhraní

Status rozhraní nabývá následujících sedmi různých hodnot.

- *Problém nepřipraven* („Problem not Prepared“). Je-li zobrazen tento status, je nutné vybrat úlohu k řešení a tuto volbu potvrdit tlačítkem „Prepare Problem“. Je také možné zvolit umělého hráče a nastavit jeho parametry.
- *Problém připraven* („Problem ready“). Problém je připraven k řešení a je-li zároveň zvolen umělý hráč, je možné pomocí tlačítka „RUN“ spustit algoritmus a změnit tak status na *Řešení probíhá*. Při změně úlohy se status změní zpět na *Problém nepřipraven*.
- *Řešení probíhá* („Solving“). Je-li tento status aktivní, pak je právě hledáno řešení. V případě, že uživatel zvolil živé sledování průběhu řešení, může jej na hrací ploše sledovat stejně jako informace o výkonu řešitele. Uživatel má možnost řešení manuálně zastavit tlačítkem „STOP“. V závislosti na způsobu terminace algoritmu řešení je aktivní jeden z následujících statusů.
- *Řešení nalezeno* („Solved“). Tento status značí úspěšné vyřešení úlohy zvoleným řešitelem. Je nyní možné mezi stavy úlohy procházet od počátečního stavu úlohy až po nalezený koncový stav.
- *Řešení zastaveno manuálně* („Solver Stopped Manually“). Je-li zobrazen tento status, uživatel ukončil algoritmus řešení stisknutím tlačítka „STOP“. Mezi stavy úlohy je možné procházet stejně jako při statusu „Solved“. Poslední obrazová podoba stavu úlohy neodpovídá žádnému z koncových stavů, ale odpovídá poslednímu expandovanému stavu v okamžiku terminace algoritmu řešení.
- *Řešení nenalezeno, čas vypršel* („Solver Ran Out of Time“). Je-li zobrazen tento status, byl překročen maximální povolený čas pro řešení. Je možné mezi

stavy úlohy procházet stejně jako při statusu „Solved“ a platí totéž co při statusu „Solver Stopped Manually“.

- *Řešení neexistuje* („Solution Does not Exist“). Tento status značí, že byl prohledán celý stavový prostor úlohy, ale řešení nalezeno nebylo. Není možné řešení procházet. Uživatel tak může pouze zvolit jinou úlohu nebo jiného řešitele a připravit nový problém k řešení.

Je-li status úlohy roven jednomu z posledních čtyř výše jmenovaných, může uživatel exportovat protokol o řešení. Obrazovou podobu stavu úlohy může uživatel exportovat při všech statusech rozhraní krom *Řešení probíhá*. Stejně tak má mimo status *Řešení probíhá* uživatel možnost měnit umělého hráče, nebo zvolit jinou úlohu a tu následně připravit k řešení.

3.2 Implementace problému hry Sokoban

Třída stavového problému hry Sokoban obsahuje několik doplňků, které sice nejsou nezbytné pro nalezení řešení úlohy, ale je díky nim hledání zásadně urychleno. V dalším je popsán jejich princip.

3.2.1 Hashování stavů kvůli detekci netriviálních uváznutí

Zásadním problémem pro řešitele hry Sokoban je detekce netriviálních uváznutí, kterým je věnována podkapitola 2.1.4 v teoretické části. Snahou sofistikovaných umělých hráčů je se stavům uváznutí vyhnout. Některé dále zmíněné doplňky umí netriviální uváznutí detekovat. Je žádoucí mít k dispozici jejich seznam, aby bylo možné zabránit zbytečným výpočtovým operacím.

V paměti je tedy vytvořena datová struktura[2], která obsahuje jednoznačné identifikátory stavů netriviálních uváznutí. Generátorem takových identifikátorů je tzv. *hashovací funkce*. Ta přiděluje každému stavu hry Sokoban textový řetězec tak, aby byly splněny podmínky:

- je-li *hash* dvou stavů stejný, pak se jedná o tentýž stav,
- dva různé stavy mají rozdílné *hashe*.

Hashování stavu hry Sokoban je v programu využito nejen při detekci netriviálních uváznutí. Je proto detailně popsáno.

Reprezentací stavu hry Sokoban je datová struktura s_i obsahující nezáporné celé číslo a_i (pozice agenta) a množinu nezáporných celých čísel B_i (množina pozicí krabic). Množina (*set*) je neuspořádaná datová struktura, a proto není možné vytvořit její *hash*. V prostředí programovacího jazyka Python je ale implementována

[2] Zvolenou datovou strukturou pro seznam netriviálních uváznutí je obyčejná množina *set*, což je struktura indexovaná. Ověření toho, zda se daný prvek v množině nachází, tedy není výpočtově náročné.

datová struktura *frozenset* [33] („zamrzlá množina“), která jednoznačnou *hashovací funkcí* h_{fs} disponuje. Platí pak například

$$h_{fs}(\{15, 20, 31\}) = h_{fs}(\{20, 31, 15\}) \tag{17}$$

a je tedy možné definovat hashovací funkci reprezentace stavu hry Sokoban jako

$$hash(s_i) = str(a_i) \oplus \text{'-'} \oplus str(h_{fs}(B_i)), \tag{18}$$

kde operace $\oplus$ odpovídá spojení dvou textových řetězců a funkce *str* převádí celé číslo na textový řetězec.

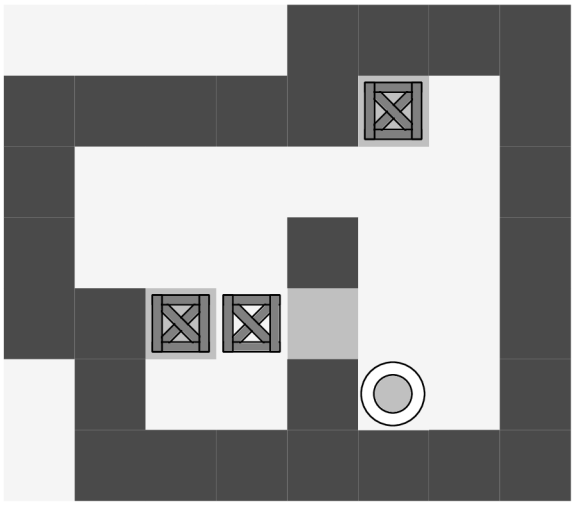
Stav úlohy na obrázku 20 je v programu reprezentován strukturou

$$s_i = (a_i, B_i) = (45, \{13, 34, 35\}) \tag{19}$$

s hashem

$$hash(s_i) = str(a_i) \oplus \text{';'} \oplus str(h_{fs}(B_i)) = \text{'45;8393433636439477075'}. \tag{20}$$

Jedná se o netriviální uváznutí. Identifikátor tohoto stavu by byl přidán do seznamu netriviálních uváznutí.



Obr. 20: Ukázka stavu úlohy.

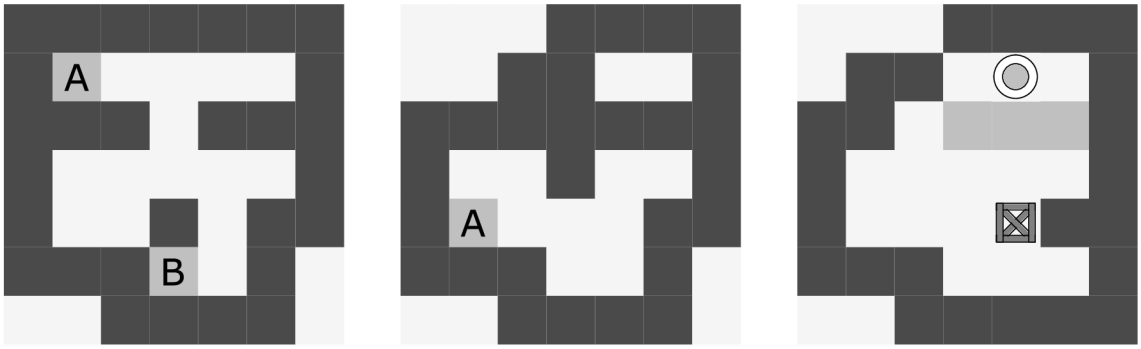
3.2.2 Pomocné podproblémy

V implementaci je hned několikrát využito řešení pomocného podproblému, který je také, stejně jako hledání řešení úlohy hry Sokoban, problémem hledání nejkratší cesty grafem. Je tedy využito modulárnosti celého systému a implementovaní umělí hráči dokážou řešit také následující problémy.

1. Hledání nejkratší cesty mezi bodem A (počáteční stav) a bodem B (koncový stav) mřížky obsahující překážky.

2. Modifikace předchozího tak, že problém neobsahuje žádný koncový stav. Tím sice není nalezeno žádné řešení, ale je docíleno toho, že seznam *CLOSED* obsahuje všechny z bodu A dosažitelné body.
3. Triviální úloha hry Sokoban obsahující pouze jednu krabici a libovolný počet cílových stavů.

Ukázky počátečních stavů těchto úloh jsou na obrázku 21.



(1) (2) (3)

Obr. 21: Ukázky počátečních stavů pomocných podproblémů.

Pomocné podproblémy jsou využity při inicializaci možných přechodů z daného stavu a při výpočtu heuristické funkce. Obojí je dále popsáno.

Hashovací tabulka provedených výpočtů

Při prvních zkouškách výkonnosti umělých hráčů se ukázalo, že pomocné výpočty podproblémů přináší velkou časovou zátěž při hledání řešení hlavního problému. Bylo ale zjištěno, že velká část podproblémů se v průběhu opakuje a jejich řešení je tak hledáno vícekrát.

Do programu byly implementovány tzv. *hashovací tabulky*. Jedná se o datové struktury určené k ukládání dvojic tvořených klíčem (*hashem*) a hodnotou [34]. V prostředí programovacího jazyka Python je vhodnou strukturou pro *hashovací tabulku* datový typ *dict*[3]. Asymptotická náročnost vyhledávacích operací struktury *dict* je konstantní $O(1)$.

Ze vstupních parametrů podproblému je podobným způsobem jako u stavů vytvořen *hash* podproblému. Nachází-li se tento *hash* v tabulce, znamená to, že daný podproblém už byl vyřešen a jeho výsledek je v tabulce uložen. V opačném případě je nutné podproblém vyřešit a dvojici *hash* a výsledek do tabulky uložit.

Bylo provedeno srovnání výkonnosti umělého hráče A* při řešení několika úloh různé složitosti s využitím a bez využití *hashovacích tabulek*. Celkové časy řešení jsou zaznamenány v tabulce 2.

[3] *Dict* je zkratkou pro anglické slovo „dictionary“, což v překladu znamená „slovník“. *Hashovací tabulku* lze tedy chápat jako slovník překládající z jazyka klíčů do jazyka hodnot.

Tab. 2: Srovnání celkového času řešení několika úloh algoritmem A* s využitím a bez využití *hashovacích tabulek*.

ID úlohy	s tabulkami [s]	bez tabulek [s]
6_b_134	17	40
6_b_012	29	61
6_b_001	62	123
6_b_099	93	159
6_b_201	71	182

Z tabulky vyplývá, že použití *hashovacích tabulek* ve všech testovaných úlohách výrazně urychluje celý proces hledání jejich řešení. Procentuální zlepšení je navíc přímo úměrné složitosti úlohy.

3.2.3 Inicializace možných přechodů z daného stavu

Procedura inicializace možných přechodů je v podstatě ověření všech podmínek proveditelnosti přechodu ze stavu $s_i \in D$ definovaných v podkapitole 2.2.3 v teoretické části práce. Tyto podmínky jsou ověřeny v cyklu pro všechny krabice $b_i^k \in B_i$ potlačené do všech čtyř směrů $dir \in \{u, r, d, l\}$.

Během procedury jsou využity pomocné podproblémy. Ještě před samotným ověřováním podmínek proveditelnosti daných přechodů jsou nalezeny všechny pozice dosažitelné agentem. Znalost této množiny je využita pro ověření podmínky (6).

Po úspěšném ověření platnosti podmínek (7) až (10) je nalezena nejkratší cesta z pozice agenta do pozice nutné pro potlačení krabice daným směrem. Pohyby, které agent při cestě do této pozice vykoná, délka této cesty a pozice posouvané krabice jsou pak součástí datové struktury, která je přidána do seznamu všech možných přechodů proveditelných v daném stavu s_i. S tímto seznamem pak pracuje řešitel při expanzi stavu s_i.

Při této inicializaci je možné detekovat netriviální uváznutí. Může se totiž stát, že je seznam po skončení cyklu prázdný. V takovém případě není možné se ze stavu s_i nikam posunout, a protože se nejedná o stav koncový (to by totiž už dříve ukončilo celý algoritmus), jedná se o netriviální uváznutí. *Hash* stavu je uložen do množiny netriviálních uváznutí a hodnota heuristické funkce je změněna na $+\infty$.

3.2.4 Heuristická funkce

Implementování kvalitní heuristické funkce je velkým tématem všech řešitelů hry Sokoban. Heuristika používaná v této práci byla inspirována tou, kterou využívá řešitel *Rolling Stone* [13]. Sami tvůrci ji nazvali *naivní* heuristickou funkcí.

Přípustnost takto implementované heuristické funkce není předpokládána. Experimentální ověření její přípustnosti je součástí Experimentu 1 v podkapitole 4.1.

Definice funkce

Oborem hodnot heuristické funkce h je množina nezáporných reálných čísel $\mathbb{R}^+$, přičemž platí následující dvě podmínky:

$$h(s_i) = 0 \Leftrightarrow s_i \in C, \tag{21}$$

kde C je množina koncových stavů úlohy a

$$h(s_i) = +\infty \Leftrightarrow s_i \text{ je uváznutí.} \tag{22}$$

Obecně pak platí, že čím nižší je hodnota heuristické funkce, tím menší počet přechodů je nutné vykonat k nalezení koncového stavu.

Postup výpočtu

Do výpočtu mimo stav $s_i = (a_i, B_i)$ vstupují také vlastnosti úlohy $Q = (r, c, W, T)$. Před samotným výpočtem hodnoty heuristické funkce h stavu $s_i \in D$ je ověřeno, zda *hash* stavu s_i není v množině netriviálních uváznutí. V takovém případě je výpočet ukončen a platí $h(s_i) = +\infty$.

Je sestavena tabulka H, jejíž řádky odpovídají pozicím krabic $b_i^k \in B_i$ a sloupce odpovídají pozicím cílů $t_j \in T$. Prvek tabulky h_{kj} pak odpovídá nejmenšímu možnému počtu potlačení krabice b_i^k tak, aby se dostala do cílové pozice t_j.

Pro každou dvojici (krabice, cíl) je tedy tato hodnota spočítána jako pomocný podproblém triviální úlohy hry Sokoban obsahující pouze danou krabici a jeden daný cíl. Vstupem podproblému je také pozice agenta. Ostatní krabice jsou považovány za neexistující[4] a pozice zdí jsou stejné jako u hlavního problému. Výsledkem může být buďto nejmenší nutný počet potlačení krabice, nebo $+\infty$ v případě, že je daný cíl krabicí nedosažitelný.

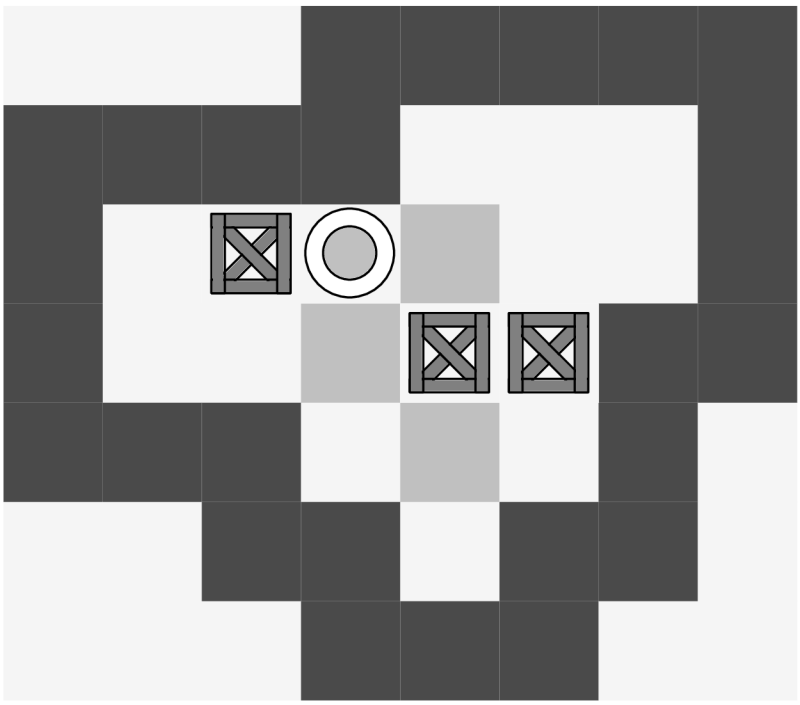
Není nutné počítat celou tabulku znovu pro každý stav, ale stačí vždy upravit tabulku rodičovského stavu. Z té je odstraněn řádek odpovídající původní pozici té krabice, do které bylo při přechodu tlačeno. Dopočítány pak jsou pouze podproblémy s touto krabicí.

Hodnota heuristické funkce je spočítána jako minimum ze součtu vzdáleností všech kombinací dvojic (krabice, cíl). Na tomto místě je tedy hledáno minimum množiny obsahující $t!$ prvků, kde t je počet cílů, resp. krabic. Takovýto výpočet heuristické funkce je na běžných počítačích možné reálně aplikovat na úlohy obsahující maximálně 7 cílů, resp. krabic.

Takto implementovaná heuristika umožňuje detekci netriviálních uváznutí, a to v případech, kdy alespoň jedna krabice nemůže být dotlačena do žádné z cílových pozic, nebo alespoň jeden cíl není dosažitelný žádnou krabicí. V tabulce H se detekované netriviální uváznutí projeví celým řádkem, resp. celým sloupcem naplněným pouze hodnotami $+\infty$.

[4] Právě z důvodu ignorování ostatních krabic je funkce nazývána jako *naivní*.

Princip výpočtu hodnoty heuristické funkce bude ukázán na konkrétním příkladu stavu znázorněného na obrázku 22.



Obr. 22: Stav úlohy pro demonstraci výpočtu hodnoty heuristické funkce.

Reprezentace stavu je

$$s_i = (a_i, B_i) = (19, \{18, 28, 29\}), \tag{23}$$

vlastnosti úlohy jsou

$$Q = (7, 8, W, \{20, 27, 36\}) \tag{24}$$

a tabulka 3 odpovídá tabulce H stavu s_i.

Tab. 3: Tabulka H stavu s_i.

	20	27	36
18	2	4	4
28	1	1	1
29	2	4	4

Hodnota heuristické funkce je pak rovna

$$h((19, \{18, 28, 29\})) = \min\{7, 7, 9, 7, 7, 9\} = 7. \tag{25}$$

3.3 Implementace umělých hráčů

V rámci praktické části práce bylo implementováno celkem šest umělých hráčů.

1. Breadth First Search (BFS),
2. Depth First Search (DFS),
3. Uniform-Cost Search (UCS),
4. Greedy Best First Search (GBFS),
5. A*,
6. Hybridní Monte Carlo Tree Search (H-MCTS).

Algoritmy 1 až 5 patří do skupiny klasických algoritmů a byly implementovány přesně dle popisu v podkapitolách 2.3.1 a 2.3.2 v teoretické části. V dalším je popsána volba datových struktur pro klasické algoritmy a rozdíly v implementaci Hybridního MCTS a klasického MCTS algoritmu.

3.3.1 Vhodné datové struktury

Při implementování a prvních testech výkonnosti umělých hráčů se objevil problém extrémní časové a paměťové náročnosti operací některých algoritmů. V klasických neinformovaných algoritmech je při nakládání s novým expandovaným stavem ověřováno, zda se nachází v jedné ze struktur *OPEN* a *CLOSED*. V klasických informovaných algoritmech je pak k expanzi vybírán ten stav z množiny *OPEN*, jehož hodnotící funkce má nejnižší hodnotu.

Při řešení menších úloh nemusí ani jedna situace činit znatelné potíže. U obtížnějších problémů se ale může stát, že jsou prohledávány struktury, které obsahují až stovky tisíc stavů. Problémem je především to, že struktury postupem algoritmu bobtnají a časová náročnost prohledávacích operací se neustále zvyšuje.

V tabulce 4 je srovnání průměrné asymptotické náročnosti základních operací vybraných datových struktur. Porovnávány jsou operace *contains* (ověření, zda struktura obsahuje daný prvek), *enqueue* (zařazení nového prvku do struktury) a *dequeue* (vyřazení prvku ze struktury dle pravidel[5] dané struktury).

Tab. 4: Srovnání základních operací vybraných datových struktur. [24]

datová struktura	*contains*	*enqueue*	*dequeue*
spojový seznam	$O(n)$	$O(1)$	$O(1)$
binární halda	$O(n)$	$O(logn)$	$O(logn)$
Fibonacciho halda	$O(n)$	$O(1)$	$O(logn)$
indexovaný spojový seznam	$O(1)$	$O(1)$	$O(1)$
indexovaná Fibonacciho halda	$O(1)$	$O(1)$	$O(logn)$

[5] Spojový seznam vrací prvek buďto na začátku nebo na konci. Haldy vracejí minimum.

Na základě informací v tabulce 4 byl pro *CLOSED* ve všech klasických algoritmech zvolen indexovaný spojový seznam. Strukturou *OPEN* u neinformovaných algoritmů je rovněž indexovaný spojový seznam a u algoritmů informovaných je to indexovaná Fibonacciho halda.

Vybrané datové struktury byly v programu implementovány a jejich zdrojové kódy jsou součástí archivu v příloze této práce.

3.3.2 Hybridní Monte Carlo Tree Search

Jakýmsi experimentem v rámci implementace umělých hráčů je využití principů algoritmu Monte Carlo Tree Search a jeho Single Player modifikace pro řešení hry Sokoban. Princip algoritmů MCTS a SP-MCTS je v teoretické části práce v podkapitole 2.3.3.

Hlavním problémem při implementaci MCTS pro řešení hry Sokoban bylo najít způsob, jakým přistoupit k proceduře náhodné simulace a především určení hodnot pro zpětnou propagaci stromem. Hybridní Monte Carlo Tree Search algoritmus je také vybaven mechanismem předcházení cyklů.

Fáze simulace

Byl zaveden parametr maximální hloubky simulace, jehož hodnota byla experimentálně nastavena jako počet krabic úlohy. Náhodná simulace může být ukončena následujícími způsoby.

- Simulace se dostala do stavu uváznutí. Zpětně propagovaná hodnota x je tedy rovna hodnotě heuristické funkce tohoto stavu $x = h(s_i) = +\infty$.
- Simulovaný stav nemá žádné potomky. *Hash* stavu je uložen do množiny netriviálních uváznutí a zpětně propagováno je $x = +\infty$.
- Při simulaci není možné vyhnout se cyklu. Nejedná se o uváznutí, ale zpětně propagováno je $x = +\infty$.
- Při simulaci bylo nalezení řešení úlohy. V tomto případě je ukončen celý algoritmus prohledávání.
- Simulace dosáhne své maximální povolené hloubky, aniž by byla ukončena jedním z důvodů výše. Zpětně propagována hodnota x je rovna hodnotě heuristická funkce posledního simulovaného stavu $x = h(s_i)$.

Normalizace zpětně propagované hodnoty

Obecný MCTS zpětně propaguje hodnoty 1 v případě výhry, 0 v případě remízy a -1 v případě prohry. Simulací nalezená hodnota x určená ke zpětnému propagování je tedy normalizována následující funkcí:

$$x_{norm}(x) = \begin{cases} -1, & \text{jestliže } x = \infty, \\ (1 - \frac{x}{x_{max}})^{10}, & \text{jinak.} \end{cases} \tag{26}$$

Pro případ hry Sokoban odpovídá x_{max} odhadované maximální hodnotě heuristické funkce. Oborem hodnot funkce x_{norm} je množina $\{-1\} \cup [0, 1]$. Hodnotu -1 můžeme považovat za ekvivalent porážky, protože se jedná o uváznutí. Hodnota 0 odpovídá tomu úplně nejhoršímu možnému stavu, který není uváznutím. Hodnota 1 odpovídá pouze koncovému stavu, což je ekvivalent výhry.

Učinění rozhodnutí

Parametr maximálního rozhodovacího času určuje, jak dlouho probíhá iterování jednotlivých fází algoritmu. Po uplynutí tohoto času je na základě selekční strategie, vybrán vybrán jeden z potomků kořenu stromu. Tomu odpovídající akce je učiněným rozhodnutím algoritmu. Vybraný potomek je kořenem pro nové iterování.

Takto se pokračuje až do okamžiku, kdy je buďto při fázích simulace nebo expanze nalezen jeden z cílových stavů úlohy, nebo je prohledán celý stavový prostor úlohy.

4 EXPERIMENTY

V rámci praktické části byly s implementovanými umělými hráči provedeny tři experimenty. Dále používaný pojem *měření* odpovídá aplikaci jednoho ze šesti implementovaných algoritmů na jednu danou úlohu. Všechny exportované protokoly jednotlivých měření v rámci daných experimentů se nachází v souborové příloze této práce ve složce *Experimenty*.

4.1 Experiment 1: Ověření úplnosti a optimálnosti algoritmů

Cílem tohoto experimentu je ověřit teoretické poznatky o optimálnosti řešení nalezených jednotlivými algoritmy a o úplnosti těchto algoritmů. Rovněž by mělo být ověřeno, zda implementovaná heuristická funkce je, nebo není přípustná.

4.1.1 Provedení a výsledky

Všechny algoritmy[1] byly aplikovány na řešení 15 nejnáročnějších úloh z kolekce *MiniCosmos.slc*, jejíž tvůrce je Aymeric du Peloux. Kolekce je zabudována v GUI a obsahuje celkem 40 úloh hry Sokoban.

Tab. 5: Počet kroků vedoucích k nalezení řešení vybraných úloh všemi algoritmy.

ID úlohy	Optimální	BFS	DFS	UCS	GBFS	A*	H-MCTS
MINICOSMOS 26	**48**	**48**	64	**48**	54	54	64
MINICOSMOS 27	**26**	**26**	36	**26**	**26**	**26**	30
MINICOSMOS 28	**38**	**38**	64	**38**	52	52	64
MINICOSMOS 29	**13**	**13**	23	**13**	17	17	15
MINICOSMOS 30	**45**	**45**	129	**45**	53	53	65
MINICOSMOS 31	**14**	**14**	16	**14**	**14**	**14**	16
MINICOSMOS 32	**20**	**20**	24	**20**	**20**	**20**	**20**
MINICOSMOS 33	**19**	**19**	35	**19**	**19**	**19**	31
MINICOSMOS 34	**21**	**21**	41	**21**	**21**	**21**	35
MINICOSMOS 35	**16**	**16**	**16**	**16**	**16**	**16**	20
MINICOSMOS 36	**22**	**22**	30	**22**	**22**	**22**	26
MINICOSMOS 37	**18**	**18**	32	**18**	28	28	20
MINICOSMOS 38	**17**	**17**	49	**17**	19	19	21
MINICOSMOS 39	**21**	**21**	37	**21**	27	27	39
MINICOSMOS 40	**17**	**17**	25	**17**	**17**	**17**	23

[1] Algoritmus H-MCTS není deterministický, což znamená, že se počet kroků může lišit při dvou měřeních téže úlohy. V tabulce je uveden medián ze vzorku 5 měření.

4.1.2 Diskuse

Optimálnost nalezeného řešení byla prokázána pouze u algoritmů BFS a UCS. Z toho plyne, že implementovaná heuristická funkce, kterou využívají informované algoritmy GBFS, A* a H-MCTS, *není přípustná*. *Úplní*, tedy schopni nalézt řešení, pokud existuje, jsou všichni umělí hráči.

4.2 Experiment 2: Srovnání implementovaných umělých hráčů

Cílem tohoto experimentu je vzájemně porovnat výkonnost implementovaných umělých hráčů a z dosažených výsledků vyhodnotit jejich silné a slabé stránky.

4.2.1 Provedení a výsledky

Pro účely tohoto experimentu byla z několika kolekcí vytvořena jedna speciální. Ta obsahuje celkem 30 úloh vybraných tak, aby velikosti stavového prostoru byly co nejvíce variabilní. Kolekce je zabudovaná v GUI pod názvem *experiment2.slc*.

Z protokolu o měření jsou zaznamenávány počet kroků k nalezení řešení (par_l) a uplynulý čas řešení (par_t). Všechna měření výkonnosti jsou provedena pětkrát. Pro H-MCTS je parametr délky řešení par_l určen jako medián ze všech měření. Časový parametr par_t všech algoritmů je stanoven jako aritmetický průměr ze všech měření.

Pro řešitele hry Sokoban neexistuje žádné obecně používané hodnocení výkonnosti. Na základě parametrů par_l a par_t je tedy vypočítáno bodové ohodnocení (skóre) řešení dané úlohy dle vzorce

$$score = 50 \cdot \frac{l_{opt}}{par_l} + 50 \cdot \frac{t_{min}}{par_t}, \tag{27}$$

kde l_{opt} je délka optimálního řešení úlohy a t_{min} je nejmenší uplynulý čas řešení ze všech provedených měření dané úlohy. Výsledné skóre je na škále od 0 do 100. Umělí hráči jsou mezi sebou porovnání dle průměrné hodnoty skóre ze všech řešených úloh.

Tabulka s podrobnými výsledky tohoto experimentu je v příloze B této práce. V tabulce 6 je porovnáno průměrné skóre všech umělých hráčů v závislosti na velikosti stavového prostoru úlohy.

Tab. 6: Porovnání skóre umělých hráčů v závislosti na velikosti stavového prostoru.

velikost stavového prostoru	BFS	DFS	UCS	GBSF	A*	H-MCTS
nespecifikováno	**62**	**49**	**62**	**79**	**79**	**30**
do 10^5	66	59	65	85	86	38
od 10^5 do 10^6	67	51	65	75	75	23
od 10^6	31	22	41	84	82	19

4.2.2 Diskuse

Nejvyššího skóre dosáhli umělí hráči využívající heuristickou funkci (A* a GBFS). Mezi neinformovanými hráči dopadly shodně algoritmy BFS a UCS. Nejhůře si počínala implementovaná heuristika H-MCTS.

Z tabulky 6 vyplývá, že neinformované metody prohledávání vykazují výrazně horší skóre při řešení úloh s velkým stavovým prostorem. Naopak u klasických informovaných algoritmů nebyl zaznamenán zásadní výkonností výkyv v závislosti na velikosti stavového prostoru. Limitem metod využívajících heuristickou funkci je její časově náročný výpočet.

4.3 Experiment 3: Srovnání s algoritmy jiných autorů

Cílem experimentu je srovnat implementovaný algoritmus A*, který dosáhl v předchozím experimentu nejvyššího skóre, s nejvýznamnějšími algoritmy jiných autorů.

4.3.1 Provedení a výsledky

Pro účely tohoto experimentu bylo vybráno prvních 60 úloh z kolekce *Microban.slc*, jejíž tvůrce je David W. Skinner. Kolekce je zabudována v GUI. Maximální čas řešení jedné úlohy je 30 minut.

Tabulka s výsledky algoritmů jiných autorů při řešení kolekce *Microban.slc* je dostupná na webu *sokobano.de* [35]. V tabulce jsou uvedeny počty kroků vedoucích k řešení všech úloh algoritmy *Takaken*, *YASS*, *JSoko*, *Sokolution* a *Festival*. Tabulka obsahující pouze prvních 60 úloh i s výsledky umělého hráče A* je přílohou C této práce a srovnání algoritmů je v tabulce 7.

Tab. 7: Srovnání průměrného počtu kroků vedoucích k řešení jednotlivých algoritmů.

řešitel	průměrný počet kroků	pořadí
A*	**16,32**	**2**
Takanen	17,08	4
YASS	15,92	1
JSoko	18,32	6
Sokolution	16,85	3
Festival	17,38	5

4.3.2 Diskuse

Umělý hráč A* dokázal v časovém limitu 30 minut vyřešit všech 60 vybraných úloh z kolekce *Microban.slc* a dosáhl při tom lepších výsledků než někteří srovnávaní umělí hráči. Je ale nutné zdůraznit, že řešitelé, se kterými je A* srovnáván, jsou stavěni

na řešení daleko obtížnějších úloh, které by žádný algoritmus implementovaný v této práci nedokázal vyřešit.

5 ZÁVĚR

Cílem práce bylo analyzovat hru Sokoban, možnosti jejího řešení pomocí metod umělé inteligence, dále implementovat umělé hráče hry Sokoban využívající vybrané metody umělé inteligence a nakonec provést ověřovací a srovnávací experimenty.

Teoretická část se věnovala problematice hry Sokoban, prohledávání stavového prostoru a vybraným metodám. V rámci praktické části práce byly implementovány algoritmy a grafické uživatelské rozhraní pro provedení srovnávacích experimentů, kterým byla věnována závěrečná část.

Práce byla zaměřena na klasické neinformované i informované metody prohledávání stavového prostoru a na verzi algoritmu Monte Carlo Tree Search určenou pro hraní her jednoho hráče. Implementováno bylo celkem šest různých umělých hráčů. Bylo vytvořeno grafické uživatelské rozhraní, které usnadňuje práci s kolekcemi úloh a jednotlivými řešiteli. Implementováno bylo několik doplňků jako hashovací tabulky či pomocné výpočty podproblémů, jejichž cílem je urychlit prohledávání stavového prostoru hry Sokoban.

V rámci praktické části byly s implementovanými algoritmy provedeny tři experimenty. První experiment ověřuje úplnost a optimálnost řešení umělých hráčů. Bylo ověřeno, že všechny algoritmy jsou úplné a algoritmy Breadth First Search a Uniform-Cost Search nalézají optimální řešení úlohy, což odpovídá teoretickým poznatkům.

Druhý experiment porovnal implementované umělé hráče dle speciálně vytvořené metriky skóre, která je přímo úměrná rychlosti řešení a nepřímo úměrná délce nalezeného řešení. Nejlépe si v tomto srovnání vedly klasické informované algoritmy A* a Greedy Best First Search. Z neinformovaných algoritmů dosáhly nejvyššího skóre metody Breath First Search a Uniform-Cost Search, a to zejména z toho důvodu, že jimi nalezené řešení je optimální. Algoritmus Depth First Search je sice rychlejší, ale jím nalezené řešení má k optimálnímu daleko. Implementovaná heuristika Hybridní Monte Carlo Tree Search se z vytvořených řešitelů ukázala jako nejhorší a musela by doznat značných vylepšení, aby mohla být k řešení úloh hry Sokoban plnohodnotně využívána. Kostra algoritmu Monte Carlo Tree Search dosahuje skvělých výsledků při hraní více hráčů, k řešení her jednoho hráče se ale příliš nehodí.

Třetí experiment porovnal umělého hráče A* s algoritmy jiných autorů. Výsledky ukázaly, že je s ostatními algoritmy srovnatelný. Zároveň je ale potřeba zdůraznit, že limitem všech implementovaných umělých hráčů v této práci jsou úlohy se sedmi (u neinformovaných spíše méně) krabicemi. Lze ale říct, že při řešení snazších úloh hry Sokoban si implementované algoritmy vedou velmi dobře.

6 SEZNAM POUŽITÉ LITERATURY

[1] Wikipedia contributors: *Sokoban — Wikipedia, The Free Encyclopedia*. 2021, [Online; cit. 2021-04-23].
URL `https://en.wikipedia.org/w/index.php?title=Sokoban&oldid=1019913310`

[2] Spectrum HoloByte: *Sokoban : Free Borrow and Streaming : Internet Archive*. 2015, [Online; cit. 2021-04-23].
URL `https://archive.org/details/msdos_sokoban_1984_spectrum_holobyte`

[3] Project sokoban: *History*. [Online; cit. 2021-04-23].
URL `http://sokobano.de/en/history.php`

[4] Project sokoban: *Sokoban Wiki*. [Online; cit. 2021-04-23].
URL `http://sokobano.de/wiki/index.php?title=Main_Page`

[5] Junghanns, A.; Schaeffer, J.: *Sokoban: Evaluating standard single-agent search techniques in the presence of deadlock*. In *Conference of the Canadian Society for Computational Studies of Intelligence*, Springer, 1998, s. 1–15.

[6] Fryers, M.; Green, M. T.: *Sokoban. Eureka 54*, 1995.

[7] Culberson, J.: *Sokoban is PSPACE-complete*. 1997.

[8] Rei, L.; Teixeira, R.: *Willy: A Sokoban Solving Agent*.

[9] Junghanns, A.: *Pushing the limits: new developments in single-agent search*. University of Alberta, 2000.

[10] Junghanns, A.; Schaeffer, J.: *Single-agent search in the presence of deadlocks*. In *AAAI/IAAI*, 1998, s. 419–425.

[11] Junghanns, A.; Schaeffer, J.: *Sokoban: Improving the search with relevance cuts*. *Theoretical Computer Science*, ročník 252, č. 1-2, 2001: s. 151–175.

[12] Junghanns, A.; Schaeffer, J.: *Sokoban: Enhancing general single-agent search methods using domain knowledge*. *Artificial Intelligence*, ročník 129, č. 1-2, 2001: s. 219–251.

[13] University of Alberta: *Our Program - Rolling Stone*. [Online; cit. 2021-04-23].
URL `http://webdocs.cs.ualberta.ca/~games/Sokoban/program.html`

[14] Pereira, A. G.; Ritt, M. R. P.; Buriol, L. S.: *Finding optimal solutions to Sokoban using instance dependent pattern databases*. In *Sixth Annual Symposium on Combinatorial Search*, 2013.

[15] Pereira, A. G.; Ritt, M.; Buriol, L. S.: *Optimal sokoban solving using pattern databases with specific domain knowledge*. *Artificial Intelligence*, ročník 227, 2015: s. 52–70.

[16] Damgaard, B.; Meger, M.: *Sokoban YASC*.

[17] Sokoban Wiki: *Solver Statistics*. [Online; cit. 2021-04-23].
URL `http://sokobano.de/wiki/index.php?title=Solver_Statistics#XSokoban_Test_Suite`

[18] Shoham, Y.; Schaeffer, J.: *The FESS Algorithm: A Feature Based Approach to Single-Agent Search*. In *2020 IEEE Conference on Games (CoG)*, IEEE, 2020, s. 96–103.

[19] Mařík, V.: *Umělá inteligence*. Praha: Academia, 1993, ISBN 978-80-200-2276-9.

[20] Dvořák, J., Březina, T.: *Algoritmy umělé inteligence*. [Studijní opora].

[21] Šeda, M.: *Teorie grafů*. [Studijní opora].

[22] Yannakakis, G. N.; Togelius, J.: *Artificial intelligence and games*, ročník 2. Springer, 2018.

[23] Russell, S.; Norvig, P.: *Artificial intelligence: a modern approach*. 2002.

[24] Wengrow, J.: *A Common-Sense Guide to Data Structures and Algorithms: Level Up Your Core Programming Skills*. Pragmatic Bookshelf, první vydání, 2017, ISBN 1680502441.

[25] Yang, X.-S.: *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[26] Wikipedia contributors: *AlphaGo — Wikipedia, The Free Encyclopedia*. 2021, [Online; cit. 2021-05-09].
URL `https://en.wikipedia.org/w/index.php?title=AlphaGo&oldid=1021541854`

[27] Uriarte, A.; Ontanón, S.: *Game-tree search over high-level game states in RTS games*. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ročník 10, 2014.

[28] van den Herik, J.; Xu, X.; Ma, Z.; aj.: *Computers and Games (6 conf.)*. Springer, 2008.

[29] Schadd, M. P.; Winands, M. H.; Tak, M. J.; aj.: *Single-player Monte-Carlo tree search for SameGame. Knowledge-Based Systems*, ročník 34, 2012: s. 3–11.

[30] Wikipedia contributors: *Python (programming language) — Wikipedia, The Free Encyclopedia*. 2021, [Online; cit. 2021-05-09].
URL `https://en.wikipedia.org/wiki/Python_(programming_language)`

[31] Wikipedia contributors: *Object-oriented programming — Wikipedia, The Free Encyclopedia*. 2021, [Online; cit. 2021-05-09].
URL `https://en.wikipedia.org/wiki/Object-oriented_programming`

[32] Python Software Foundation: *PyQt5*. [Online; cit. 2021-05-09].
URL `https://pypi.org/project/PyQt5/`

[33] Python Software Foundation: *PyQt5*. [Online; cit. 2021-05-11].
URL `https://pypi.org/project/PyQt5/`

[34] Hordějčuk, V.: *Hashovací tabulka (hash table)*. [Online; cit. 2021-05-11].
URL `http://voho.eu/wiki/datova-struktura-hash-tabulka/`

[35] Wiki, S.: *Solver Statistics - Microban - David W. Skinner*. [Online; cit. 2021-05-14].
URL `http://sokobano.de/wiki/index.php?title=Solver_Statistics_-_Microban_-_David_W._Skinner`

[36] Sedgewick, R.; Wayne, K.: *Algorithms 4th Edition*. 2020, [Online; cit. 2021-05-08].
URL `https://algs4.cs.princeton.edu/home/`

[37] Setiani, R. T.; Indriyono, B. V.: *Implementation of A* Algorithm for Solving Sokoban Logic Games. Journal of Applied Intelligent System*, ročník 4, č. 2, 2019: s. 104–111.

Seznam zkratek a symbolů

Zkratky

BFS	Breadth First Search
DFS	Depth First Search
FESS	Feature Space Search
GBFS	Greedy Best First Search
GUI	Graphic User Interface
H-MCTS	Hybridní Monte Carlo Tree Search
MCTS	Monte Carlo Tree Search
NP	nedeterministický polynomiální
RTS	Real Time Strategy
|SP|	Velikost stavového prostoru
SP-MCTS	Single Player Monte Carlo Tree Search
SVG	Scalable Vector Graphics
UCB	Upper Confidence Bound
UCS	Uniform-Cost Search
XML	Extensible Markup Language
YASS	Yet Another Sokoban Clone

Symboly

a_i	pozice agenta
b	faktor větvení
b_i^k	prvek množiny pozicí krabic
B_i	množina pozicí krabic
c	počet sloupců hracího pole
c_e	explorační koeficient
C	množina cílových stavů
d	velikost stavového prostoru, konečná maximální hloubka stromu
dir	množina směrů pohybu
D	množina stavů
e	hrana grafu
E	množina hran grafu
f	hodnotící funkce
$\phi, \phi_{dir}, \phi_{dir}^{b_i^k}$	operátory
Φ	množina operátorů
g	funkce nejkratší dosud nalezené vzdálenosti mezi stavy
G	orientovaný graf

h	heuristická funkce
h_{fs}	hashovací funkce struktury frozenset
$hash$	hashovací funkce
H	tabulka vzájemných vzdáleností krabic a cílů v heuristické funkci
i, j	uzel/index stavu
l	délka nejkratší cesty
l_{opt}	délka optimálního řešení úlohy
m	konečná maximální hloubka stromu
m_{dir}	argument změny pozice při přechodu
$\mathbb{N}$	množina přirozených čísel
$O()$	O-notace asymptotické náročnosti operací
par_l	počet kroků k nalezení řešení
par_t	uplynulý čas řešení
P	plán pro úlohu
q	počet prvků množiny triviálních uváznutí
q_i, q_j	počet dřívějších výběrů uzlu ve fázi selekce
Q	vlastnosti úlohy
r_a	celkový počet dosažitelných polí agentem
r	počet řádků hracího pole
$\mathbb{R}^+$	množina nezáporných reálných čísel
R_i	množina dosažitelných pozic agentem ve stavu s_i
s_i	stav hry Sokoban
str	funkce převádějící celé číslo na textový řetězec
S	stavový prostor
t	počet cílů úlohy
t_{min}	nejmenší uplynulý čas řešení napříč měřeními
T	množina pozicí cílů
U	úloha nad stavovým prostorem
v_i	uzel grafu
V	množina uzlů grafu
W	množina pozicí zdí
x, x_i	součet zpětně propagovaných odměn
x_{max}	odhadovaná maximální hodnota heuristické funkce
x_{norm}	normalizační funkce zpětné propagace
z	obecné označení pozice na hracím poli

Seznam obrázků

Seznam tabulek

7 SEZNAM PŘÍLOH

A Ukázka protokolu řešení

solving_protocol.txt

```
[begin]          Solver was initialized properly.
[information]    Solver algorithm is CommunicatingSolver: MonteCarlo.
[information]    Solving problem type is SokobanSt.
[information]    Initial state of problem is:
                    ####
                   ##  #
                 ###.  #
                 #   $ #
                 #@ .$##
                 ###  #
                   ####
[information]    Problem has conditional termination.
[information]    Terminal states are defined by following condition: All
                 boxes must be in goal positions.


[started]        Solving algorithm has started finding solution of SokobanSt
                 problem.


[finished]       Solving algorithm has finished finding.
[reason]         Solving algorithm successfully found solution.
[length]         Length of solution is: 11.
[actions]        Terminal state was found making following actions:
                 rururrdLLddrUUldlluRdrruruulDlDurrdLdLUlldR

[performance]    Time elapsed while solving was 0.06546711921691895s.
[performance]    Loop was iterated 4 times.
[performance]    New state was initialized 67 times.
[performance]    States were expanded 57 times.
[end]            Solving protocol ends here.
```

B Podrobné výsledky Experimentu 2

Porovnání délek nalezeného řešení par_l

Level Id	|SP|	BFS	DFS	UCS	GBFS	A*	H-MCTS
Level 1	17920	18	28	18	18	18	26
Level 2	2880	10	30	10	18	18	18
Level 3	3630	13	25	13	25	25	15
Level 4	4125	16	22	16	20	20	22
Level 5	9100	13	33	13	13	13	13
Level 6	27132	29	65	29	37	37	41
Level 7	6006	12	16	12	12	12	14
Level 8	67298	43	147	49	61	61	67
Level 9	168150	49	119	53	59	59	83
Level 10	16445	35	59	35	35	35	45
Level 11	43680	60	154	60	90	90	98
Level 12	179550	59	157	61	87	87	91
Level 13	91800	52	130	54	72	72	76
Level 14	538200	41	273	41	63	63	101
Level 15	40040	18	18	18	20	20	22
Level 16	1000692	25	79	25	25	25	43
Level 17	1581503	8	284	8	8	8	26
Level 18	20357568	-	-	-	14	14	82
Level 19	511632	24	1168	24	38	38	-
Level 20	299880	19	345	19	35	35	45
Level 21	148512	19	43	19	19	19	27
Level 22	948024	24	46	26	30	30	64
Level 23	8584290	-	55	21	21	21	55
Level 24	538356	32	92	32	48	48	56
Level 25	130130	10	14	14	12	12	30
Level 26	66066	26	30	26	28	28	38
Level 27	624036	31	41	31	35	35	-
Level 28	208208	25	35	25	27	27	31
Level 29	135135	32	52	32	52	52	64
Level 30	297024	15	97	15	19	19	23

Porovnání časů řešení par_t

Level Id	|SP|	BFS	DFS	UCS	GBFS	A*	H-MCTS
Level 1	17920	0.39	0.11	0.38	0.06	0.06	1.27
Level 2	2880	0.11	0.19	0.15	0.1	0.11	0.64
Level 3	3630	0.22	0.18	0.25	0.13	0.14	1.31
Level 4	4125	0.4	0.19	0.39	0.27	0.27	3.65
Level 5	9100	0.49	0.29	0.67	0.14	0.14	5.22
Level 6	27132	1.01	0.43	0.94	0.29	0.3	4.05
Level 7	6006	0.21	0.02	0.22	0.03	0.03	0.19
Level 8	67298	9.13	2.38	9.27	0.48	0.46	9.7
Level 9	168150	28.4	1.11	24.73	0.75	0.72	6.85
Level 10	16445	1.3	1.1	1.33	1.24	1.28	29.41
Level 11	43680	5.09	4.38	5.17	3.48	3.47	91.3
Level 12	179550	8.93	5.42	9.14	8.01	7.95	119.15
Level 13	91800	8.04	4.18	8.21	1.65	1.61	82.91
Level 14	538200	35.39	20.42	33.77	8.98	9.01	232.46
Level 15	40040	2.75	1.97	2.73	2.45	2.43	37.45
Level 16	1000692	69.68	1.26	63.49	0.18	0.17	6.2
Level 17	1581503	219.0	12.01	79.02	0.19	0.2	39.26
Level 18	20357568	-	-	-	0.34	0.35	80.35
Level 19	511632	125.02	39.65	113.99	2.04	2.0	-
Level 20	299880	20.6	10.17	21.24	6.32	6.28	199.85
Level 21	148512	13.93	8.5	13.46	4.41	4.31	238.92
Level 22	948024	48.54	8.98	42.47	6.07	6.04	145.97
Level 23	8584290	-	54.78	445.15	9.63	12.18	222.57
Level 24	538356	31.53	5.71	32.49	31.06	31.16	149.74
Level 25	130130	2.96	4.42	3.15	0.39	0.44	112.75
Level 26	66066	3.14	0.8	2.87	0.71	0.71	4.45
Level 27	624036	63.37	43.76	63.86	53.18	53.42	-
Level 28	208208	9.2	4.89	7.85	7.47	7.43	110.71
Level 29	135135	11.69	8.24	12.51	84.12	83.26	298.1
Level 30	297024	27.41	32.53	35.84	12.48	12.29	104.94

Porovnání skóre

Level Id	|SP|	BFS	DFS	UCS	GBFS	A*	H-MCTS
Level 1	17920	56	53	56	87	90	36
Level 2	2880	90	38	79	68	68	34
Level 3	3630	76	59	73	70	67	48
Level 4	4125	71	80	71	70	70	39
Level 5	9100	62	40	59	91	92	51
Level 6	27132	63	54	64	85	84	39
Level 7	6006	52	53	51	60	62	45
Level 8	67298	52	24	46	80	82	34
Level 9	168150	51	52	48	88	90	35
Level 10	16445	91	78	90	93	92	41
Level 11	43680	83	58	83	82	82	32
Level 12	179550	80	68	78	67	67	35
Level 13	91800	60	39	58	84	85	35
Level 14	538200	62	29	63	81	81	22
Level 15	40040	85	99	86	85	85	45
Level 16	1000692	50	22	50	95	97	30
Level 17	1581503	50	2	50	94	92	16
Level 18	20357568	0	0	0	46	45	9
Level 19	511632	51	4	51	80	81	0
Level 20	299880	65	33	64	76	76	23
Level 21	148512	65	47	66	98	99	36
Level 22	948024	56	56	53	85	85	21
Level 23	8584290	0	28	51	99	89	21
Level 24	538356	58	64	58	42	42	30
Level 25	130130	56	40	41	88	83	17
Level 26	66066	60	84	61	92	92	41
Level 27	624036	84	87	84	85	84	0
Level 28	208208	76	84	80	78	78	42
Level 29	135135	84	79	82	36	36	1
Level 30	297024	72	26	67	87	88	38

C Podrobné výsledky Experimentu 3

Porovnání délek nalezeného řešení

ID	A*	Takaken	YASS	JSoko	Sokolution	Festival
Level 1	8	8	8	8	8	8
Level 2	3	3	3	3	3	3
Level 3	13	13	13	13	13	13
Level 4	7	11	7	7	7	13
Level 5	8	6	6	14	10	8
Level 6	29	29	29	29	31	29
Level 7	6	6	6	6	6	6
Level 8	32	34	32	34	32	34
Level 9	10	10	10	10	10	10
Level 10	21	21	21	21	21	21
Level 11	18	16	16	16	18	18
Level 12	11	13	11	13	11	13
Level 13	23	23	21	21	27	23
Level 14	10	10	10	10	10	10
Level 15	14	12	12	12	14	12
Level 16	39	41	39	39	45	41
Level 17	9	9	9	9	9	9
Level 18	13	15	13	13	15	15
Level 19	20	20	20	20	20	20
Level 20	16	16	16	18	16	16
Level 21	5	5	5	5	5	9
Level 22	15	15	15	23	15	15
Level 23	10	10	10	10	10	10
Level 24	9	9	9	13	9	9
Level 25	7	7	7	7	7	7
Level 26	10	14	10	10	14	10
Level 27	10	14	10	10	10	18
Level 28	9	9	9	9	9	9
Level 29	22	22	22	28	22	22
Level 30	5	5	5	5	5	7
Level 31	6	6	6	8	6	6
Level 32	9	9	9	9	9	9
Level 33	10	10	10	10	10	10
Level 34	10	14	8	12	8	22
Level 35	31	37	31	31	31	31

Level 36	59	63	59	127	59	59
Level 37	23	23	23	23	25	23
Level 38	8	8	8	8	12	10
Level 39	27	27	27	27	27	27
Level 40	7	9	7	7	7	11
Level 41	15	15	13	13	13	15
Level 42	17	17	15	17	15	17
Level 43	22	22	22	24	22	24
Level 44	1	1	1	1	1	1
Level 45	11	11	11	11	11	11
Level 46	8	8	8	8	8	8
Level 47	22	22	22	24	26	22
Level 48	14	14	14	14	14	14
Level 49	21	23	21	21	23	21
Level 50	23	21	17	17	19	19
Level 51	8	8	8	10	8	8
Level 52	8	10	8	8	8	10
Level 53	12	12	12	14	20	12
Level 54	30	32	30	34	30	32
Level 55	27	27	27	27	27	27
Level 56	6	10	6	6	6	10
Level 57	23	23	23	23	23	23
Level 58	11	11	11	11	11	13
Level 59	50	50	50	66	56	50
Level 60	48	56	44	52	44	60