Dokumentace k projektu Gomoku

Jan Sedlák a Aleš Dujíček

28. března 2010

Obsah

1 Úvod

Piškvorky je jedna z nejznámějších her, jejíž kořeny můžeme najít ve starověkém Japonsku. Již od pradávna se mnozí snaží najít co nejlepší výherní taktiku, programování umělé inteligence (AI) pro piškvorky patří po programování AI pro šachy mezi nejoblíbenější úlohy.

Gomoku je aplikace pro hraní piškvorek, která obsahuje myší ovládané grafické prostředí pro hru proti jedné ze dvou AI, hru inteligencí proti sobě navzájem nebo hru člověka proti člověku. Program je napsán pro platformu Linux, ale návrh, struktura použitého jazyka a multiplatformní knihovna SDL zaručují úspěšnou kompilaci i pro jiné platformy (vč. MS Windows).

Tato dokumentace by měla sloužit jak pro jednodušší orientaci v GUI[1] programu, tak pro jednoduché vysvětlení principu fungování v tomto programu obsažených AI.

V sekci 1 se budeme zabývat hrou piškvorky, vysvětlením jejích pravidel a její stručné historii. V sekci 2 se budeme zabývat obsluhou grafického rozhraní programu Gomoku, v sekci 3 budeme popisovat fungování našich AI. V sekci 4 se podíváme na obecné styly řešení AI pro piškvorky. Sekce 5 je sekce programátorská, popíšeme si postup kompilace pro různé platformy. V této sekci také bude stručná charakteristika použitého jazyka a knihovny pro operace s grafikou.

[1] grafické uživatelské rozhraní

2 O hře piškvorky

2.1 Pravidla

Hra piškvorky je jednoduchá hra pro dva hráče. Hraje se za pomocí čtverečkované jednobarevné šachovnice. Každý hráč má své tokeny (většinou křížky a kolečka). Cílem hry je umístit své tokeny na šachovnici tak, aby vodorovně, svisle či diagonálně vznikla neporušená přímá řada pěti hráčových tokenů. Hráči se v umisťování tokenů střídají. Povoleno je samozřejmě umisťování tokenů pouze na volná místa. První, kdo vytvoří neporušenou řadu pěti tokenů, vyhrává.

Existuje samozřejmě několik taktik hry, rozumné je používat vlastní tokeny k blokování soupeřových řad. Častá je také snaha o vytvoření několika konkrétních kombinací, které hráče dostávají do výhodnější pozice.

2.2 Historie

Dnešní podoba piškvorek (angl. Tic-Tac-Toe či Five-in-a-row) se nejspíše vyvinula z oblíbené japonské hry gomoku. V japonském gomoku nejsou tokeny rozlišeny tvarem, ale barvou. Kladou se na křížení čar tvořících šachovnici. Oproti dnešním piškvorkám se dále liší pravidlem, zakazujícím vyhrát pomocí řady delší jak pět tokenů. Její kořeny sahají až k tradiční japonské hře Go, se kterou Gomoku sdílí nejen herní šachovnici (tzv. Goban), ale také některé charakteristiky.

V dnešních dobách vznikají snahy o vytvoření co nejlepšího algoritmu pro hraní piškvorek. Holandský programátor L. Victor Allis naprogramoval algoritmus, který na šachovnici 15*15 zajistí začínajícímu hráči výhru. Algoritmus však bohužel nebyl nikdy zveřejněn. Existují jednoduché matematické důkazy o tom, že na neomezené herní ploše musí existovat neprohrávající strategie pro začínajícího hráče.

3 Obsluha rozhraní programu Gomoku

Program Gomoku má jednoduché GUI, ovládané myší a klávesnicí. Po spuštění programu (např. z konzole příkazem ‘./piskvorky’ vyvolaného z adresáře, ve kterém se nachází program gomoku) se objeví jednoduché okno, ve kterém bude zobrazena prázdná herní šachovnice a dole ve stavovém řádku nápis ‘[F1] AIxAI, [F2] AI1, [F3] AI2, [F4] human x human’. Toto je jednoduché herní menu. Stisknutí příslušné klávesy vyvolá požadovanou akci:

- Stisknutím **F1** se automaticky spustí hra umělé inteligence proti umělé inteligenci.
- Stisknutím **F2** se spustí hra hráče proti první (jednodušší) AI.
- Stisknutím **F3** se spustí hra proti druhé (těžší) AI.
- Stisknutím **F4** začne hra hráče proti hráči.

Při hře AI vs. AI se po automatickém odehrání hra ukončí. Při hře hráče hráč umisťuje svoje tokeny pomocí myši, kliknutím na příslušné místo.

Pokud je ve hře na řadě hráč, může stisknout klávesy ‘q’ pro ukončení hry.

Po ukončení hry se zobrazí ve stavovém řádku nápis ‘vyhraly krizky’, ‘vyhrala kolecka’, ‘hra byla ukoncena’ popřípadě ‘remiza’ a za tímto nápisem jednoduchá nabídka ‘[q]uit [r]estart [m]enu’. Pokud stisknete ‘q’, ukončí se program Gomoku. Pokud stisknete ‘r’, restartuje se právě ukončená hra. Po stisknutí ‘m’ se dostanete zpět do hlavního herního menu.

4 Realizace a algoritmy našich AI

4.1 AI1

Naše první umělá inteligence funguje na opravdu jednoduchých principech. Pro vlastní potřebu jsme ji pokřtili na metodu „Franta hraje piškvorky“. Jedná se o specifickou odnož metody zvané „Algoritmus prohledávání do šířky - hloubky“. Umělá inteligence obsahuje funkci, která zjišťuje pro zadaný token posloupnost o zadané délce. Na začátku svého tahu zkopíruje celé herní pole, poté na každou volnou pozici dá svůj znak a zkontroluje, jestli se nejedná o výherní posloupnost (tím by vytvořila pětici a vyhrála).

Pokud neexistuje žádné takové místo, které by vedlo k výherní posloupnosti, zopakuje totéž, nicméně s nepřítelovým tokenem. Dosazuje nepřítelův token a zkouší, jestli by nepřítel nemohl vyhrát. Pokud ano, zabrání výhře dosazením vlastního tokenu. Algoritmus se opakuje znovu, nejdříve pro zabránění nepřítelových čtveřic, poté trojic a pak se už inteligence soustředí na tvoření co nejdelší vlastní posloupnosti. Jedná se částečně o brute-force metodu. Při hře AI1 nedochází k ohodnocování tahů.

AI1 je kratší a jednodušší inteligence, při soupeření s naší druhou inteligenci většinou prohrává. Mezi její výhody však patří „pozornost“ - AI1 nemůže udělat chybu špatným ohodnocením políčka díky špatně nastavené prioritě (v konečném důsledku nevznikne situace kdy není jasné, proč AI1 položilo token na pozici na kterou ho položilo). Jedná se také o transparentnější AI.

4.2 AI2

Druhá umělá inteligence, kterou jsme vytvořili, pracuje na principu ohodnocování tahů. Pro každé volné políčko vyjádří číselně, jak by byl tah na něj výhodný, a to pro křížky a kolečka zvlášť.

Jak je tah výhodný neovlivňuje jen velikost řady znaků, která by takovýmto tahem vznikla, ale také to, zda je v daném směru dostatek místa pro celou výherní řadu, zda je řada znaků přerušená mezerou a zda je řada z jedné nebo obou stran blokovaná okrajem pole nebo soupeřovým znakem. Všechny tyto informace je pak potřeba převést na jediné číslo. Pro tento účel slouží vzorec $z \cdot (z - m) \cdot (z + 24 \cdot v)$, kde z je počet znaků, které už jsou v řadě, m je počet mezer, které řada znaků obsahuje, a v je počet volných konců řady znaků. Pokud není dostatečný prostor pro výherní posloupnost, pak je ještě číslo násobeno nulou. Vyjímku z tohoto vzorce má situace, kdy je počet znaků o jedna menší než počet výherních a současně není řada přerušena žádnou mezerou. V tomto případě je výhodnost tahu nastavena vysokou konstatou tak, aby tam AI za každých okolností hrála. Tato výjimka je z toho důvodu, že snížení priority tahu tam, kde je znaky z obou stran blokovaná, je příliš vysoké a přestože je takový tah určitě výherní, AI by tam nehrála.

Výhodnost tahu ale nezávisí jen na znacích v jednom směru, ale na celkovém postavení v poli. Je výhodnější hrát tak, aby vznikly dvě trojice znaků, než jen jedna, proto jsou priority tahů ze všech čtyř směrů nakonec sečteny, tento výsledný součet konečně udává, jak je tah výhodný.

Pozice všech tahů s nejvyšší prioritou jsou potom uchovávány v zásobníku, ze kterého je nakonec jeden tah náhodně vybrán.

Tento postup vyhledávání se ukázal jako výhodnější, při hře proti předchozí AI ve většině případů vyhrává a nabízí mnohé další výhody, kterou je třeba možnost procházet možnosti několik tahů dopředu a vyhledávat, jakým tahem by AI později získala výhodu. Toto již ale není součástí řešení.

5 Obecné algoritmy AI pro piškvorky

Zde si popíšeme několik obecných řešení algoritmů umělých inteligencí pro piškvorky.

5.1 Prohledávání do šířky - hloubky

Jedná se o principiálně nejjednodušší algoritmus. Pracuje tak, že vezme množinu všech možných následujících tahů a nalezne-li nejlepší z doposud prohlédnutých, zapamatuje si jej. Jestliže následně nalezne ještě lepší, zapamatuje si tento nový (předchozí dobrý tah zapomene). Projde-li všechny tahy, oznámí tah, který si zapamatoval jako nejlepší. Vzhledem k tomu, že daný algoritmus pracuje jen s množinou právě možných tahů, principiálně nezohledňuje vaši možnou následnou reakci a může tedy provést špatné rozhodnutí a prohrát celou hru. Lidově řečeno, nevidí ani o krok dále.

5.2 Algoritmus Mini-Max

Na rozdíl od předchozího algoritmu nevyhledává Mini-Max nejcennější tah, ale posuzuje tahy i z hlediska možné reakce soupeře, tedy vaši reakce. Proto je vhodné neprověřovat všechny volné pozice-tahy, ale jen ty, které by on sám odehrál v případě použití algoritmu prohledávání do šířky-hloubky. To znamená, že zjistí nejvyšší hodnotu ze všech tahů a následně se pokusí odehrát postupně všechny tahy, které mají tuto nejvyšší cenu (kterých může, ale nemusí být více, než jeden). Při každém simulativně odehraném tahu pak vyzkouší simulativně odehrát i váš tah a zjistit tak úspěšnost, resp. význam či životaschopnost, svého tahu odehraného prvně. Takto zkouší rozehrát hru sám se sebou až do stanovené hloubky.

Jak ale porovnává ceny? Když všechny zkoumané tahy mají shodné ohodnocení? Při stanovené hloubce rovné nule se Mini-Max chová naprosto shodně, jako prohledávání do šířky-hloubky, odehraje tedy tah a hledá nejvyšší cenu. Zjištěnou cenu ale vynásobí -1, takže když ve zkoumané větvi nalezne tah, kde vaše reakce má vysokou cenu, je tato cena (protože je

záporná) ve skutečnosti nižší, ve srovnání s další větví, kde simulování vaší reakce nemá takovou cenu. Jako větev chápejte každé testování z výchozích možných tahů. Algoritmus tedy hledá takovou větev, ve které je jeho zisk vždy co nejvyšší a za vás simulované reakce na ně co nejnižší. Je pochopitelné, že nemá smysl hrát takový tah, který by na vašem místě on sám následně ihned „zarazil“.

Bohužel, algoritmus pracuje rekurzivně a jedinou zarážkou je parametricky zadaná hloubka anebo zjištění, že simulativním odehráním již hra skončila vítězstvím jedno z hráčů. Algoritmus Mini-Max je proto ve svém principu docela pomalý.

5.3 Algoritmus Alfa-Beta prořezávání

Algoritmus Alfa-Beta prořezávání Jedná se o úpravu algoritmu Mini-Max. Oproti Min-Maxu je ale vylepšena o rozhodování, zda-li má ještě smysl pokračovat v propočtech až do stanovené hloubky. Toto rozhodnutí Alfa-Beta učiní na základě zjištění, zda-li by již v tomto simulovaném tahu „zazdil“ předchozí simulovaný tah, ze kterého nyní vychází. Oproti Mini-Max-u tak šetří drahocenný čas. Tohoto systémového zisku pak můžeme využít navýšením hloubky propočtů, aby všechny prověřované větve mohly být propočítávány pečlivěji.

K tomu, aby byla Alfa-Beta schopna rozhodnout, zda-li pokračovat v propočtech do další úrovně hloubky, potřebuje informace o tom, jak dopadl předchozí tah. Je tedy navíc rozšířená o dva parametry, tzv. Alfa a Beta, přičemž v první úrovní (tedy při volání z kořene stromu) je Alfa nastavena na –nekonečno a Beta na nekonečno. Vznikne nám tedy otevřený interval (-∞,∞). Odehraje tedy první simulovaný tah, na něhož zareaguje dalším simulovaným tahem a tak dále až do stanovené hloubky a do hodnoty Alfa uloží cenu výsledné pozice. Takže celou levou větev prozkoumá až do konce. Nastavený interval pak může být (3, ∞). Poté se navrátí o úroveň výše. V tuto chvíli je v bodě n-1 nejlepší zjištěnou cenou hodnota 3. Pak vyzkouší druhý možný tah z této n-1 hlouby zanoření (tedy přejde do hloubky n, ale pro jiný tah), kde zjistí, že výsledná hodnota je třeba lepší a nastaví interval na (4, ∞). Obě hodnoty nyní vynásobí -1, prohodí je na (- ∞,-4) a vrátí se do bodu n-1. Tady ceny porovná tak, že nejprve zjistí, která z níže analyzovaných větví je vyšší. V tuto chvíli to je Alfa=∞. Pak ale zjistí, zda-li interval není záporný, tedy zda-li Beta není menší než Alfa. V případě, že ano, navrátí o úroveň výše hodnotu Beta=-4 (v našem případě), v opačném případě hodnotu Alfa. V úrovní n-2 pak prověří hned další možný tah, ale již mu předá ořízntutý interval (-∞,-4). Tedy nejlepší cenu jako nekonečně malou, ale horní hranici pro zkoumání nastavenu na -4. Pokud zjištěná další zjištěná cena bude vyšší, než -4, nemá smysl dále propočítávat a vrátí se hodnota -4. takto až do uzlu na úrovni n, který je vlastně návratovou hodnotou rekurzivní funkce AlfaBeta.

Je vidět, že na základě předávání předávaní parametrů AlfaBeta se prověřují jen takové tahy, jejichž ceny vždy leží v intervalu od-do zjištěném při předchozím propočtu. Úspora tedy spočívá ve vynechání propočtů takových tahů, které již z dané pozice nemají žádný smysl.

Zdroj

Popis těchto tří metod je vyňat ze seminární práce Tomáše Novosáda
http://www.node.cz/download/XUIMIN_R06631_novosadtomas_sempr.pdf

6 Struktura a členění zdrojových kódů, kompilace

6.1 O použitém jazyce a grafické knihovně SDL

Program byl napsán v programovacím jazyce C a využívá knihovny SDL a SDL_gfx. Programovací jazyk C je jeden z nejoblíbenějších a nejrozšířenějších programovacích jazyků. Používá se pro programování operačních systému i aplikací. Program Gomoku byl kompilován C kompilerem GCC (který je šířen pod svobodnou licencí GNU/GPL). Používá také svobodnou grafickou knihovnu SDL, která byla vyvinuta pro programování počítačových her. SDL poskytuje funkce pro grafický výstup programu a také rozhraní pro používání myši a klávesnice.

6.2 Kompilace

Kompilace (čili překlad zdrojových kódů do výsledného binárního spustitelného souboru) se liší použitím různých kompilerů či různých platforem. Postup pro kompilaci pro Linux je následující: z repozitářů stáhněte balíky obsahující knihovnu SDL, knihovnu SDL_gfx a jejich developer verze (obsahují hlavičkové soubory, většinou označené příponou -devel), C kompiler (například GCC) a pro snadnější kompilaci program make. Kompilaci proveďte jednoduše z terminálu, kdy ve složce se zdrojovými kódy napíšete příkaz ‘make’.

Pro MS Windows je potřeba stáhnout některé IDE (například DEV-C++), dále knihovny SDL a SDL_gfx (možno je i dané knihovny přímo zkompilovat) a poté postupovat jako při kompilaci jakéhokoliv jiného projektu v daném IDE.

6.3 Struktura a rozčlenění programu

Zdrojový kód programu Gomoku je zcela v souladu s principy programování v jazyce C rozčleněn do několika souborů, podle typů funkcí v něm obsažených. Hlavní program je uložen v souboru main.c. Zde se provádí nejnutnější volání funkcí programu, základní obsluha knihovny SDL a volání fcí jednotlivých hráčů nebo inteligencí. Hlavičkové soubory ai1.h, ai2.h a k nim

náležící soubory se zdrojovým kódem obsahují funkce jednotlivých umělých inteligencí. Funkce obsluhující grafickou část programu jsou obsaženy v souborech graphics.c a graphics.h. V souboru human.c a .h je obsaženo rozhraní pro ovládání hry člověkem. Soubory game.c a .h obsahují herní mechanizmy (např. kontrolu výhry). A konečně soubor defines.h obsahují základní konstanty, jako například konstanty reprezentující velikost herního pole, konstanty reprezentující jednotlivé tokeny a podobně.

7 Licence

Licenci programu Gomoku můžete najít v souboru LICENCE, který by měl být distribuován zároveň se zdrojovými kódy programu.