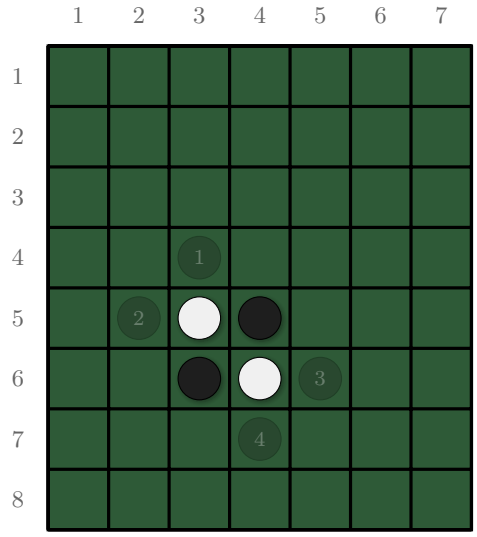
ZADÁNÍ SEMESTRÁLNÍ PRÁCE
Algoritmus Minimax v deskové hře Othello

Zadání

Naprogramujte v jazyce ANSI C přenositelnou[1] **konzolovou aplikaci**, která umožní lidským uživatelům či umělým inteligencím změřit si své síly v deskové hře **Othello** (možný počáteční stav hry je ukázán na obrázku 1). Vámi implementovaná umělá inteligence bude pro zjištění optimálního tahu využívat algoritmus **Minimax** rozšířený o tzv. **Alfa-beta prořezávání**. Jednotlivé hry bude možné uložit a případně později načíst v definovaném formátu do textového souboru. Podoba konzolového rozhraní je plně ve vaší kompetenci.



Obrázek 1: Možný počáteční stav hry Othello. První na tahu je vždy hráč s černými kameny. Možné tahy jsou naznačeny průhlednými kameny a *očíslovány postupně podle čísla řádku a sloupce.*

Program bude spouštěn příkazem `othello.exe`[2] s následujícím seznamem *nepovinných* argumentů – výrazy v lomených závorkách (`<>`), resp. hranatých závorkách (`[]`) označují povinné, resp. nepovinné argumenty (příklad spuštění programu je uveden na ukázce konzolového výstupu 1):

`-pb <desc>` Popis hráče s černými kameny. Povinný argument `<desc>` popisující hráče má strukturu `"<human/minimax>[,name][,d]"`, kde `human`, resp. `minimax` označuje lidského hráče, resp. umělou inteligenci (výchozí: `human`), `name` představuje jméno hráče ve hře (výchozí: `"Anna"`, případně `"Karel"`), nakonec výraz `d` určuje hloubku rozhodovacího stromu vytvářeného algoritmem Minimax (výchozí: `3`). Speciálně v případě hloubky stromu `d` = 0 algoritmus vždy vybere možný tah s postupně nejmenším číslem řádku a sloupce (na obrázku 1 by tedy zahrál možný tah označený číslem 1). V případě lidského hráče bude hodnota `d` ignorována. Pokud je popis hráče

[1] Je třeba, aby bylo možné váš program přeložit a spustit na PC s operačním prostředím Win32/64 (tj. operační systémy Microsoft Windows NT/2000/XP/Vista/7/8/10/11) a s běžnými distribucemi Linuxu (např. Ubuntu, Debian, Red Hat, atp.). Server, na který budete vaši práci odevzdávat a který ji otestuje, má nainstalovaný operační systém Debian GNU/Linux 11 (bullseye) s jádrem verze 5.10.0-20-amd64 a s překladačem gcc 10.2.1.

[2] Přípona `.exe` je povinná i při sestavení pro Linuxu, zejm. při automatické kontrole validačním systémem.

zadán nesprávně, tj. hráč není `human` ani `minimax`, jméno není unikátní, či hloubka stromu není větší nebo rovna nule, program vypíše chybové hlášení `"Invalid player description!\n"` a vrátí hodnotu 1.

`-pw <desc>` Inicializace hráče s bílými kameny. Obdobně jako v případě přepínače `-pb`.

`-size <r> <c>` Argumenty `r` a `c` označují počet řádků a sloupců hracího pole. Hra na obrázku 1 byla inicializována s hodnotami `r` = 8 a `c` = 7. Výchozí hodnoty jsou `r` = 8 a `c` = 8. Pokud hodnoty argumentů `r` a `c` nejsou ostře větší než 1 (není kam umístit počáteční kameny), program vypíše chybové hlášení `"Invalid board size!\n"` a vrátí hodnotu 2.

`-init <ir> <ic>` Na počátku hry jsou na hrací pole umístěny 4 kameny ve schématu ukázaném na obrázku 1. Argumenty `ir` a `ic` poté označují pozici prvního kamene. V případě hry na obrázku 1 byly hodnoty nastaveny `ir` = 5 a `ic` = 3. Výchozí hodnoty jsou `ir` = 4 a `ic` = 4. Pokud není možné kameny na uvedenou pozici umístit, program vypíše chybovou hlášku `"Unable to place initial stones!\n"` a vrátí hodnotu 3.

`-o <gamefile>` Do souboru `gamefile` bude v předepsaném formátu uložen aktuální stav hry. Uložení bude provedeno automaticky na konci hry (když je znám vítěz hry) nebo v jejím průběhu zadáním příkazu `"save"`. Do souboru budou ukládány pouze akce hráčů, tj. speciální příkazy interpretu `"save"` a `"quit"` budou ignorovány. Pokud umístění souboru `gamefile` není zapisovatelné, program vypíše chybové hlášení `"Invalid output file destination!\n"` a vrátí návratovou hodnotu `4`. Výchozí hodnotou je prázdný řetězec.

`-i <gamefile>` Hra může být inicializována pomocí vstupního souboru `gamefile`, který má totožnou strukturu jako v případě přepínače `-o`. Popis hry uvedený v souboru `gamefile` může být přepsán argumenty `-pb`, `-pw`, `-size` nebo `-init`. Pokud vstupní soubor `gamefile` nelze číst, program vypíše chybové hlášení `"Invalid input file!\n"` a vrátí návratovou hodnotu `5`. Výchozí hodnotou je prázdný řetězec.

Vámi vyvinutý program tedy bude vykonávat následující činnosti:

1. Při spuštění bez přepínače `-i <gamefile>` bude hra inicializována argumenty `-pb`, `-pw`, `-size` a `-init` a dále pokračovat v *interaktivním režimu*, tj. postupně čekat na vstupy obou hráčů. Zadané příkazy vyhodnotí a bude vyžadovat další, dokud nebude hra dokončena nebo nebude uveden příkaz `"quit"`. Při zadání chybného výrazu bude vypsána odpovídající chybová hláška a hráč bude vyzván k opětovnému zadání vstupu.

2. Při spuštění s přepínačem `-i <gamefile>` bude hra načtena ze souboru `<gamefile>`. V souboru uvedené inicializační hodnoty je možné přepsat přepínači `-pb`, `-pw`, `-size` nebo `-init`. Součástí vstupního souboru bude i seznam akcí obou hráčů a příkazů interpretu, které budou dávkově zpracovány. Pokud během zpracování příkazů nedojde k chybě nebo k ukončení hry (dokončením hry nebo uvedením příkazu `"quit"`), přejde aplikace do interaktivního režimu, ve kterém budou hráči moci hru dokončit. Pokud během zpracování příkazů dojde k chybě, program vypíše odpovídající chybovou hlášku a vrátí dedikovaný chybový kód.

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, **Makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný `Makefile` a pro Windows `Makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX (LATEX). Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Specifikace vyhodnocovaných výrazů

Výrazem může být pouze příkaz interpretu, nebo akce hráče. Interpret je **case-insesitive**, tj. nerozlišuje velká a malá písmena (zadané výrazy budou převedeny na malá písmena).

Použití příkazů interpretu nebude zapsáno do výstupního souboru zadaného přepínačem `-o`. Příkazy ovšem budou vyhodnoceny, pokud se vyskytnou v souboru vstupním, který je dán přepínačem `-i`. Interpret bude pracovat s následující minimální množinou podporovaných příkazů:

`save` Pokud je zadán výstupní soubor (přepínač `-o`), uloží do něj aktuální stav hry. V opačném případě neprovede žádnou akci. Pokud se uložení stavu hry do výstupního souboru nepodaří, program vypíše chybové hlášení `"Invalid output file destination!\n"` a v případě režimu dávkového zpracování vrátí s návratovou hodnotou `4`.

`quit` Ukončí program s návratovou hodnotou `EXIT_SUCCESS`.

Hráči se během hry střídají (začíná hráč s černými kameny) v zadání svých akcí, které postupně mění stav hry. Akce hráčů budou nabývat následujících hodnot:

`<r>_<c>` Hráč umisťuje svůj kámen na pole v řádku `r` a sloupci `c`. V případě, že se jedná o neplatný tah, program vypíše chybové hlášení `"Invalid move!\n"`. Pokud k neplatnému tahu dojde navíc při načítání hry ze souboru, program skončí s návratovou hodnotou `10`.

`pass` Ve hře může nastat situace, kdy hráči nezbývají žádné možné validní tahy. V takovém případě zahraje tzv. *pass*. Pokud ale validní tah existuje, pak musí táhnout, i kdyby to pro něj bylo nevýhodné. Na neplatném využití této akce reaguje program stejným způsobem jako v případě `<r>_<c>`.

Specifikace formátu vstupních a výstupních souborů

Vstupní a výstupní soubory budou mít stejnou strukturu. V první části souboru budou v blocích (hranatých závorkách) uvedeny hodnoty inicializačních argumentů `-pb`, `-pw`, `-size` nebo `-init`. Bezprostředně po bloku `"[game]"` bude následovat seznam hráči provedených akcí nebo příkazů interpretu. Ukázku vstupního a výstupního souboru můžete vidět na konzolovém rozhraní 2.

Specifikace konzolového rozhraní

Konzolové rozhraní aplikace je plně ve vaší kompetenci a nebude tedy validátorem kontrolováno. Kvalita rozhraní bude zhodnocena 3 body. Například tedy při překreslování hracího pole po každém tahu včetně výpisu aktuálního hráče apod. získáte 3 bodů. Oproti tomu minimální konzolové rozhraní, tj. hra „naslepo“ jenom s příkazovou řádkou bude nutně znamenat nulový bodový zisk.

Vaší fantazii se v tomto případě meze nekladou!

Algoritmus Minimax

Vámi implementovaná umělá inteligence bude pro určení optimálního stavu využívat algoritmus Minimax (společně s jeho vylepšením Alfa-beta prořezávání). V rámci detailní analýzy jistě zjistíte, že algoritmus Minimax využívá k ohodnocení konkrétních stavů hry funkci, jejíž výpočet bude v tomto případě

$$\text{eval}(s) = \text{maxstones}(s) - \text{minstones}(s), \tag{1}$$

kde maxstones(s), resp. minstones(s) označuje počet kamenů maximalizujícího, resp. minimalizujícího hráče ve stavu s. V rámci analýzy problému navrhněte lepší hodnotící funkci, nežli je navržená funkce eval(s).

Dbejte na to, aby akce (hrany stromu), které je možné v různých stavech hry provést, byly zleva seřazeny postupně podle čísla řádku a sloupce taženého kamene. V případě, že algoritmus nalezne více stavů maximalizujících hodnotu funkce eval, vaše implementace vždy vybere akci nejvíce vlevo.

Popsaná upřesnění implementovaného algoritmu Minimax demonstruje obrázek 2 na straně 7.

Užitečné techniky a odkazy

Uvedené dokumenty je možné využít při řešení úlohy. Protože se jedná o postupy standardní, lze k nim nalézt velké množství dokumentace:

1. **Desková hra Othello**
   `https://cs.wikipedia.org/wiki/Othello_(desková_hra)`
2. **Algoritmus Minimax**
   `https://cs.wikipedia.org/wiki/Minimax_(algoritmus)`
3. **Alfa-beta prořezávání**
   `https://en.wikipedia.org/wiki/Alpha-beta_pruning`
4. **Algoritmus Minimax v podání Patricka Henryho Winstona**
   `https://www.youtube.com/watch?v=STjW3eH0Cik`

Řešení úlohy je zcela ve vaší kompetenci – zvolte takové algoritmy a techniky, které podle vás nejlépe povedou k cíli. Pokud bude něco nejasného, neváhejte mne kontaktovat.

Varování! Alfa-beta prořezávání je vylepšením algoritmu Minimax, které jej průměrně zrychluje. Semestrální práce Vám tedy validátorem projde i se samotným Minimax algoritmem. Autor takové práce musí ovšem počítat s penalizací ve formě chyby P202, která je uvedena v chybovníku na webových stránkách předmětu.

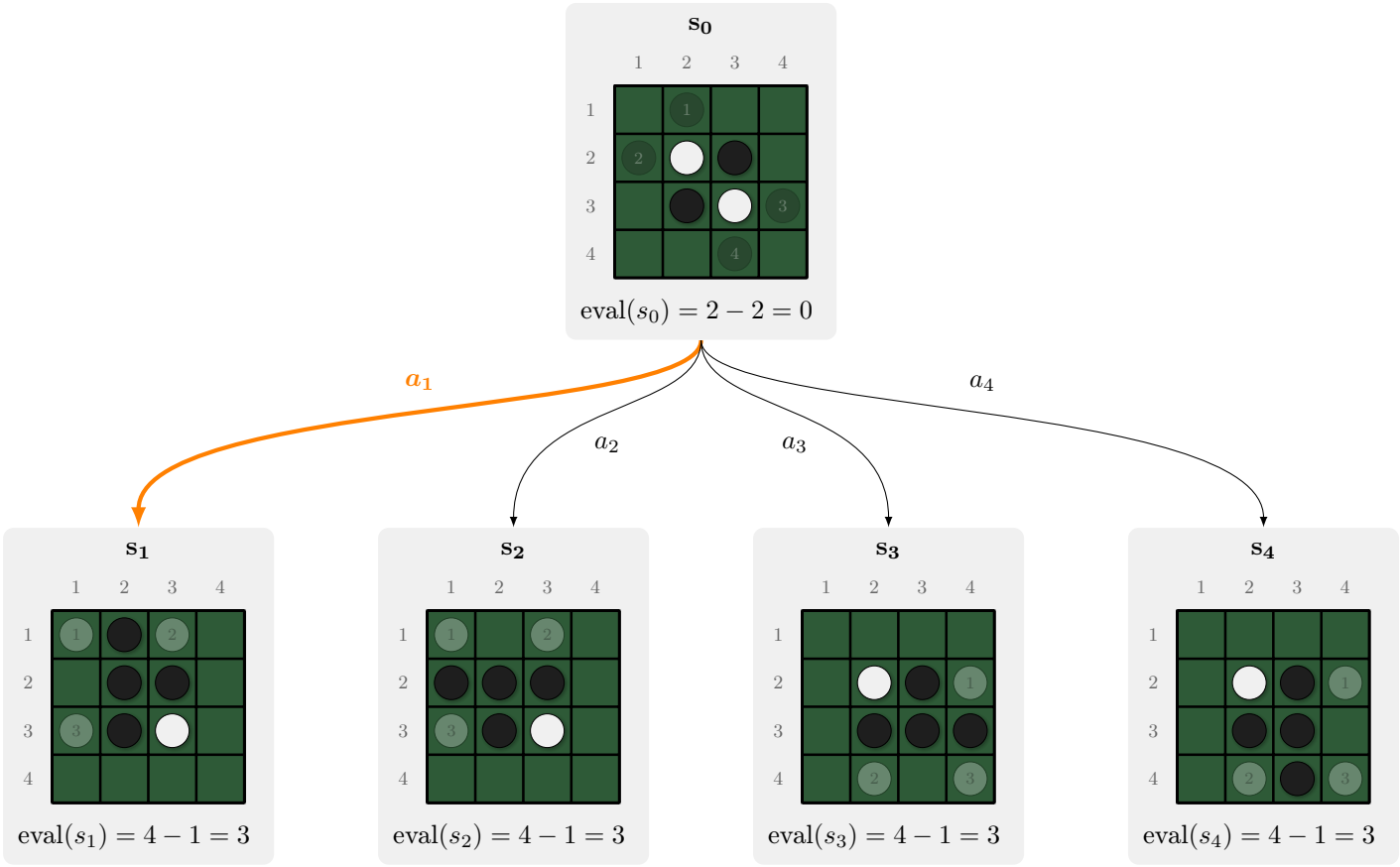
Přílohy

Konzolové rozhraní 1: Ukázka práce programu `othello.exe` v interaktivním a dávkovém režimu. Pro jednoduchost zde není vypisován stav hry, což by bohužel nutně vedlo ke ztrátě 3 bodů.

```
user@machine:~$ cd semestralka && ls
doc src CMakeLists.txt dokumentace.pdf Makefile Makefile.win
user@machine:~/semestralka$ make &>/dev/null && ls
build doc src othello.exe CMakeLists.txt dokumentace.pdf Makefile Makefile.win
user@machine:~/semestralka$ ./othello.exe -pw me_Frantisek
Invalid player description!
user@machine:~/semestralka$ echo $?
1
user@machine:~/semestralka$ ./othello.exe -size 8 8 -init 10 10
Unable to place initial stones!
user@machine:~/semestralka$ echo $?
3
user@machine:~/semestralka$ ./othello.exe -o /validator/interactive_output.txt
(Anna:human)> 5_6
(Karel:human)> pass
Invalid move!
(Karel:human)> pass!!!
Invalid move!
(Karel:human)> pass you little ***
Invalid move!
(Karel:human)> save
(Karel:human)> exit
Invalid move!
(Karel:human)> quit
user@machine:~/semestralka$ echo $?
0
user@machine:~/semestralka$ ./othello.exe -i non_existing_file.txt
Invalid input file!
user@machine:~/semestralka$ echo $?
5
user@machine:~/semestralka$ ./othello.exe -i batch_input_invalid.txt
Invalid move!
user@machine:~/semestralka$ echo $?
10
user@machine:~/semestralka$ ./othello.exe -i batch_input.txt
(Martin:minimax)> 1_2
(Anna:human)> save
Invalid output file destination!
(Anna:human)> quit
user@machine:~/semestralka$ ./othello.exe -i batch_input.txt -o /validator/batc
h_output.txt
(Martin:minimax)> 1_2
(Anna:human)> save
user@machine:~/semestralka$ echo $?
0
user@machine:~/semestralka$
```

Konzolové rozhraní 2: Ukázka vstupních a výstupních souborů použitých v ukázce konzolového rozhraní 1 – konkrétně soubor `interactive_output.txt` vznikl zavoláním příkazu `"save"` na řádku 21.

```
user@machine:~$ cd /validator && ls
batch_input_invalid.txt batch_input.txt batch_output.txt batch_output_valid.txt
interactive_output.txt interactive_output_valid.txt
user@machine:/validator$ cat batch_input_invalid.txt
[size 4 4]
[init 2 2]
[game]
1_1
user@machine:/validator$ cat batch_input.txt
[pb minimax,Martin,1]
[size 4 4]
[init 2 2]
user@machine:/validator$ cat batch_output.txt
[pb minimax,Martin,1]
[pw human,Anna]
[size 4 4]
[init 2 2]
[game]
1_2
user@machine:/validator$ diff gamefile_output.txt gamefile_output_valid.txt &>/d
ev/null
user@machine:/validator$ echo $?
0
user@machine:/validator$ cat interactive_output.txt
[pb human,Anna]
[pw human,Karel]
[size 8 8]
[init 4 4]
[game]
5_6
user@machine:/validator$ diff interactive_output.txt interactive_output_valid.
txt &>/dev/null
user@machine:/validator$ echo $?
0
```



Obrázek 2: Ukázka stanovení optimálního kroku pomocí algoritmu Minimax s hloubkou stromu $d = 1$, tj. díváme se pouze jeden tah dopředu. Hrany rozhodovacího stromu jsou seřazeny postupně podle řádků a sloupců souřadnic umístěných kamenů. Hodnotící funkce $\text{eval}(s)$ zde pro všechny tahy nabývá stejné hodnoty – algoritmus tedy vybere hranu nejvíce vlevo.