MUNI
FI

Fakulta informatiky Masarykovy univerzity

Bot pro hru Gomoku

Dokumentace

Předmět: PA026

Vypracoval: Málek Jan

Obsah

Úvod

Cílem tohoto projektu bylo naprogramovat umělou inteligenci pro hru Gomoku. Hra Gomoku je strategická desková hra, ve které se střetávají dva hráči. Hraje se na desce s názvem goban, což je deska, které původně vznikla pro hru Go. Hraje se s černými a bílými kameny, které se pokládají na průsečíky. Původní deska goban má velikost 18x18 stejných čtvercových polí, ale pro hru Gomoku se používá goban velikosti 15x15. Hráči se tedy střídají v pokládání kamenů jejich barvy na průsečíky a vyhrává ten, kdo jako první vytvoří řadu pěti kamenů v horizontálním, vertikálním nebo diagonálním směru.

Tento projekt je podobný mnoha dalším projektům Gomoku dostupných na platformách jako GitHub, kde jsou často přizpůsobeny pro různé programovací jazyky a technologie. Co tento projekt odlišuje, je integrace grafického uživatelského rozhraní pomocí knihovny Tkinter, která zpřístupňuje hru širšímu spektru uživatelů.

Swap2

Tento projekt se také zaměřuje na různé možnosti rozehrání. Hraním hry Gomoku se zjistilo, že má začínající hráč výhodu. Proto vznikly různé typy rozehrání, jako je třeba SWAP2, které tuto výhodu eliminují. Toto rozehrání se také od roku 2009 používá pro mistrovství světa ve hře Gomoku, aby se dosáhlo maximální vyrovnanosti.

Rozehrání SWAP2 spočívá v tom, že začínající hráč položí na desku dva černé kameny a jeden bílí. Protihráč má potom na výběr ze tří možností:

1. Hrát za bílého (právě na tahu)
2. Hrát za černého
3. Zahrát ještě jeden bílí a jeden černý kámen a možnost výběru nechat na začínajícím hráči.

Při testování mého projektu jsem testoval normální hru i SWAP2, abychom mohli rozdíly ve vyváženosti hry porovnat (viz. Testování).

Algoritmus a teorie

Minimax

Minimax je algoritmus, používaný pro hraní strategických her mezi dvěma a více hráči. Principem algoritmu je procházení stromu hry a minimalizace maximálních možných ztrát. Algoritmus bývá základem většiny počítačových programů pro hraní her, jako jsou piškvorky, dáma nebo šachy.

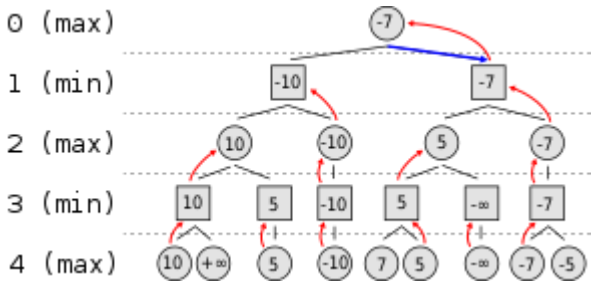
Typickým úkolem je nalézt nejlepší tah v dané pozici. Předpokladem je existence nějaké funkce, která je schopna obodovat libovolnou pozici. Této funkci se říká heuristická funkce. V našem případě se používá taková heuristická funkce, která pro každý kámen hráče spočítá součet všech nastavených bonusů:

Pro zápis výpočtů si zavedeme pár proměnných:
x = souřadnice kamene na ose x
y = souřadnice kamene na ose y
b_size = velikost herní plochy (pro nás 15)

1. Bonus za blízkost ostatních kamenů stejné barvy.
   Výpočet: *1 pro každý kámen do vzdálenosti 1*
2. Bonus za blízkost ke středu herní plochy
   Výpočet: *2^((x – (b_size / 2)) + (y – (b_size / 2))*
3. Bonus za každou trojici, které je kámen součástí
   Výpočet: *1 pro každou trojici*

Toto skóre (součet všech zmíněných bonusů) potom udává, jak moc je daný tah přínosný pro daného hráče.

Algoritmus tedy projde všechny možné tahy, provede obodování vzniklých pozic a vybere ten tah, který přinese nejvýhodnější pozici. Procházení všech možností je velmi časově náročné, proto aby algoritmus skončil v rozumném čase, musí být ohodnocovací funkce velmi rychlá.



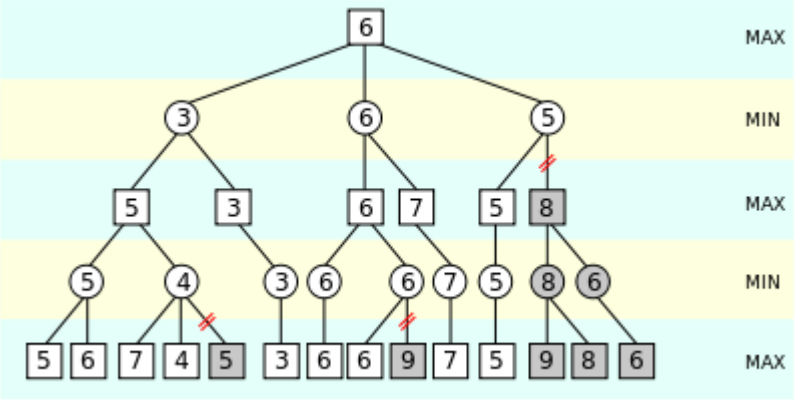
Obrázek 1: Příklad průběhu algoritmu Minimax

Alpha-beta pruning

Alpha-beta pruning je vylepšení algoritmu minimax, které zrychluje jeho běh. Je založeno na pozorování, že pokud právě zpracovávaný půl tah už nemůže obstát v konkurenci s jiným, nemusíme dál prohledávat jeho důsledky.

Metoda byla pojmenována alfa-beta, protože v ní rozšiřujeme původní minimaxový algoritmus o dvě další hodnoty alfa a beta, které nám určují potřebné meze. Z tohoto pohledu jde o algoritmus používající metodu větví a mezí, kde meze jsou dvě, každá pro jednoho hráče.

Účinnost ořezání nepotřebných větví lze zvýšit vhodnou volbou pořadí, v němž jsou procházeny. Pokud budeme mít při volbě pořadí procházení tahů smůlu, alfa-beta ořezávání běh nezrychlí vůbec. Naopak, při výběru optimálního tahu na začátku se dosažená hloubka prohledávání (při stejném čase) zdvojnásobí.



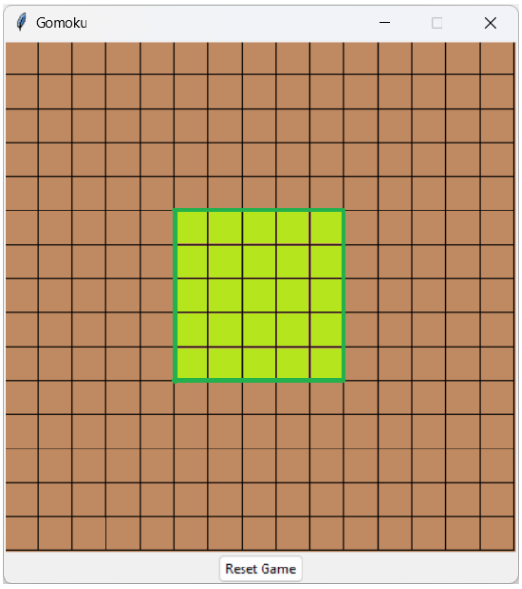
Obrázek 2: Příklad průběhu algoritmu Alpha-beta pruning

Implementace

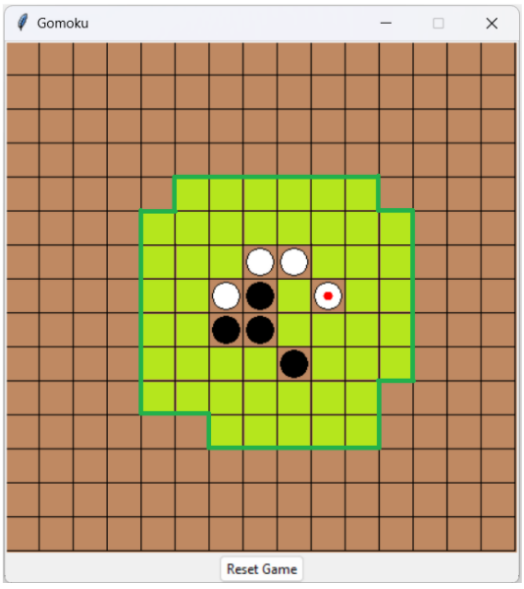
V mé implementaci využívám algoritmus Alpha-beta pruning. Jako heuristickou funkci používám již zmíněný součet bonusů pro každý kámen hráče.

Při psaní tohoto algoritmu, jsem přemýšlel, jak co nejvíce zkrátit čas, který mému programu zabere zahrání jednoho tahu. Došlo mi, že na začátku hned nemusíme spouštět časově náročný algoritmus, protože můžou být nějaké tahy, které je nutné zahrát přednostně. Bot tedy zkontroluje, jestli neexistuje nějaký tah, ve kterém by nemohl ihned vyhrát. Dále je potřeba kontrolovat určité paterny, které nám mohou zajistit jistou výhru, jako jsou například _x_xx_ nebo _xxx_. Pokud se nenašel žádný takto výhodný tah, tak se ještě podíváme, jestli nějaký takovýto tah není pro našeho soupeře. Pokud ano, tak tento tah musíme ihned začít bránit.

Dále jsem také pozoroval, že procházení celé herní plochy (15x15 = 225 polí) je i u malé hloubky prohledávacího algoritmu příliš časově náročné. Jen kdybychom chtěli procházet všechny možné tahy do hloubky 3 (3 mé tahy, 2 soupeřovy), museli bychom projít okolo 500 miliard možných tahů a u každého vyhodnotit jeho přínos. Proto jsem zavedl dynamickou množinu tahů, které se dají považovat za relevantní. Na začátku jde jen o malou plochu uprostřed herní plochy a při každém přidáním nějakého kamene se přidá do této množiny jeho okolí. Všechny výpočty poté pracují s touto množinou tahů. Tímto způsobem máma mnohem menší množství tahů, které nám stačí procházet. A i když se nám velikost naší množiny bude v průběhu hraní zvětšovat, bude se zároveň zmenšovat o tahy zahrané právě v této množině tahů.



Obrázek 3: Množina relevantních tahů na začátku hry

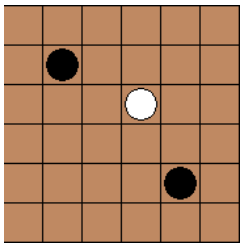


Obrázek 4: Množina relevantních tahů v průběhu hry

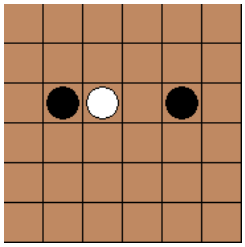
Rozehrání

Při normální hře začíná vždy hráč. Je to z toho důvodu, abychom mohli pozorovat zlepšení našeho AI ve hře kdy je v nevýhodě (normální) a ve hře která by měla být vyrovnaná (SWAP2).

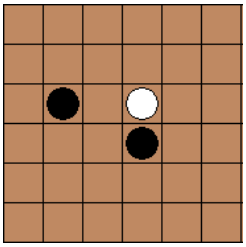
Při SWAP2 rozehrává vždy AI tím způsobem, že si náhodně vybere z předem zadaných rozložení kamenů. Všechny tyto rozložení byly vybrány tak, aby byly pro obě strany zajímavé a vyrovnané. Tato množina není moc velká (jen 8 možností), ale pro základní hraní to bude stačit. Zde jsou nějaké příklady rozehrání, které vybralo AI:



Obrázek 5: Rozehrání 1



Obrázek 6: Rozehrání 2



Obrázek 7: Rozehrání 2

Instalace a použití

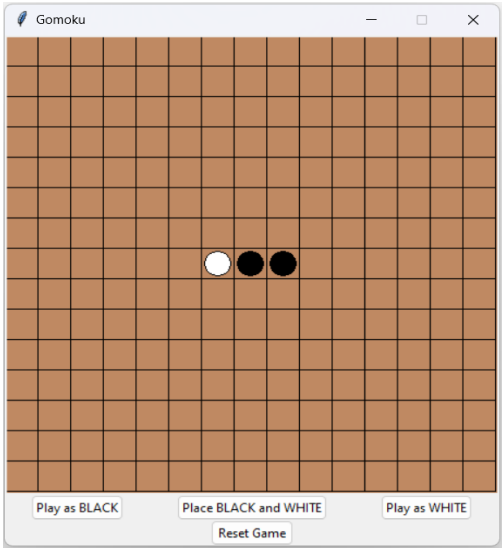
Softwarové požadavky:

- Python 3.6 nebo novější
- Knihovna Tkinter (standardní součást Pythonu)

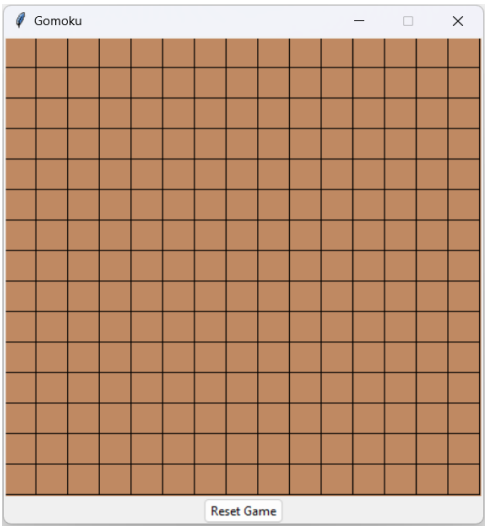
Projekt můžete najít a stáhnou na mém Githubu (https://github.com/JanMalek03/Gomoku). Pro spuštění budete potřebovat mít nainstalovaný Python (https://www.python.org/downloads/). Ve složce s projektem pak stačí do příkazového řádku napsat: *python Gomoku.py*. Pro přepínání rozehrání (viz. SWAP2) není udělané žádné GUI, lze ho ale lehce změnit v kódu. V souboru GomokuGUI.py stačí na 8. řádku přepsat proměnnou SWAP2 na False, pokud rozehrání SWAP2 nechcete.

Příklady běhu

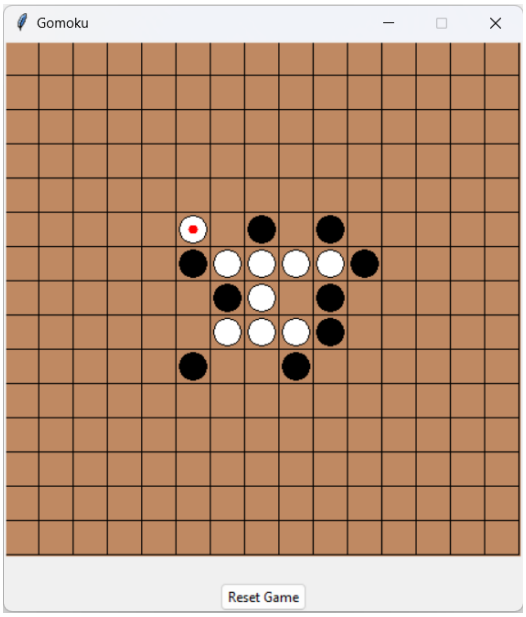
Projekt obsahuje grafické rozhraní, kde uživatelé mohou vidět hrací pole a interaktivně umisťovat kameny. Po spuštění hry se zobrazí okno s hracím polem, tlačítky pro novou hru a (pouze u SWAP2) možnostmi pro výběr barvy kamenů. Pro přehlednost je poslední zahraný tah zvýrazněný a při výhře se zabarví výherní pětice.



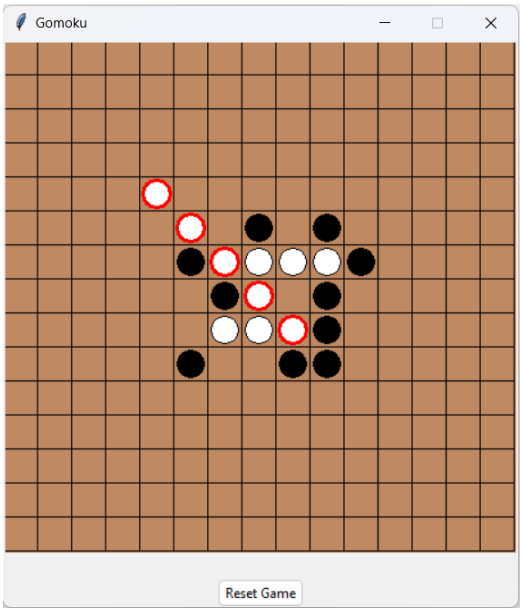
Obrázek 8: Rozehrání SWAP2



Obrázek 9: Normální rozehrání



Obrázek 10: Příklad průběhu hry



Obrázek 11: Příklad výhry

Testování

Umělá inteligence byla testována proti lidem. Každý hráč si zahrál určitý počet her bez rozehrání SWAP2 a poté stejný počet s rozehráním SWAP2. V tabulce níže můžete vidět záznamy z jednotlivých her

	NORMAL			SWAP2		
Jméno hráče	Počet výher AI	Počet výher Hráč	Her celkem	Počet výher AI	Počet výher Hráč	Her celkem
VN	5	10	15	9	6	15
FialKate	15	0	15	13	2	15
Myslivir	14	6	20	16	4	20
Majda	5	10	15	10	5	15
Hana	5	10	15	7	8	15
Kuba	4	1	5	5	0	5
Verca	3	3	6	5	1	6
Katka	5	0	5	5	0	5
Nela	2	8	10	3	7	10
SM	1	1	2	2	0	2
Martin	8	2	10	9	1	10
AAAAAA	5	15	20	8	12	20

Tabulka 1: Záznam her

Výsledky testování

V tabulce s výsledky můžete vidět, jak si AI vedlo celkově. Dohromady jsem sehnal výsledky 276 her od 12ti lidí. V řádku *Úspěšnost AI (%)* můžete vidět, jaké procento her proti hráči vyhrálo AI. To je u normální hry 52 % a u vyrovnaného rozehrání je to 67 %. Celkově se tedy za tyto testy dostalo na úspěšnost 60 %. Můžeme tedy u rozehrání SWAP2 pozorovat značné zlepšení, což jen poukazuje na to, o kolik asi je začínající hráč ve výhodě.

Výsledky	NORMAL	SWAP2	TOTAL
Celkový počet her	138	138	276
Počet výher AI	72	92	164
Počet výher Hráč	66	46	112
Úspěšnost AI (%)	52,17	66,67	59,42
SWAP2 zlepšení (%)	27,78		

Tabulka 2: Výsledky

Pozorování v průběhu z testování

Během testů jsem si všiml určitých slabých míst mé implementace. Proti slabším hráčům bot neměl problém téměř stále vyhrávat, ale u zkušených hráčů měl problém ve vícenásobných útocích. Pokud hráč dokázal jedním kamenem zaútočit třeba i dvakrát, tak bot přestal stíhat bránit.

Druhá věc byla ta, že bot nebyl dostatečně agresivní, takže nechával hráči velký prostor na připravení již zmíněných útoků.

Celkově ale bot vyhrál okolo 67 % her při vyrovnaném rozehrání, což se dá označit alespoň za mírně pokročilého hráče.