Algoritmy použité ve hře *Obrana pevnosti*

1 Strategie počítačového hráče

1.1 Strategie obránce

Obránce musí jednak chránit svou pevnost, ale také musí poskákat co nejvíce útočníků, což je nejjednodušší cesta k jeho výhře. Vzhledem k tomu, že obránci jsou ve hře právě dva, je rozdělení úloh takové, že jeden obránce se snaží zůstávat přímo v pevnosti, aby ji mohl efektivně bránit, a druhý obránce se snaží aktivně skákat útočníkovy kameny mimo pevnost. Strategie obránce spočívá především v rozmístění dvou svých kamenů – jeden v pevnosti a druhý mimo pevnost. Obránce si samozřejmě vybírá takové tahy, aby poskákal útočníkovi co nejvíce kamenů. Snaží se také zabránit útočníkovi, aby se dostal na okraj pevnosti, kde může útočníka hůře přeskočit, a hlavně do rohu pevnosti, kde už ho obránce nemůže přeskočit vůbec. Když už se má útočník dostat do pevnosti, snaží se obránce, aby to bylo středem. Obránce se dále snaží vracet se do pevnosti, pokud mu útočník vystrnadí oba kameny ven z pevnosti. Obránce se také snaží zablokovat útočníka tak, aby žádný útočníkův kámen už nemohl táhnout, a tím pro sebe získat vítězství.

1.2 Strategie útočníka

Úloha útočníka je obsadit pevnost, čehož nejlépe dosáhne tak, pokud se bude pomalu posouvat dopředu a bude si nejvíce chránit své kameny, protože čím více kamenů bude mít, tím menší je šance, že ho obránce může někde skočit. Útočník si tak nejvíce hlídá to, aby mu obránce nebral zbytečně kameny. Útočník se snaží obsadit především rohy a kraje pevnosti, kde je nejmenší (či žádná) šance, že ho obránce přeskočí a útočníktak přijde o své získané pozice. Útočník se zároveň snaží vystrnadit oba obránce pryč z pevnosti, aby ji mohl jednodušeji obsadit. Nejvíce se snaží dostat obránce do nejspodnějších řad hrací desky, odkudto má obránce nejdále zpět do pevnosti. Útočník se zčásti snaží zablokovat obránce, ale vzhledem k pružnosti obránce to není příliš efektivní taktika, takže primární cíl zůstává obsadit pevnost. V případě ohrožení přeskočení se útočník autmaticky kryje dalšími kameny, pokud nějaké má – zkrátka se snaží dosáhnout toho, aby přišel o co nejméně kamenů.

1.3 Obecné poznámky

- V obou algoritmech je použita sůl, tj. ke každému výslednému ohodnocení je ještě přičteno náhodné číslo, které efektivně zabraňuje tomu, aby počítač hrál po každé stejně.

- Algoritmus ohodnocení si hlídá, zda se neopakují stále tytéž pozice na hrací desce, zda nedochází k opakování tahů. Pokud se během 12 kol opakuje určitá pozice na hrací desce více než dvakrát, dojde ke snížení ohodnocení daného tahu. Tím lze snížit riziko vzniku opakovaných tahů, ale ne zcela vyloučit – nemá-li počítač jiný tah, či je ten jiný tah pro něj zásadně horší, stejně pojede tahem, kterým už jel vícekrát.

- Finální řešení, které zabraňuje opakování tahů, je omezený počet tahů. Limit je stanoven na 250 tahů (půltahů). Pokud se překročí tento počet, je hra ukončena s výsledkem *remíza*. V praxi bylo ověřeno, že každá smysluplná hra stejně končí v daleko menším počtu tahů, takže při počtu tahů nad dvě stě už je téměř jisté, že dochází k opakování tahů a počítač nemá jak táhnout jinak.

2 Použité algoritmy

2.1 Generování všech platných tahů

Třída *Rozhodci* obsahuje metodu pro vygenereování všech platných tahů z dané pozice pro daného hráče. Jako první si necháme vygenerovat všechny jednoduché tahy, ve dvou cyklech projedeme všechny pozice na hrací desce (dvoudimenzionální pole) a zjistíme, jestli se na dané políčko můžeme přesunout. Na to máme metody, které rozhodnou, zda posun z jednoho políčka do druhého je validní. Po vygenerování jednoduchých tahů zkontrolujeme, zda je na tahu útočník. Pokud ano, generování končí, útočník může provést pouze jendoduchý tah. Pokud je na tahu obránce, zkontrolujeme, zda se ve vygenerovaných tazích nevyskytuje nějaký skok. Pokud ano, přistoupíme ke druhé fázi generování – musíme najít všechny skoky. Tahy, ve kterých obránce neskáče, zahodíme. Po vygenerování všech skoků (viz další kapitola) vrátíme tento nový seznam paltných tahů. Generování končí.

2.2 Generování všech vícenásobných skoků

Předpoklad je, že už máme seznam nějakých skoků (ty vygenerujeme v první fázi generování tahú, viz předchozí kapitola). Poté postupujeme tak, že si vytvoříme kopii současné hrací desky (vyhneme se tak nepříjemným kolizím) a na tuto kopii provedeme tah (skok) a z této pozice dále generujeme skoky stejným způsobem. Takto rekurzivně projdeme všechny varianty skoků a během toho si program pozice skoků uchovává ve stromové datové struktuře (n-ární strom). Po vygenerování stromu už stačí tento strom zlinearizovat do nějakého seznamu (listu).

2.3 Minimax

Pro generování budoucích tahů je použit klasický algoritmus minimaxu. Jednotlivé pozice jsou ohodnoceny pomocií ohodnocovací funkce a následně je vybrán ten pro danou chvíli nejvýhodnější tah. Alfa-beta ořezávání není implementované, minimax funguje i bez něj dostatečně rychle.

2.4 Obtížnost hry

Obtížnost počítačového hráče je realizována rozdílnou hloubkou při generování možných tahů v Minimaxu. Největší obtížnost má zároveň největší hloubku při generování tahů. Ve třídě `Manazer` je pak v metodě `Tahni` volána metoda `Minimax.NejlepsiTah`, které je jako jeden z argumentů předána hloubka generování. Lehké obtížnosti odpovídá hloubka dva, normální tři a těžké čtyři.