Gymnázium, Praha 6, Arabská 14

Obor programování

Ročníková práce

Tomáš Vondra

Piškvorky

Duben 2024

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

Datum:_____ Podpis:_____

Název ročníkového projektu: Piškvorky

Autor: Tomáš Vondra

Anotace: Hlavním cílem tohoto projektu bylo vytvoření desktopové aplikace, která umožňuje hrát oblíbenou hru piškvorky. Tento program byl navržen s vlastním výpočetním algoritmem, který umožňuje hráči vyzvat počítač k partii. Aplikace nabízí také možnost hry mezi dvěma hráči nebo mezi dvěma počítači. Kromě toho hra poskytuje několik dalších variant, včetně možnosti výběru vlastního symbolu namísto tradičního křížku a kolečka. Hra je navíc připravena na další vylepšení a rozšíření o další varianty hry.

Obsah

1 Úvod 5
2 O Piškvorkách 6
2.1 Pravidla hry 6
2.2 Historie hry 6
2.3 Původ názvu hry 7
3 Algoritmus hledání optimálního tahu 8
3.1 Minimální a maximální skóre 8
3.2 Hledání hratelných políček 9
3.3 Výpočetní algoritmus n-tic 9
3.4 Výpočetní vzorec pro ohodnocení stavu plochy 10
4 Optimalizace algoritmu 12
4.1 Neexistence hratelného pole 12
4.2 Nutné tahy 12
4.2.1 Nezablokované čtveřice 12
4.2.2 Blokace děr 13
4.2.3 Nezablokované trojice 13
4.3 Vlákno 13
5 Grafické rozhraní 14
5.1 Hrací plocha 14
5.2 Ovládací panely 15
5.2.1 Nastavení hráčů 15
5.2.2 Nastavení hry 15
6 Závěr 17
7 Seznam obrázků 18
8 Bibliografie 19

1. Úvod

Aplikace byla vyvinuta v programovacím jazyce *Java* s využitím softwarové platformy *JavaFX* pro grafické rozhraní. Primárním cílem vývoje bylo vytvoření algoritmu, který je schopen řešit a hodnotit tahy tak, aby byly pro něj nejvýhodnější a pro soupeře naopak nejméně výhodné. Algoritmus je založen na rozsáhlém množství výpočtů a vzorců. Pro jeho rychlejší a přesnější fungování bylo použito několik optimalizacích.

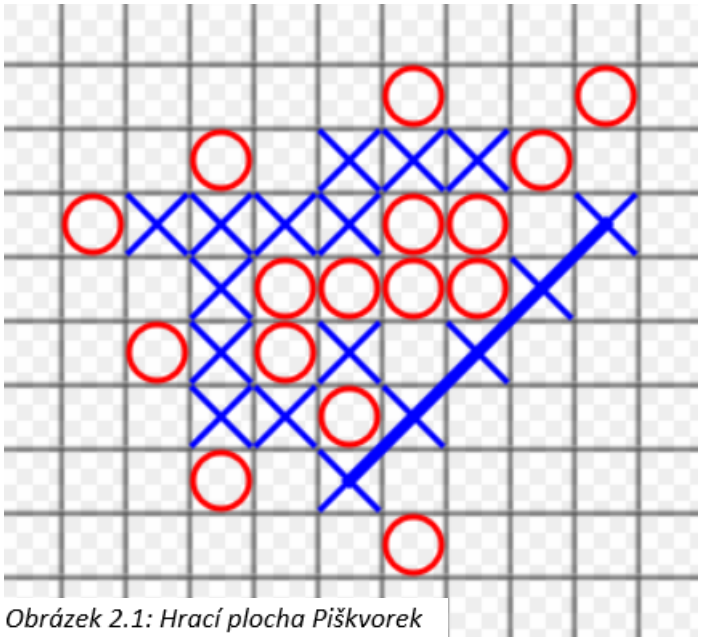
Hra byla navržena tak, aby podporovala všechny varianty hry. Hráč může vyzvat ke hře jiného hráče, případně může hrát tréninkovou hru sám se sebou. Hráč může také vyzvat ke hře počítač (s výpočetním algoritmem) nebo sledovat, jak by hra probíhala, pokud by se proti sobě postavily dva počítače.

Jedním z hlavních rozlišovacích prvků této aplikace oproti ostatním programům s hrou piškvorky je možnost neomezeného přepínání mezi hráčem a počítačem (výpočetním algoritmem) během hry. To umožňuje uživateli sledovat průběh hry a vidět, jak by hru za něj dále hrál počítač.

2. O Piškvorkách

2.1 Pravidla hry

Piškvorky je hra pro dva hráče, která se hraje na čtverečkovaném hracím poli (viz obrázek 2.1). Jejím cílem je vytvořit nepřerušenou řadu pěti svých symbolů (buď křížků, koleček nebo libovolně vybraných znaků) horizontálně, vertikálně nebo diagonálně. Hra začíná tak, že jeden hráč umístí svůj symbol do libovolného čtverečku na hracím poli, a poté druhý hráč umístí svůj. Hráči se střídají v umisťování svých symbolů na hracím poli, dokud jeden z hráčů nevytvoří souvislou řadu pěti symbolů nebo dokud nejsou všechna pole obsazena, což znamená remízu. Hra vyžaduje dobrou strategii a předvídání tahů soupeře.



Obrázek 2.1: Hrací plocha Piškvorek

2.2 Historie hry

Hra piškvorky je hrou s bohatou historií, která sahá tisíce let zpět. První zmínky o hře, která se podobá dnešním piškvorkám, pocházejí z oblasti okolí delty Žluté řeky v Číně a jsou datovány do období přibližně 2000 let před naším letopočtem. Tato hra, známá také pod názvem "*gomoku*" nebo "*pět v řadě*", se nezávisle na sobě vyvinula v různých částech světa, včetně antického Řecka a předkolumbovské Ameriky.

V České republice se piškvorky tradičně hrají na čtverečkovaném papíře, kde hráči střídavě kreslí křížky a kolečka, aby vytvořili nepřerušenou řadu pěti svých symbolů. Na mezinárodní úrovni se hraje s černými a bílými kameny na tradiční japonské desce Goban.

Zajímavostí je, že pro piškvorky na omezené i neomezené hrací ploše existuje neprohrávající strategie pro začínajícího hráče. Tento koncept, známý jako argument o kradení strategie, ukazuje, že pokud by existovala výherní

strategie pro druhého hráče, první hráč by ji mohl “ukrást” a použít ve svůj prospěch.

Soutěžní hraní piškvorek má v České republice silnou tradici, s hlavním organizátorem turnajů jako je *Česká federace piškvorek a renju*. V roce 2017 se v Praze dokonce konalo 7. mistrovství světa v piškvorkách. Každoročně se koná také Mistrovství republiky na letním festivalu Czech Open v Pardubicích, kde se hraje na 7 kol s časovým limitem 60 minut.

Piškvorky nejsou jen hrou pro dva hráče u jednoho stolu. Od roku 2000 se každoročně koná počítačový turnaj Gomocup, kde úkolem soutěžících je naprogramovat co nejlepší počítačovou umělou inteligenci, která bude hru řešit. V roce 2019 se stal mistrem světa v piškvorkách český hráč Martin Muzika, což byl historický úspěch pro české hráče této hry.

2.3 Původ názvu hry

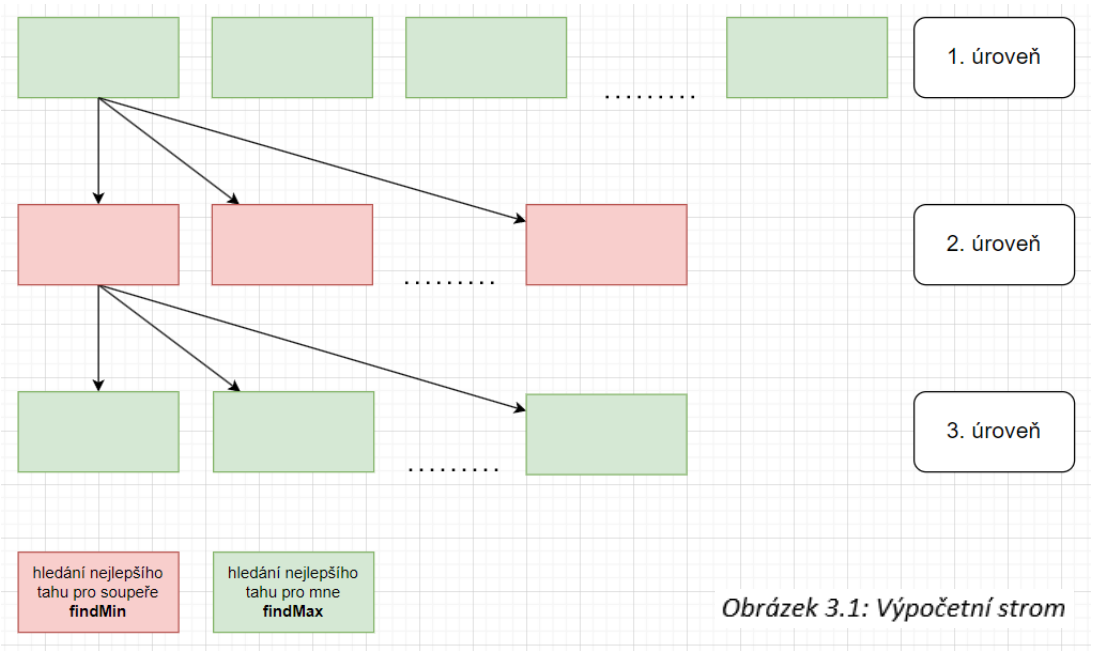
Původ pojmenování hry “piškvorky” v českém jazykovém kontextu zůstává do značné míry nejasný. Podle dostupných informací se zdá, že kořeny tohoto názvu vedou k národnímu podniku Tesla v Přelouči, kde skupina inženýrů běžně hrála karetní hry. V rámci této skupiny působil inženýr, který, přestože neovládal pravidla her, často poskytoval rady, což vedlo k jeho snížené popularitě mezi kolegy. V reakci na tuto situaci se zaměstnanci rozhodli vytvořit hru s názvem “piškorky”, která byla založena na náhodném odhazování karet a jejíž pravidla nebyla pevně stanovena. Ironií osudu se stal fakt, že méně oblíbený kolega začal nabízet rady i k této improvizované hře.

Tento příběh byl později zaznamenán mým učitelem klavíru, panem Václavem Rákosem, který byl svědkem události a příhodu vyprávěl v divadle ABC. Jeho vyprávění zaujalo herce Lubomíra Lipského natolik, že na základě tohoto vyprávění napsal krátkou televizní hru. Z důvodu respektování autorských práv došlo k úpravě názvu z “piškorek” na “piškvorky”. Popularizaci tohoto názvu lze připsat také vydavateli deskových her, který začal distribuovat hru pod tímto označením na trhu. Původní název hry není doložen.

3. Algoritmus hledání optimálního tahu

3.1 Minimální a maximální skóre

Pro řešící algoritmus jsou důležité dvě metody: *findMax*, *findMin*. Tyto metody se rekurzivně střídají v hledání nejlepšího tahu pro daného hráče (viz obrázek 3.1). Metoda *findMax* se stará o hledání maximálního skóre tahu, tedy nejvýhodnější pro aktuálního hráče, který je na tahu. Pro výpočet hodnoty tohoto tahu vychází z volání metody *findMin* (druhá úroveň). Metoda *findMin* má stejnou funkci, ale na rozdíl od *findMax* hledá minimální skóre, tj. tah pro aktuálního hráče nejméně výhodný. Metoda *findMin* se volá vždy, když se propočítává tah protihráče. Vždy předpokládáme, že zahraje tah pro sebe nejvýhodnější, tudíž pro aktuálního hráče nejméně výhodný. Výpočetní algoritmus následně propočítává další úroveň rekurze, která je v případě hry dvou hráčů prohledávána znovu metodou *findMax*. Na každé úrovni rekurze se vyhodnotí hodnota nejlepšího tahu na základě maxima, resp. minima a vrátí se



Obrázek 3.1: Výpočetní strom

volajícímu.

V obou metodách je nastaveno několik podmínek, které kontrolují, kdy mají dané metody skončit. Nejprve se ověří, jestli není konec hry (kdyby byl konec hry, nemělo by smysl vypočítávat tahy). Dále ověří, jestli program nedošel na maximální hloubku rekurze. Konec metod může být nastaven i ve chvíli, kdy

na tah byl nalezen protitah takový, že původní tah nemůže být ve vyšší úrovni již použit a bude stejně vyloučen. Např. pokud hledám minimum a našel jsem protitah s hodnotou -1 a vyšší úroveň rekurze našla v předchozím výpočtu již tah s hodnotou 0, pak po návratu do vyšší úrovně rekurze již nevrátím větší číslo než -1, jelikož hledám minimum, ale předchozí úroveň rekurze počítá maximum, tak tah zde bude eliminován tahem s hodnotou 0, který je lepší.

Velikost rekurze byla nastavena na hodnotu 3 na základě rychlosti a přesnosti výpočetního algoritmu. S hodnotou 4 je algoritmus velmi zdlouhavý a bylo by potřeba dalších optimalizací, s hodnotou 2 je zase příliš nepřesný. Hodnota 3 se zdá být optimální variantou.

3.2 Hledání hratelných políček

Pro snížení náročnosti výpočtu se vyberou nejdříve ta pole, která má smysl zkoumat. Program vytvoří objekt třídy *HracPocitacStav*, kde se vytvoří dvojrozměrné pole *stavHraciPlochy*, které představuje kopii pole *hraciPlocha*.

Na každém políčku, což je jednotka hrací plochy, je zpočátku jedna z hodnot: 1, 0, -1. Hodnota „0“ je udělena každému políčku, které patří hráči 1, to znamená, že na políčku se nachází jeho symbol. Hodnota „1“ je udělena každému políčku, které patří hráči 2. Zpočátku je hodnota „-1“ udělena každému políčku, které je prázdné a není označené za hratelné. V průběhu označování je políčkům udělena hodnota „-2“, pokud je označené za hratelné. Hratelné políčko je políčko, které se nachází v určeném okolí od již odehraného políčka. V tomto programu je okolí nastavené na hodnotu odpovídající maximální hloubce rekurze, to znamená, že hratelná políčka se v našem případě nacházejí nejdále 3 políčka od některého z odehraných políček (s hodnotou „0“ nebo „1“).

Výsledkem tohoto postupu je *ArrayList<Tah> potencionalniTahy*, kde je seznam všech pozic s hodnotou -2.

3.3 Výpočetní algoritmus n-tic

Celý algoritmus výpočtu hodnocení aktuálního stavu plochy vzhledem k aktuálně hrajícímu hráči stojí na počítání n-tic (nejmenší n-tice jsou dvojice a největší jsou n-tice potřebné pro výhru) a výpočetními vzorci (viz podkapitola 3.4). Všechny potřebné údaje jsou ukládány do pole n-tic (objekty třídy *Ntice*). N-tice obsahuje tyto údaje – **index hráče, počátek n-tice** (souřadnice políčka)**,**

konec n-tice (souřadnice políčka), **tahy k blokaci nebo rozšíření** (políčka, která zablokují **soupeřovu** n-tici nebo to jsou naopak políčka, která rozšíří **moji** n-tici na větší), **směr**. Směr je udáván jako dvě čísla, první představuje číslo, o kolik se posouvá n-tice na řádcích a druhé, o kolik se posouvá n-tice na sloupcích. Vznikají tak čtyři různé varianty směru n-tice z jejího počátku: (+1, 0), (0, +1), (+1, +1), (-1, +1).

Program pracuje s *boolean* polem *zpracovano* o rozměrech hrací plochy, které si průběžně ukládá políčka, která byla započítána do nějaké n-tice. Žádné políčko tak nebude započítáno vícekrát, ani nevzniká více n-tic z právě jedné n-tice. Pokud je políčko zpracované nebo prázdné, vyhledávání jde na další políčko. Pokud narazí program na nezpracované políčko, vyhledá lokaci konce a jeho následující políčko, zda je zablokované (pokud narazí na konec hrací plochy, znamená to, že je také zablokované). Postupně tak projde všechna políčka, která jsou zahraná nebo hratelná. Na políčka s hodnotou „-1“ výpočetní algoritmus ani nedojde.

N-tice, které jsou blokovány z obou stran, hned zahazujeme, takový případ nás nezajímá, protože n-tice je neměnná a již se s ní nedá pracovat.

Na základě takto sestavených n-tic se vygeneruje statistika do trojrozměrného pole s počty n-tic, kde prvním rozměrem je **index hráče** („0“ nebo „1“), druhým **délka n-tice** (zapisováno číslem, dvojice: 2) a třetím **stav blokace**. Stav blokace udává, jestli je n-tice zablokovaná z jedné strany (stav: 1) nebo není nijak blokovaná (stav: 0). Pokud se najde n-tice s požadovanou délkou na výhru, ukončí se vyhodnocování. Hodnota takového stavu plochy je buď MAX_VALUE v případě vyhrávající situace a nebo MIN_VALUE v případě prohrávající situace.

3.4 Výpočetní vzorec pro ohodnocení stavu plochy

Výpočetní vzorec vyhodnocuje skóre plochy po každém potencionálním tahu v okolí (tj. umístění symbolu na hratelné pole). Skóre n-tice je číselný údaj a vypočítává se ze statistiky n-tic (viz. 3.3) součtem za každou nalezenou n-tici:

hodnota = sum(délka n-tice * volná nezablokovaná místa * koeficient hráče (1, -1) * počet stejných n-tic)

Délka n-tice je počet symbolů, které obsahuje. *Volná nezablokovaná místa* je stav blokace, *koeficient hráče*, který algoritmus řeší, je kladný a *koeficient protihráče* je záporný.

Jakmile dojde kdykoli během výpočtu k vyhrávající n-tici, nastaví ohodnocení stavu hrací plochy na *MAX_VALUE*, pokud však najde prohrávající n-tici, nastaví ohodnocení na *MIN_VALUE*. V řešícím algoritmu je n-tice přesahující počet potřebný na výhru nastavena taktéž na vyhrávající (to znamená, že i devítice vyhraje každou hru v tomto programu).

4. Optimalizace algoritmu

Ve svém programu jsem musel provést několik optimalizací pro přesnější nebo rychlejší výpočetní algoritmus.

4.1 Neexistence hratelného pole

První optimalizace byla s hratelnými políčky. V případě, kdy měl hru zahájit počítač, byl problém v tom, že hratelné políčko na hrací ploše neexistovalo, a proto logicky vyplnil svým symbolem první políčko (se souřadnicí [0,0]). Bylo třeba nastavit, aby počítač hrál první tah na políčko, které se nachází uprostřed, pokud tedy neexistuje hratelné pole, pak se nastaví jedno a tím je střed plochy.

4.2 Nutné tahy

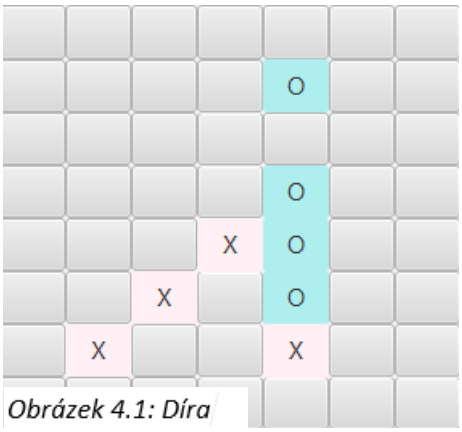
Cílem hledání nutných tahů je redukce rozvětvení rekurzivního stromu na důležité tahy. Pojem nutný tah má dva významy. V jednom případě jde o tah, který je nutný k vítězství aktuálního hráče a v druhém případě jde o tah, který je nutný, aby nedošlo k porážce protihráčem. Pokud taková políčka nalezne, nastaví je jako hratelné pole a ostatní hratelná pole z kapitoly 3.2 se ignorují. K vyhledávání je využit seznam všech n-tic z kapitoly 3.3.

4.2.1 Nezablokované čtveřice

Ve výchozím nastavení je počet symbolů na vítězství roven 5. V takovém případě program nejprve prohledá čtveřice (ve hře s počtem symbolů na vítězství 6 by stejným způsobem prohledával pětice). Pokud takovou čtveřici najde a je na tahu, pak nastaví jako hratelné pouze jedno pole, protože jde o závěrečný vítězný tah. Pokud je čtveřice soupeře přidá tahy k blokaci do hratelných polí a zkoumá další nalezené čtveřice.

Následný výpočet s takto omezenými hratelnými poli je rychlý a v případě více nalezených hratelných polí se pokusí najít tu lepší variantu.

4.2.2 Blokace děr



Obrázek 4.1: Díra

Pokud žádnou takovou čtveřici nenalezl, hledá, jestli se na hrací ploše nenachází díra, což je prázdné políčko nacházející se mezi trojicí a vyplněným políčkem téhož hráče a jehož vyplněním hráč vyhraje hru (viz obrázek 4.1).

Program vyhledává díry následujícím způsobem, při procházení každé trojice zjišťuje 2 následující políčka z obou stran a zjistí, zda se v dané vzdálenosti nevyskytuje vyplněné políčko. Pokud takovou situaci našel, přidá díru do hratelných tahů stejně jako nutný tah v 4.2.1.

4.2.3 Nezablokované trojice

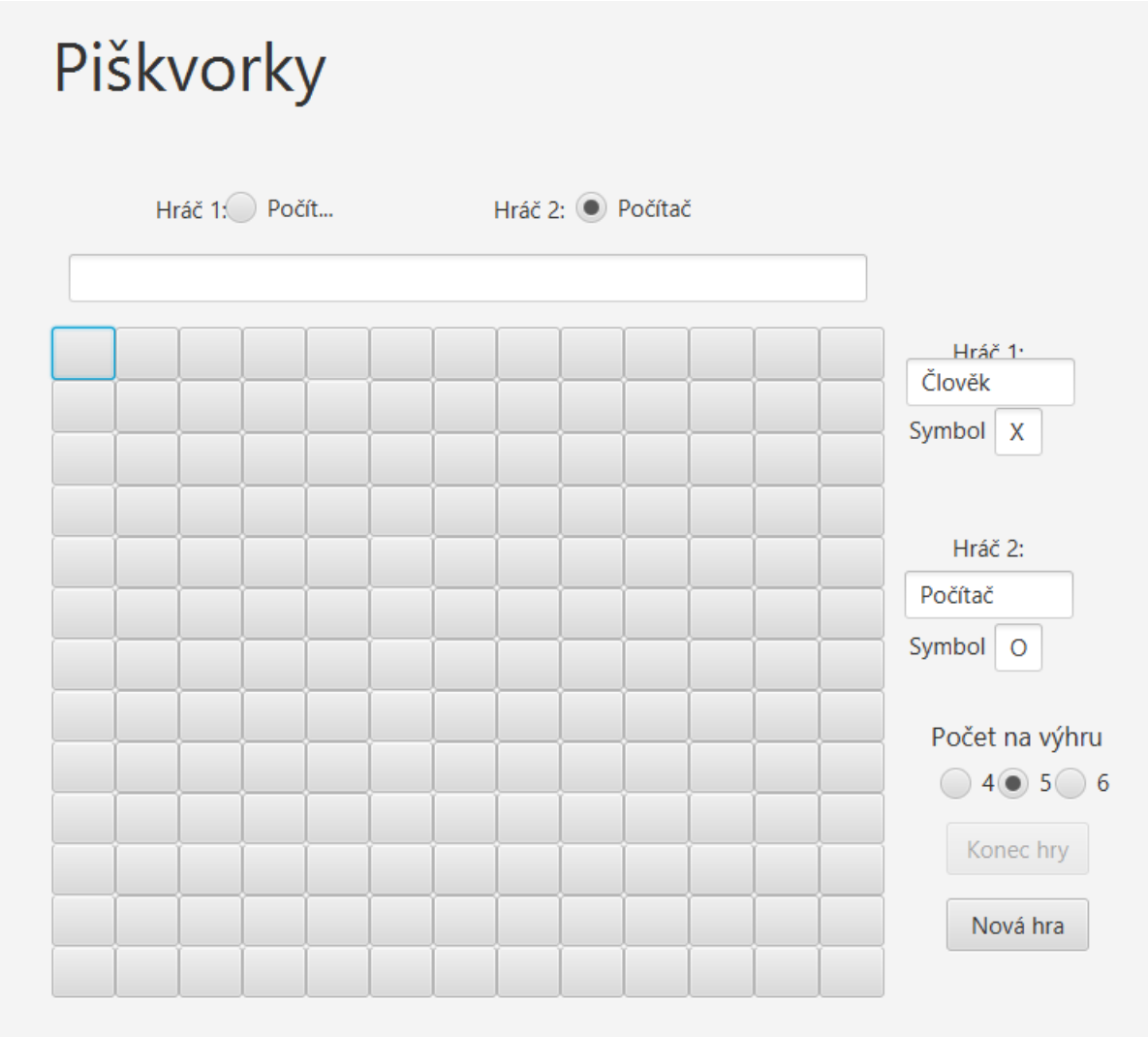
Pokud nenalezne ani případ z podkapitoly 4.2.1 a ani z podkapitoly 4.2.2, program hledá nezablokované trojice, které nejsou ani z jedné strany zablokované (stav blokace je „0“). V takovém případě přidá do hratelných polí obě blokující pole. Pokud trojice náleží počítači, rozšíří trojici na čtveřici, která povede k vítěznému tahu. Pokud trojice náleží soupeři, zablokuje ji, aby soupeř neprovedl vítězný tah v následujících tazích. Pokud program nenalezl ani jeden z předešlých případů, spustí celý výpočetní algoritmus (viz kapitola 3).

4.3 Vlákno

Pro výpočet je spuštěno pracovní vlákno, které se stará zvlášť o výpočetní algoritmus a neblokuje grafické vlákno. Tudíž máme dvě vlákna, jedno spravující pouze grafiku a druhé spravující pouze řešící algoritmus.

Musíme ale předejít chybám. Například, aby uživatel nemohl klikat na další tlačítka během výpočtu, kdy algoritmus hledá nejlepší tah.

5. Grafické rozhraní



Obrázek 5.1: Hrací okno

5.1 Hrací plocha

Hrací plocha se skládá celkem ze 169 políček (13 x 13). Každé políčko představuje jedno tlačítko *GridPanu*, které má ve složce *code* uložené **fxid: pole_x_y**, přičemž „*x*“ představuje souřadnici řádku a „*y*“ představuje souřadnici sloupce. Každé tlačítko má též ve složce *code* přidělenou metodu *udelejTah*. Tato metoda se spustí, jakmile nějaké tlačítko zmáčkneme. Metoda z *fxid* tlačítka dokáže rozpoznat, o jaký prvek v poli hrací plochy se jedná a zobrazí symbol hráče jako text tlačítka. Tah se ne vždy provede, v metodě jsou nastaveny podmínky: musí být na tahu hráč, který není počítačem nebo nesmí dojít k žádné

chybě nebo hra musí být spuštěna (před začátkem a ani po konci nelze zadávat tahy, pro hru je nutné zmáčknout tlačítko Nová hra).

Pro lepší orientaci na hrací ploše jsou políčka se symboly defaultně zabarvená. Pro lepší viditelnost symbolů byly vybrány světlejší barvy. Pro hráče jedna je nastavena barva jeho políček na světle růžovou, pro hráče 2 na světle modrou. Volitelnost vlastní barvy hráče je jedna z potenciálních vylepšení aplikace.

Pro lepší přehlednost na hrací ploše je na hracím okně přidáno textové pole, popisující děj na hrací ploše. V poli se během hry zobrazuje jméno hráče, který je na tahu. Na konci hry se v poli zobrazí jméno vítěze nebo v případě ukončení hry se zobrazí „*Hra byla ukončena*“.

5.2 Ovládací panely

Ovládacími panely jsou všechny prvky kolem hrací plochy v hracím okně, které je nutné nastavit před tím, než začne hra (většinu z nich nelze nastavovat během hry). Pokud budou textová pole vyplněna špatným způsobem, objeví se výpis hlášek požadavků, které nebyly splněny. Hra může začít jedině tehdy, když budou všechny požadavky splněny. Pro spuštění hry je třeba stisknout tlačítko *Nová hra*. Kdyby uživatel potřeboval nebo chtěl hru ukončit, stačí stisknout tlačítko *Konec hry* a hra skončí bez vítěze.

5.2.1 Nastavení hráčů

Před spuštěním hry je nutné zadat jméno hráče 1 a hráče 2 do textového pole. Defaultně jsou zde nastavena jména hráčů: **Člověk**, **Počítač**. Jména hráčů nesmí být prázdný řetězec znaků. Jména hráčů musejí být rozdílná.

Hráči si také vybírají svůj symbol, kterým budou vyplňovat tlačítka na hrací ploše. Opět jsou zde defaultně nastavené hodnoty: **X**, **O**. Symbol musí být právě jeden znak, to znamená, že znak nesmí být prázdný řetězec znaků a zároveň nesmí být delší než jeden znak. Symboly hráčů nesmí být shodné. Vybraný symbol je jakýkoli libovolný znak.

5.2.2 Nastavení hry

Před spuštěním hry je ještě nutné nastavit na dvou *RadioButton*-ech, jestli hráč 1 a hráč 2 budou hráči (lidé), výpočetní algoritmy (počítače) nebo jejich kombinace. Ve hře jsou tři možnosti, které mohou nastat. Za prvé, člověk bude hrát proti člověku (2 uživatelé). Za druhé, člověk proti počítači. Za třetí počítač

proti počítači. Tyto dva *RadioButton*-y jako jediné dva panely lze v průběhu hry libovolně ovládat (na začátku hry mohou hrát 2 lidé proti sobě a na konci hru dohrají 2 počítače).

V hracím okně se nacházejí ještě 3 *RadioButton*-y, kterými uživatel nastaví, kolik bude potřeba navazujících symbolů k výhře. Na výběr je ze tří možností: **4**, **5**, **6**. Méně než 4 by bylo pro takovou velkou hrací plochu málo a více než 6 by bylo zase příliš. Možnosti jiné než 5 nedávají moc smysl v případě dvou hráčů, jelikož při 4 se nelze prakticky bránit a při 6 by musel být hráč velmi nevšímavý, aby prohrál. Hodnoty by mohly mít význam v případě experimentování s více hráči.

6. Závěr

V závěrečné části mé ročníkové práce bych rád konstatoval, že výsledný program splnil všechny požadavky stanovené v zadání. Hlavním úkolem bylo vytvořit výpočetní algoritmus, který je schopen efektivně hrát hru piškvorky, čehož se mi podařilo dosáhnout. Navíc jsem představil originální funkci, která umožňuje libovolné přepínání hráčů za počítač během hry.

V budoucnosti vidím několik možností pro další vylepšení, na kterých mohu svůj program dále rozvíjet. Mezi tato vylepšení patří například integrace umělé inteligence do výpočetního algoritmu. Aplikace by mohla mít nastavení několika různých úrovní obtížnosti (lehká, střední, těžká), která by se projevila třeba v jisté náhodnosti výpočtu, a uživatel by si mohl vybrat, jakou úroveň obtížnosti preferuje pro svého soupeře ve hře typu hráč proti počítači.

Hra je navržena tak, aby umožňovala přidání více hráčů do jedné hry. Tento koncept by mohl být v budoucnu dále rozvíjen, avšak pravidla hry by pravděpodobně musela být upravena, aby hra byla stále hratelná bez komplikací, vzhledem k tomu, že tato varianta hry není běžně hraná.

7. Seznam obrázků

2.1 Hrací plocha Piškvorek 6

3.1 Výpočetní strom 10

4.1 Díra 12

5.1 Hrací okno 13

8. Bibliografie

Obrázek 2.1:

https://cs.wikipedia.org/wiki/Pi%C5%A1kvorky#/media/Soubor:Tic-tac-toe_5.png

MiniMax solving algorith tutorial:

https://github.com/DavidHurst/MiniMax-TicTacToe-Java

https://stackoverflow.com/questions/65689637/tic-tac-toe-ai-minimax-function-in-java

https://github.com/LazoCoder/Tic-Tac-Toe/blob/master/TicTacToe/ArtificialIntelligence/MiniMax.java

Algoritmus:

https://www.itnetwork.cz/java/diskuzni-forum-java/pomoc-s-piskvorky-algoritmus-pro-zjisteni-viteze--587a13eea297b

https://github.com/plastovicka/Piskvork

Task<V>

https://docs.oracle.com/javafx/2/api/javafx/concurrent/Task.html

Resource Bundle:

https://docs.oracle.com/javafx/2/api/javafx/concurrent/Task.html

Tvorba obrázku 3.1:

https://app.diagrams.net/