Střední průmyslová škola Třebíč

Maturitní práce

PIŠKVORKY

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITA4

Školní rok: 2024/2025 Jakub Libor Fejta

Zadání práce

Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Zadání ročníkové práce

Obor studia: **18-20-M/01 Informační technologie**

Celé jméno studenta:	**Jakub Libor Fejta**		
Třída:	**ITA4**	Školní rok:	**2024/2025**
Číslo tématu:	**39**		
Název tématu:	**Piškvorky**		
Rozsah práce:	**15 - 25 stránek textu**		

Specifické úkoly, které tato práce řeší:

Realizujte projekt, který bude představovat klasickou hru „Piškvorky“ pro dva hráče nebo hráče proti počítači s možností přerušení rozehrané hry. Hrací plocha bude rozdělena na matici N x M políček. Při kliknutí myší na kterékoliv políčko bude nakreslen uživatelem vybraný grafický symbol (např. křížek nebo kolečko). Při následujícím kliknutí se symbol automaticky změní. Pokud bude kliknuto na již nakreslené políčko, musí být signalizována chyba, např. zvukovým signálem. Program bude obsahovat automatickou kontrolu, zda počet stejných symbolů v jedné řadě v libovolném směru dosáhl požadovaného počtu. V takovém případě se tato řada vyznačí a hráči, kterému symbol náleží, se přičte bod. Pokud se vyznačí více řad, započítá se více bodů. Stav hry se bude zobrazovat ve stavovém řádku pod hrací plochou. Program bude obsahovat hlavní menu s položkami „Nová hra“, „Otevření“, „Uložení“, „Ukončení“, „Historie nejlepších“, „Demo“ a „Nastavení“. V menu Nastavení lze nastavit hrací symboly, počet hracích symbolů pro výhru, počet kol hry, obtížnosti počítače. Zajistěte, aby se při změně velikosti formuláře úměrně překreslila hrací plocha. Vývojové prostředí VS, jazyk C#, sdílení a ukládání GitHub.

Termín odevzdání:	**28. března 2025, 23.00**
Vedoucí projektu:	**Mgr. Andrea Odehnalová**
Oponent:	**Ing. Marek Hrozníček**
Schválil:	**Ing. Petra Hrbáčková, ředitelka školy**

ABSTRAKT

Tématem maturitní práce je zpracování klasické hry „Piškvorky“ do digitální formy. Cíl práce je vytvořit funkční desktopovou aplikaci s možností hry jednoho hráče proti počítači nebo dvou hráčů proti sobě. Hra bude zároveň obsahovat nastavení různých herních parametrů (velikost hrací plochy, počet symbolů pro výhru a další).

KLÍČOVÁ SLOVA

maturitní práce, piškvorky, hra, C#, .NET, GitHub

ABSTRACT

The topic of the graduation thesis is to digitalize the classic game „Piškvorky “(Five in a Row). This work aims to create a functional desktop application with the option of one player playing against a computer or two players playing against each other. The game will also include the settings of different parameters in the application (game board size, number of symbols to win etc.).

KEYWORDS

graduation thesis, five in a row, game, C#, .NET, GitHub

PODĚKOVÁNÍ

Děkuji Mgr. Andreje Odehnalové za cenné připomínky a rady, které mi poskytla při vypracování maturitní práce.

V Třebíči dne 28. března 2025 podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne 28. března 2025

podpis autora

Obsah

Úvod

Již dlouhodobě trávím svůj volný čas hraním hry Piškvorky, což zahrnuje i účast na různých turnajích. Byl jsem dokonce 5.nejlepší v kategorii jednotlivců soutěže pIsQworky 2023 i pIsQworky 2024 a nejlepší junior a nováček v turnaji Brnocup 2024. Už delší dobu jsem přemýšlel o vytvoření vlastní platformy, takže když jsem viděl maturitní téma „Piškvorky“ jako projekt v C#, rozhodl jsem se tento nápad realizovat.

Cílem projektu je vytvořit platformu pro hraní populární hry „Piškvorky“ v digitální formě. Jedná se o logickou hru, která se hraje typicky ve dvou hráčích na čtverečkovaném papíru. Ve světě je hra známá pod názvem „Gomoku“ a hraje se na dřevěné desce s černými a bílými kameny. Hra má jediný cíl, tím je vytvořit nepřerušenou řadu 5 symbolů (5 a více u Piškvorek a tedy i v aplikaci).

V platformě jde počet symbolů pro výhru změnit, stejně tak velikost hracího pole nebo i hrací symboly. Platforma obsahuje hru pro dva hráče na jednom zařízení nebo hru pro jednoho hráče proti počítači.

1 Teoretická část

Práce se zabývá přetvořením hry Piškvorky do digitální formy. Používá programovací jazyk C#. Celý projekt je tvořen v prostředí Microsoft Visual Studiu. Projekt byl vytvořen ve frameworku .NET. Pro zálohování projektu a správu verzí se využívá GitHub.

1.1 Programovací jazyk C#

„Jazyk C# je nejoblíbenějším jazykem pro platformu .NET, bezplatné, multiplatformní open-sourcové vývojové prostředí. Programy jazyka C# se můžou spouštět na mnoha různých zařízeních, od zařízení Internetu věcí (IoT) až po cloud a všude mezi sebou. Můžete psát aplikace pro telefony, stolní počítače a přenosné počítače a servery.“ [1]

Je to moderní objektově orientovaný programovací jazyk vyvinutý společností Microsoft jako součást platformy .NET. Navazuje na jazyky C a C++, ale přidává vyšší úroveň abstrakce a bezpečnost, spolu s velkým množstvím knihoven, díky čemuž je pátý nejoblíbenější programovací jazyk na světě.[2] C# je multiparadigmatický jazyk, což znamená, že podporuje imperativní, funkcionální, deklarativní a objektově orientované programování.[3] To zaručuje flexibilitu při řešení všech různých problémů.

Jazyk je navržený s důrazem na silnou typovou bezpečností. To znamená že veškeré proměnné a výrazy musí mít definovaný datový typ, což minimalizuje chyby způsobené nesprávným typovým převodem. Navíc díky implementaci garbage collectoru (GC) C# automaticky spravuje paměť, což zvyšuje stabilitu a výkon aplikací a ulehčuje práci vývojářům.[4] Mezi hlavní výhody C# patří jeho rozsáhlá standardní knihovna a možnost snadné integrace s technologiemi Microsoftu, jako jsou Windows, Azure a další.[4]

C# nachází uplatnění v široké škále aplikací – od desktopových a webových řešení po herní vývoj, například prostřednictvím nástroje Unity. Díky své univerzálnosti a snadné čitelnosti je ideálním jazykem jak pro začátečníky, tak pro pokročilé vývojáře.

1.2 Microsoft Visual Studio

Microsoft Visual Studio je moderní a výkonné integrované vývojové prostředí (IDE), které vývojářům umožňuje vytvářet různé typy aplikací – od desktopových přes

webové až po mobilní nebo cloudové. Toto prostředí je navrženo tak, aby bylo flexibilní a efektivní při práci na projektech jakékoliv velikosti. Podporuje širokou škálu programovacích jazyků, zejména C#, C++, Python a další, a integruje pokročilé nástroje pro ladění, testování, správu verzí i nasazování aplikací.

Visual Studio klade důraz na produktivitu vývojářů – obsahuje funkce jako IntelliSense pro rychlé doplňování kódu a IntelliCode, které využívá umělou inteligenci ke zlepšování návrhů kódu na základě předchozích zkušeností. Pro týmy vývojářů nabízí možnosti spolupráce v reálném čase a integraci s platformami jako GitHub a Azure. Kromě toho podporuje i vývoj multiplatformních aplikací prostřednictvím .NET MAUI či webových aplikací pomocí Blazoru. [5][6]

Tyto funkce dělají z Visual Studia jedno z nejrozšířenějších prostředí, které používají pokročilý vývojáři, ale zároveň je ideální i pro studenty a začínající vývojáře.

1.3 GitHub

GitHub je hlavní cloud-based platforma pro správu verzí a týmovou spolupráci, která staví na populárním verzovacím systému Git. Umožňuje vývojářům sledovat změny v kódu, spolupracovat na projektech a snadno spravovat různé verze aplikací. GitHub poskytuje uživatelům možnost vytvářet veřejné i soukromé repozitáře, což ho činí ideálním nástrojem pro open-source projekty i firemní vývoj.[7]

Jednou z nejužitečnějších funkcí platformy je větvení projektu. Každý projekt má hlavní větev, ze které mohou vycházet jiné větve. Ty umožňuje platforma pomocí pull requestů snadno integrovat do hlavní větve projektu, a to schválením vedlejší větve autorem projektu. Další funkcí je GitHub Actions, které podporují automatizaci procesů, jako je testování a nasazení aplikací. GitHub také funguje jako sociální síť pro vývojáře, kde mohou sdílet své projekty, sledovat ostatní programátory a inspirovat se jejich prací.

1.4 .NET Framework

.NET Framework je softwarová platforma od společnosti Microsoft, která umožňuje vývoj aplikací na Windows. Obsahuje bohatou knihovnu tříd, které poskytují předdefinované funkce pro širokou škálu úkolů, jako je práce s databázemi, grafickým uživatelským rozhraním, sítěmi a dalšími.

Jednou z hlavních výhod .NET Frameworku je jeho modulární architektura a podpora více programovacích jazyků, jako jsou C#, VB.NET a F#. Framework zahrnuje Common Language Runtime (CLR), což je runtime prostředí, které spravuje běh aplikací, uvolňování paměti a bezpečnost. [8]

1.5 Piškvorky

Hra „Piškvorky“ je klasická hra hraná dvěma hráči typicky na list čtverečkovaného papíru. Hráči se střídají v zakreslování symbolů do políček na papíru, přičemž cílem hry je jako první mít nepřerušenou řadu pěti nebo více symbolů, a to diagonálně, horizontálně či vertikálně.

Hra je v České republice velmi známá a typicky hrána studenty ve škole, často i v době vyučování. Z tohoto vznikla česká soutěž v piškvorkách nesoucí název „pIsQworky“ pořádaná pod spolkem Student Cyber Games. Jedná se o Mistrovství škol v České republice a na Slovensku. Soutěž se dělí na okresní, krajské a státní (Grandfinále) kolo.[9]

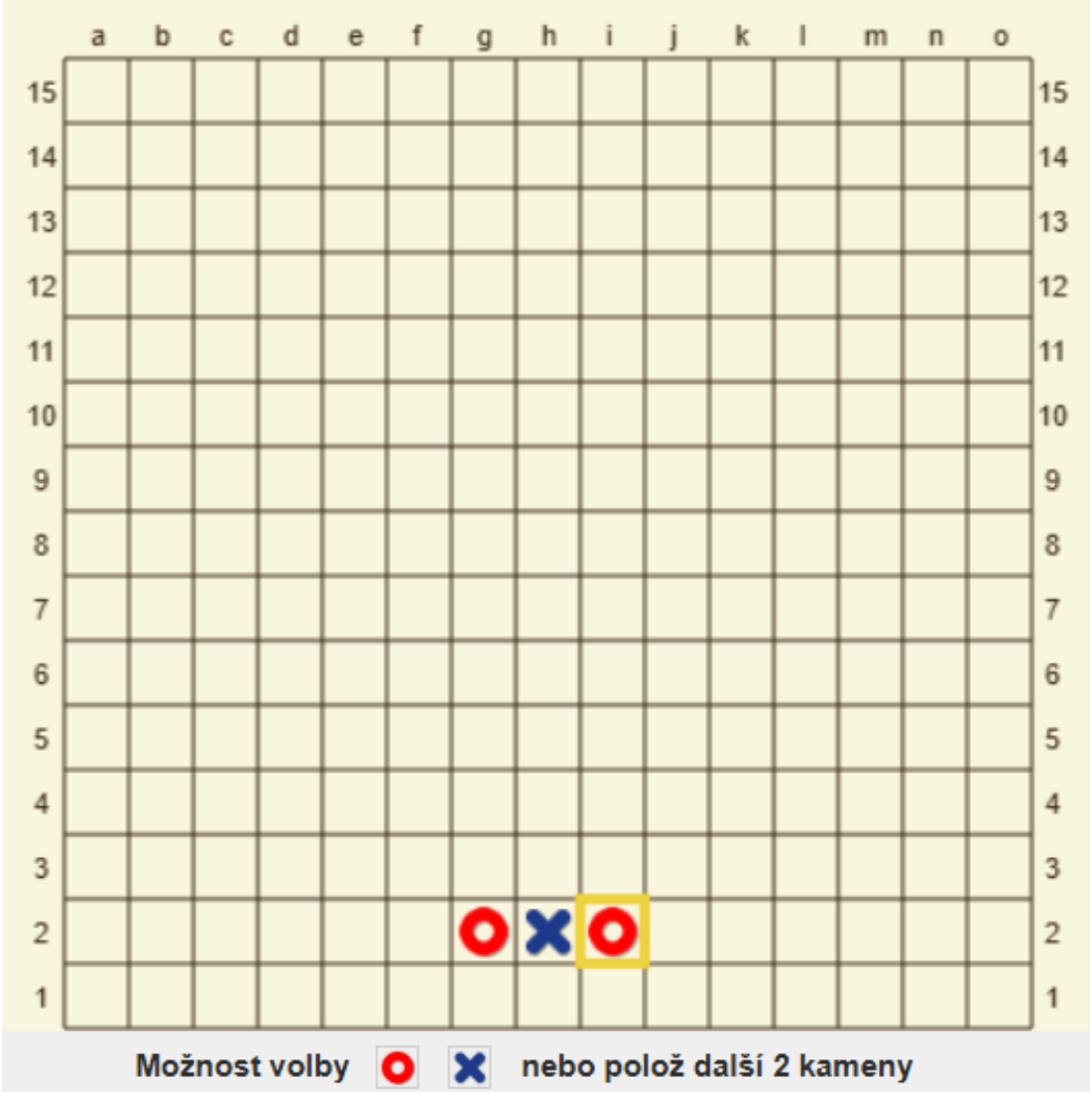
Ve světě jsou Piškvorky známé jako „Five in a Row“ v anglicky mluvících zemích nebo „Gomoku“ v Asii. „Five in a Row“ můžeme přeložit jako pět v řadě a „Gomoku“ jako poskládej pět, oba názvy tedy jednoduše vystihují hlavní cíl hry. [10]

Pro pokročilejší hráče začne u Piškvorek být znatelná výhoda u začínajícího hráče. Hráč má výhodu tak znatelnou, že za optimálních defenzivních tahů soupeře vyhraje svým 18.tahem.[11] Aby se hra stala vyrovnanou, byla vytvořena různá pravidla, ze kterých se momentálně na světové úrovni používá pravidlo SWAP2. Předchůdcem tohoto pravidla bylo pravidlo SWAP.

1.5.1 Pravidlo SWAP

„ Výhodu začínajícího lze dobře omezit takzvaně zahájením SWAP. První hráč zahraje tři symboly, z toho dva stejné (například dva křížky a jedno kolečko). Poté si druhý hráč vybere, za jaký znak bude hrát – tedy kolečko, nebo křížek. Ten s méně symboly na desce pokračuje ve hře jako normálně. Příklad: jestliže hráč A zahraje dva křížky a jedno kolečko, hráč B má dvě možnosti: vybrat si kolečko a hrát (protože koleček je na stole méně), nebo si vybrat křížek a nechat druhého hráče hrát kolečko (protože má

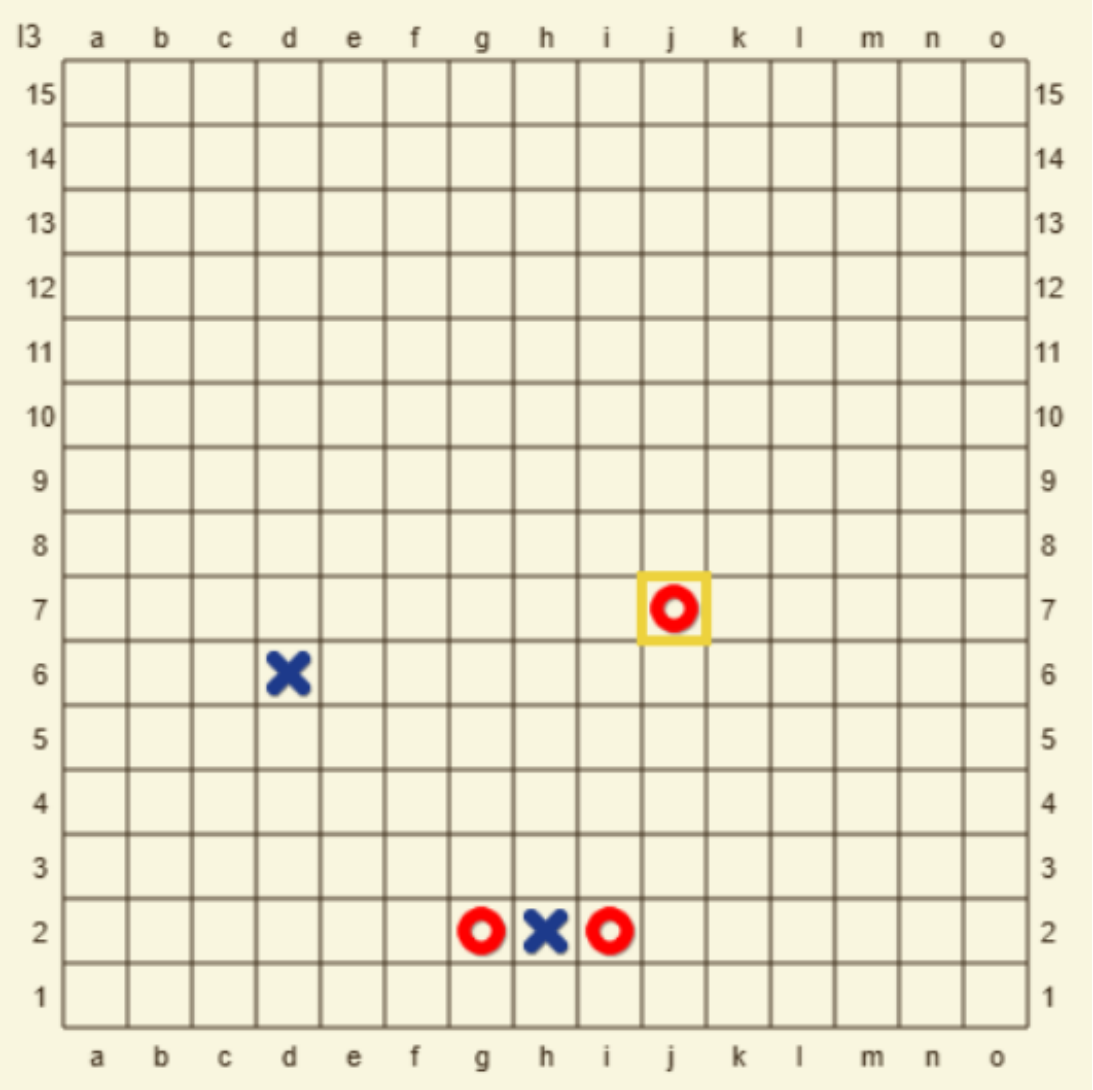
méně symbolů). Tato technika je spravedlivá, jelikož se první hráč snaží udělat pozici stejně výhodnou pro křížek, jako pro kolečko – kdyby tak neudělal, druhý hráč si vybere symbol s výhodnějším postavením na desce.“ [10]



Obr. 1 – První hráč položil SWAP, druhý hráč si vybírá symbol

1.5.2 Pravidlo SWAP2

„Při obyčejném SWAPu má začínající hráč výhodu v tom, že si může předem rozmyslet a analyzovat své zahájení za obě strany. Proto má druhý hráč ještě třetí možnost kromě výběru kolečka nebo křížku, a to zahrát další dva znaky – jedno kolečko a jeden křížek. Tím naruší prvnímu hráči jeho promyšlenou strategii a přenechává mu výběr znaku. První hráč si teď vybere kolečko nebo křížek podobně, jako mohl druhý hráč, ale na rozdíl od něj už nemůže hrát další dva znaky.“ [10]



Obr. 2 – Druhý hráč si nevybral symbol, položil další dva symboly

1.6 Gomoku

Gomoku (z japonštiny „poskládej pět“) je strategická desková hra podobná piškvorkám, známá po celém světě. Její pravidla jsou téměř totožná, ale hraje se na uzavřené desce o velikosti 15x15 políček a řada pro výhru musí být dlouhá právě pět symbolů a ne více. Hraje se na dřevěné desce vycházející z Gobanu, který se používá u hry Go, s černými a bílými kameny. Desky jsou rozdílné pouze velikostí (19x19 pro Go proti 15x15 pro Gomoku). [12][13]

Hra pochází z Asie a s největší pravděpodobností vznikla z velmi populární hry v Asii zvané „Go“. Na první pohled názvy vypadají podobně, což je však pouze shoda náhod. Gomoku můžeme totiž rozložit na „go“ znamenající pět a „moku“ neboli průsečík v japonštině. Další názvy specifické pro každou zemi mohou zahrnovat např.: kakugo, gomoku-narabe, itsutsu-ishi, gobang, morphion luffarschack, omok, wuziqi, connect5, nought&crosses, rendzu, caro nebo kolko i krzyzyk. [14]

1.7 Go

Go je desková hra, jež se hraje na hrací desce Goban s velikostí 19x19 políček. Hraje se s černými a bílými kameny a, na rozdíl od Šachů, černý začíná. Pro trénink se využívají i desky s velikostí 13x13 a 9x9. Cílem hry je obklíčit průsečíky, zajmout soupeřovy kameny a tím získat větší skóre, než soupeř.

Hra pochází původem z Číny, kde se však nazývá Wej-čchi. Dnes je hra velmi populární v celé východní Asii, i díky jejímu rozvoji v Japonsku.[15]

1.8 Umělá inteligence

Umělá inteligence (AI) je obor informatiky zabývající se tvorbou systémů, které jsou schopny analyzovat data, rozhodovat se a učit se na základě předchozích zkušeností. [16] V herním prostředí se AI využívá k vytvoření protivníků, kteří jsou schopni analyzovat herní stavy a hledat optimální tahy a tvoří tak výzvu pro uživatele.

V logických hrách, kde je hra s dokonalou informací, se nejčastěji uplatňují stromové vyhledávací algoritmy. Nejznámějším příkladem je algoritmus Minimax, který rekurzivně prochází herní strom a hodnotí možné varianty tahů tak, aby hráč maximalizoval svůj výsledek, zatímco protivník jej minimalizoval.[17][18]

Pro snížení počtu prohledávaných větví se často využívá technika alfa-beta ořezávání. Tato metoda eliminuje neperspektivní větve herního stromu a výrazně tak zrychluje výpočet optimálního tahu, což umožňuje dosáhnout hlubšího prohledávání a tím silnější strategie. [18]

Kromě stromového prohledávání se v herní AI často uplatňují i heuristické metody, které slouží k rychlému ohodnocení aktuální pozice bez nutnosti procházet celý strom do konce. Kombinace těchto technik – pravidlových metod, heuristik a stromového vyhledávání – umožňuje vytvořit robustní AI, která konkuruje zkušeným lidským hráčům. [17]

1.8.1 Stromové algoritmy pro hraní her (Minimax a alfa-beta)

Algoritmus Minimax je klasickým nástrojem pro rozhodování ve strategických hrách pro dva hráče s úplnou informací (Piškvorky, Go, atd.). Jeho princip spočívá v rekurzivním prohledávání herního stromu, kde hráč na tahu vybírá tah, který maximalizuje jeho šanci na nejlepší výsledek, zatímco předpokládá, že protivník vybírá tahy minimalizující jeho zisk. [19]

Na úrovni listů stromu nebo při dosažení maximální nastavené hloubky se využívá statická ohodnocovací funkce – heuristika, která bodově ohodnocuje danou pozici. Tato funkce zohledňuje strategické prvky, například v piškvorkách může zahrnovat počet hrozeb výhry, koncentraci kamenů nebo kontrolu středu desky.

Jedním z hlavních problémů minimaxu je kombinatorická exploze možností – počet uzlů roste exponenciálně s hloubkou prohledávání. U her jako piškvorky, kde je na prázdné 15×15 desce k dispozici stovky tahů, by bylo nutné prohledat miliony až miliardy pozic, což je výpočetně neúnosné. [20]

Aby se počet prohledávaných větví výrazně snížil, využívá se technika alfa-beta ořezávání. Tato metoda udržuje dvě hranice – hodnotu alfa (nejlepší výsledek maximalizujícího hráče) a beta (nejlepší výsledek minimalizujícího hráče). Pokud zjistí, že další prohledávání aktuální větve nemůže vést k lepšímu výsledku, větev se okamžitě ukončí. [21]

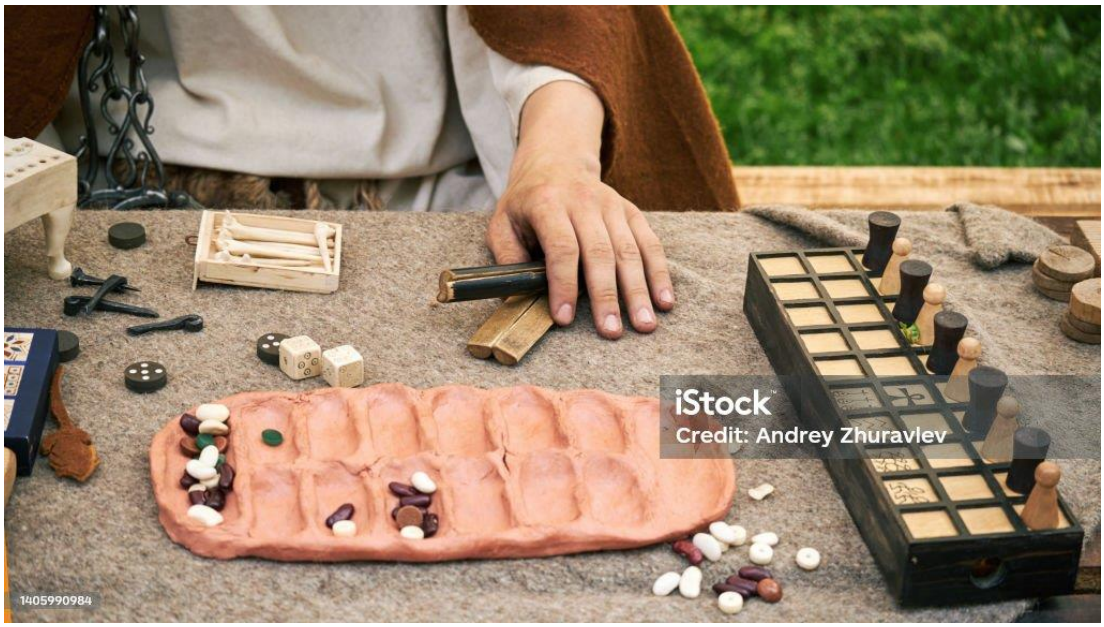
Další optimalizace zahrnují heuristiky pro řazení tahů, transpoziční tabulky a techniku killer move, které dále zvyšují efektivitu prohledávání. [20]

1.8.2 Pravidlové metody

Pravidlové metody v herní AI spočívají v explicitním vložení znalostí o hře do systému prostřednictvím pravidel, například: „pokud mohu ihned vyhrát, proveď vítězný tah; pokud soupeři hrozí výhra v příštím tahu, zablokuj ji“. Tato pravidla pomáhají ošetřit základní taktické situace, kdy je počet možných variant omezený. Čistě pravidlový přístup však nedokáže pokrýt veškerou složitost hry, a proto se v praxi kombinuje s algoritmickým hledáním (např. minimax s heuristikou), což umožňuje dosáhnout vysoké úrovně herní AI. [16][17]

1.9 Historie logických her

Deskové hry mají bohatou historii sahající tisíce let zpět. Byly nalezeny v různých kulturách po celém světě a sloužily nejen k zábavě, ale také k výuce a sociální interakci.



Obr. 3 – Starověká hra Senet

Jednou z nejstarších známých deskových her je Hra Senet, datovaná kolem roku 3500 př. n. l. a pocházející ze Starověkého Egypta. Další z prvních známých deskových her je královská hra Uru, jež pochází z období kolem 2600 př. n. l. z Mezopotámie. Nejstarší deskovou hrou, která je dodnes populární, je Go, které vzniklo před více než

2500 lety v Číně a dodnes se v něm po celé Asii soutěží. Zároveň je Go pravděpodobně předchůdcem asijské verze Piškvorek – Gomoku. [22]

1.10 Dopad logických her na přemýšlení

Logické hry, jako jsou piškvorky, šachy nebo Go, představují nejen zábavnou formu trávení času, ale zároveň působí jako efektivní nástroj pro rozvoj kognitivních schopností a přemýšlení. Tyto hry vyžadují od hráčů vysokou míru pozornosti, strategického plánování a schopnost předvídat tahy.

Jak uvedl Jakub Horák, vítěz soutěže PišQworky: *„Je to skvělý trénink na paměť, představivost a abstraktní myšlení.“* [23] Jakub Horák je také tvůrcem vlastního AI zaměřeného na piškvorky – XOXO, které se umístilo na 8. místě v turnaji AI Gomocupu 2021.[24][24] Tato hra hráče nutí přemýšlet v širším kontextu, hodnotit možnosti a efektivně se rozhodovat, což jsou dovednosti přenositelné i do každodenního života.

Dále se při hraní logických her rozvíjí trpělivost a vytrvalost. Často se totiž stává, že hráč musí dlouho čekat na příležitost ke správnému tahu nebo musí čelit složité herní situaci, která vyžaduje klid a promyšlený přístup. Podle článku na MujRozhlas.cz „piškvorky rozvíjejí mozek ve všech směrech, vyžadují trpělivost, vytrvalost i logické myšlení.“[25] Tyto vlastnosti se přirozeně promítají do dalších aspektů života hráčů, například do jejich schopnosti řešit problémy nebo zvládat stresové situace.

2 Praktická část

Praktická část práce se zabývá samotným vývojem aplikace Piškvorky – od založení projektu a konfigurace prostředí až po implementaci herní logiky, uživatelského rozhraní a umělé inteligence. Aplikace je vyvíjena v programovacím jazyce C# v prostředí Microsoft Visual Studio a využívá .NET Framework. Díky tomu je plně kompatibilní s operačním systémem Windows a může běžet na široké škále zařízení – od stolních počítačů po notebooky a tablety se systémem Windows.

Kromě funkce pro dvě lokální zařízení (dva hráči na jednom počítači) aplikace nabízí i hru proti počítači, který disponuje třemi různými obtížnostmi. Při vytváření těchto obtížností byl kladen důraz na vyváženost mezi rychlostí výpočtu tahů a kvalitou strategie virtuálního soupeře. Pro hráče, kteří chtějí mít soutěžní náboj, je k dispozici také tabulka s názvem „Historie nejlepších“, v níž se ukládají data o deseti nejlepších výsledcích.

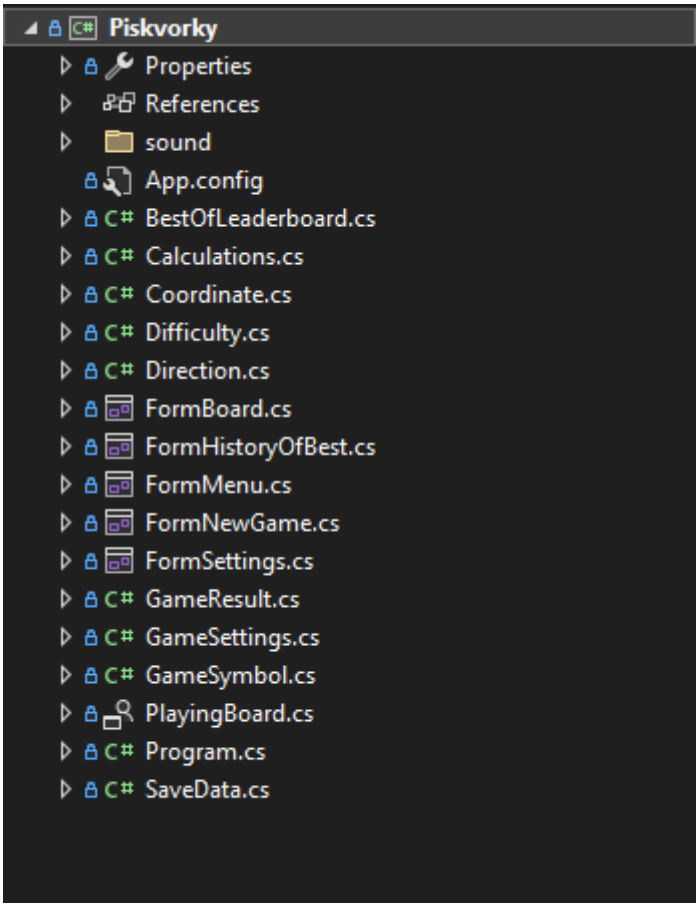
Velkou přidanou hodnotou je také možnost ukládání a načítání hry. Uživatel tak nemusí dohrát hru na jeden zátah, ale může aplikaci kdykoli zavřít a později se k rozehrané partii vrátit. Ovládání aplikace je podpořeno responzivní herní plochou – velikost hracího pole se dynamicky přizpůsobuje oknu aplikace a lze ji volitelně nastavit. Tím je zajištěno, že se Piškvorky pohodlně ovládají na různých rozlišeních obrazovek.

2.1 Vytvoření projektu

Vytvoření projektu probíhalo v Microsoft Visual Studiu. Při založení projektu byla zvolena šablona aplikace „Windows Forms (.NET Framework)“, díky níž je možné intuitivně navrhovat uživatelské rozhraní a zároveň využívat knihovny .NET.

Aplikace vznikla pod verzovacím systémem Git; repozitář je hostován na GitHubu. Tím je zajištěn přehled o vývoji, a hlavně možnost jednoduchého zálohování. Zároveň je zdrojový kód veřejně přístupný a kdokoliv může navrhovat úpravy.

Před tím, než jsem začal vývoj aplikace, došlo k hrubému návrhu rozložení aplikace a promyšlení jednotlivých funkcí. Poté jsem si připravil formuláře a hlavní třídu pro výpočty – Calculations.

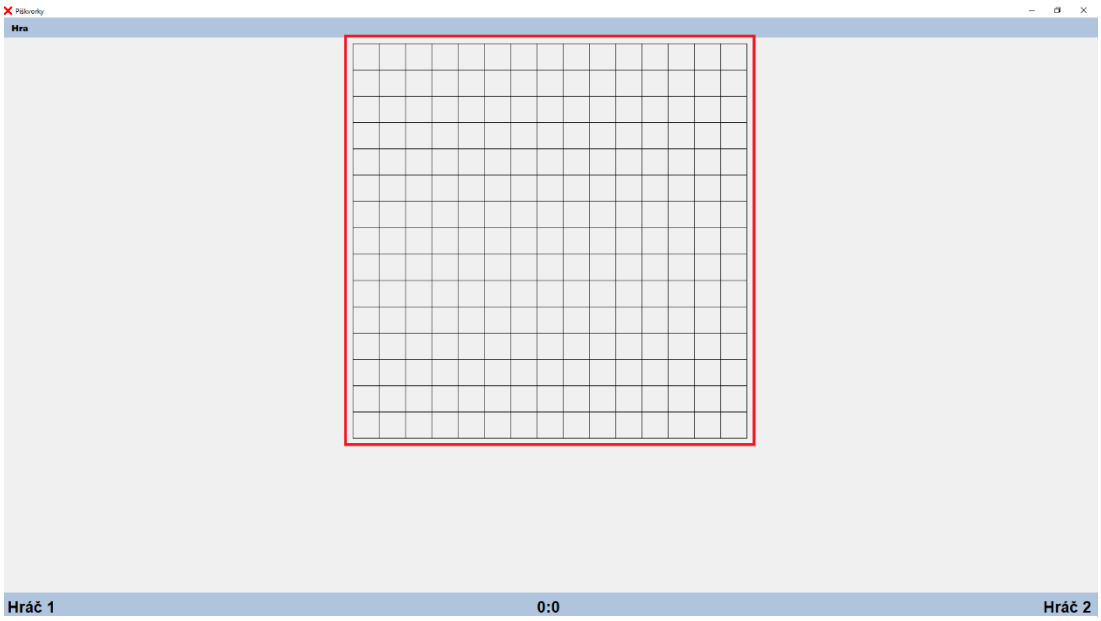


Obr. 4 – Struktura projektu

Po vytvoření základního návrhu a struktury projektu jsem implementoval novou komponentu PlayingBoard – hrací plocha, na které stojí celý projekt. Při práci na uživatelském rozhraní jsem vycházel z principů pro Windows Forms, tedy formou jednoduchého drag-and-drop umisťování ovládacích prvků (Button, Menu, Label).

2.2 Hrací plocha

Hrací plocha představuje jádro celé hry. V klasických Piškvorkách je obvykle tvořena papírem se čtverečkovanou sítí, v digitální verzi je vytvořena jako vlastní komponenta s názvem PlayingBoard pomocí použití třídy Graphics a metody DrawLine. Ta je potom přidána do formuláře FormBoard, kde je zároveň stavový řádek a menu.



Obr. 5 – Hrací plocha vyznačena červeně

2.2.1 Responzivita

Rozměry hrací plochy lze dynamicky měnit podle velikosti hlavního okna. Uživatel si v nastavení může zvolit, kolik polí bude hrací plocha obsahovat (např. 3x3, 9x9, 20×20 atp.).

Vždy, když uživatel upraví velikost formuláře, zavoláme metodu BoardRedraw a přesuneme instanci komponenty playingBoard na střed formuláře. Metoda BoardRedraw přizpůsobí velikost pole tak, aby hrací plocha vždy vyplnila prostor ve formuláři. Změna velikosti pole proběhne pouze pokud byla výška formuláře změněna o alespoň 5 pixelů.

```
public void BoardRedraw()
{
    if (Math.Abs(height - Height) > ResizeThreshold)
    {
        int currentHeight = Height;
        playingBoard1.FieldSize = (int)(fieldSize * (float)currentHeight / MinimumSize.Height);
        height = Height;
    }
}
```

Obr. 6 – Metoda BoardRedraw

Zároveň se velikost hrací plochy automaticky přizpůsobí počtu polí na hrací ploše. Například při počtu políček 3x3 se hrací plocha zvětší a při počtu 25x25 se zmenší, aby se ve formuláři vyplnil prostor, ale zároveň aby komponenta formulář nepřesahovala.

2.2.2 Vykreslení hrací plochy

Implementace hrací plochy sestává z vykreslení mřížky a interakce s uživatelem. Používá se metoda DrawLine, která vykreslí čáry horizontálně a vertikálně vždy vzdálené fieldSize od sebe. Proměnná fieldSize tedy reprezentuje velikost políčka.

```
private void DrawBoard (Graphics graphics)
{
    for (int i = 0; i <= boardSize; i++)
    {
        graphics.DrawLine(GridPen, 0, i * fieldSize, boardSize * fieldSize, i * fieldSize);
        graphics.DrawLine(GridPen, i * fieldSize, 0, i * fieldSize, boardSize * fieldSize);
    }
}
```

Obr. 7 – Metoda DrawBoard

Při kliknutí myší kdekoliv na komponentu nejprve proběhne kontrola, zda není na tahu počítač. Tím ošetříme, že uživatel nemůže hrát dřív, než se vykreslí tah počítače. Poté je vyhodnoceno, do kterého pole uživatel klikl, a tah je uložen do vícerozměrného pole pomocí metody AddMove.

```
private async void PlayingBoard_MouseClick(object sender, MouseEventArgs e)
{
    if (isAIThinking)
        return;
    int x = e.X / fieldSize;
    int y = e.Y / fieldSize;
    await AddMove(x, y);

}
```

Obr. 8 – Vlastnost MouseClick

Veškeré grafické prvky vykresluji přes vlastnost Paint, která proběhne vždy, když je potřeba vykreslit komponentu. Můžeme ji také vyvolat pomocí metody Refresh nebo Invalidate.

2.2.3 Metoda AddMove

Účelem metody AddMove je přidat symbol do pole ve třídě Calculations, ve které probíhá veškerá vnitřní logika hry. V metodě je navíc obsažena logika pro ukončení hry v případě výhry nebo remízy, a to skrze proměnnou result typu GameResult, což

```
private async Task AddMove(int x, int y)
{
    if (!Calc.CordsOnBoard(x, y))
        return;
    if (Calc.SymbolsOnBoard[x, y] != GameSymbol.Free)
    {
        fieldFullSound.Play();
        return;
    }
    movesToWin++;
    GameResult result = Calc.AddSymbol(x, y, currentPlayer, out winningRows);
    lastDrawnX = x;
    lastDrawnY = y;
    Refresh();
    if (result == GameResult.Win)
    {
        if (movesToWin < movesToWinMin && currentPlayer == GameSymbol.Symbol1)
        {
            movesToWinMin = movesToWin;
        }
        PlayerWon?.Invoke(currentPlayer);
        ResetGame();
        return;
    }
    else if (result == GameResult.Draw)
    {
        Draw?.Invoke();
        ResetGame();
```

Obr. 9 – Metoda AddMove (1.část)

je Enumerace s hodnotami Continue, Win a Draw. Jedná se o metodu asynchronní a typu Task, což nám umožňuje pouštět kód bez blokování hlavního vlákna. Pro naši metodu je toto ideální, protože potřebujeme počkat na tah počítače, a zároveň umožnit uživateli interagovat s aplikací. Metoda nejprve zkontroluje, zda uživatel opravdu kliknul do políčka na hrací ploše a poté zda je políčko volné. Až když obě podmínky projdou, zapíše se pomocí metody Calc (instance třídy Calculations) AddSymbol, ta zároveň vrátí hodnotu, která informuje o stavu hry a vrátí list listů pointů (List<List<Point>>), který v případě výhry obsahuje všechny symboly všech výherních řad, abychom je mohli později vyznačit.

Dále metoda AddMove zkontroluje, zda se jedná o výhru, pokud ne, zkontroluje, zda se jedná o remízu. V obou případech se zavolá náležitá událost, která informuje FormBoard o konci hry. V případě výhry hráče s prvním symbolem se upraví jeho nejkratší počet tahů k výhře, který se poté zapisuje do historie nejlepších (toto je určeno pro hru s počítačem, víme že počítač má vždy 2.symbol). Poté se zavolá metoda ResetGame, která vynuluje veškerá vnitřní data a vyčistí hrací plochu. Pokud nenastane výhra ani remíza, znamená to, že hra pokračuje a na tahu je soupeř, tím pádem se z currentPlayer (hráč na tahu) stává opponent (soupeř).

```
if (isPlayingAI && currentPlayer == GameSymbol.Symbol2)
{
    isAIThinking = true;
    Stopwatch stopwatch = Stopwatch.StartNew();

    var bestMove = await Task.Run(() =>
    {
        int optX, optY;
        Calc.GetBestMove(GetDifficulty(AIDifficulty), out optX, out optY, currentPlayer);
        return (optX, optY);
    });

    stopwatch.Stop();
    int elapsed = (int)stopwatch.ElapsedMilliseconds;
    int delay = Math.Max(0, 1000 - elapsed);
    await Task.Delay(delay);

    await AddMove(bestMove.optX, bestMove.optY);
    isAIThinking = false;
}
```

Obr. 10 – Metoda AddMove (2.část – AI)

Další část metody je určena pro případ hry s počítačem. Zde využíváme opět faktu, že počítač je vždy 2.symbol v podmínce, která ověřuje, zda uživatel v nastavení vybral hru proti počítači a zároveň je počítač na tahu. Počítač je nastavený tak, aby vždy

„přemýšlel“ alespoň 1 sekundu. Díky tomu je hra s počítačem přehlednější a hráč se necítí tolik pod tlakem. Tohoto zpoždění dosáhneme spuštěním stopek, vyčkáním na zpracování nejlepšího tahu podle obtížnosti vybrané uživatelem (tah se vypočítá opět ve třídě Calculations), zastavení stopek a dopočítání do jedné sekundy, pokud čas nepřesáhl sekundu. Poté dochází k rekurzivnímu volání metody (metoda volá sebe sama) s nově získanými hodnotami. Následuje opět první část kódu pro zapsání tahu a pro kontrolu, zda tah neukončil hru.

```
if (GameSettings.DemoMode)
{

    isAIThinking = true;
    Stopwatch stopwatch = Stopwatch.StartNew();

    var bestMove = await Task.Run(() =>
    {
        int aiX, aiY;
        if (currentPlayer == GameSymbol.Symbol1)
        {
            Calc.GetBestMove(Difficulty.Hard, out aiX, out aiY, currentPlayer);
        }
        else
        {
            Calc.GetBestMove(Difficulty.Medium, out aiX, out aiY, currentPlayer);
        }
        return (aiX, aiY);
    });

    stopwatch.Stop();
    int elapsed = (int)stopwatch.ElapsedMilliseconds;
    int delay = Math.Max(0, 1000 - elapsed);
    await Task.Delay(delay);

    await AddMove(bestMove.aiX, bestMove.aiY);
    isAIThinking = false;
}
```

Obr. 11 – Metoda AddMove (3.část – DemoMode)

Metoda AddMove také obsahuje podmínku, která ověřuje, zda je spuštěn demo mód. V případě že podmínka platí, probíhá hra mezi dvěma počítači. Kód vypadá podobně jako u hry s počítačem, akorát nemusíme kontrolovat kdo je na tahu. Aby se však jednalo o zajímavé hry, zvolil jsem pro každý symbol jinou obtížnost počítače, to znamená, že ze kterékoliv pozice zadané uživatelem by měl vyjít počítač se s těžkou obtížností jako vítěz. Pokud by byli obě umělé inteligence na stejné úrovni, skoro všechny jejich hry by končili remízou, leda že by byla zvolena lehká obtížnost, která však naopak dělá hloupé chyby, ze kterých se uživatel nemá tolik příležitostí učit.

2.2.4 Design

Pro lepší uživatelský zážitek je vždy zvýrazněn poslední zahraný tah a také výherní řada (řady, pokud je jich více zároveň). K tomu využijeme metodu HighlightMove, která nejprve ověří, že se souřadnice tahu nacházejí na hrací ploše a poté jednoduše pomocí metody FillRectangle ze třídy Graphics vybarvíme čtvercovou plochu o velikosti fieldSize – 1 na příslušných souřadnicích.

```
private void HighlightMove(Graphics graphics, Color color, int x, int y)
{
    if (Calc.CordsOnBoard(x, y))
    {
        graphics.FillRectangle(new SolidBrush(color),
                               x * fieldSize + 1,
                               y * fieldSize + 1,
                               fieldSize - 1,
                               fieldSize - 1);
    }
}
```

Obr. 12 – Metoda HighlightMove

Pro zvýraznění posledního tahu pak metodě jednoduše předáme souřadnice, které získáme v metodě AddMove po zavolání metody AddSymbol. Souřadnice předávám v metodě HighlightLastMove, kterou voláme ve vlastnosti Paint.

```
private void HighlightWinRows(Graphics graphics, List<List<Point>> winRows)
{
    if (winRows != null)
    {
        foreach (List<Point> winRow in winRows)
        {
            foreach (Point p in winRow)
            {
                HighlightMove(graphics, winRowColor, p.X, p.Y);
            }
        }
    }
}
```

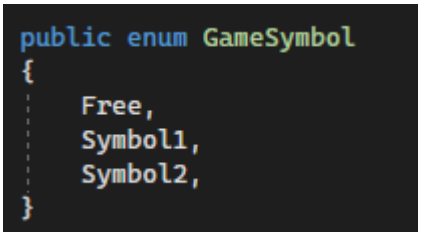
Obr. 13 – Metoda HighlightWinRows

Pro zvýraznění všech políček, které jsou součástí výhry projdeme všechny body (Point) ve všech listech bodů (jednotlivé výherní řady) v listu s názvem winRows a každý bod zvýrazníme opět metodou HighlightMove. K tomuto dochází v metodě HighlightWinRows.

2.3 Hrací symboly

V piškvorkách rozlišujeme nejčastěji křížky (X) a kroužky (O). V této aplikaci jsou symboly reprezentovány grafickými objekty, které se kreslí podle souřadnic kliknutí. Uživatel si přitom může změnit typ symbolu. Může zvolit jiné tvary nebo barvy, pokud k tomu chce dodat hře osobní styl.

Z programátorského hlediska je symbol uložen jako Enumerace:



Obr. 14 – Enumerace GameSymbol

Vykreslení symbolu probíhá v metodě DrawSymbol, která přijímá parametr symbol, aby vykreslila správný symbol.

```
private void DrawSymbol(Graphics graphics, GameSymbol symbol, int x, int y)
{
    if (!Calc.CordsOnBoard(x, y))
        throw new Exception("Souřadnice se nachází mimo hrací plochu!");

    string emoji = symbol == GameSymbol.Symbol1 ? Symbol1Emoji : (symbol == GameSymbol.Symbol2 ? Symbol2Emoji : null);
    if (emoji != null)
    {
        Color symbolColor = symbol == GameSymbol.Symbol1 ? Symbol1Color : Symbol2Color;

        using (Font emojiFont = new Font("Segoe UI Emoji", fieldSize / 2))
        using (Brush textBrush = new SolidBrush(symbolColor))
        {
            SizeF textSize = graphics.MeasureString(emoji, emojiFont);
            float posX = x * fieldSize + (fieldSize - textSize.Width) / 2;
            float posY = y * fieldSize + (fieldSize - textSize.Height) / 2;

            graphics.DrawString(emoji, emojiFont, textBrush, posX, posY);
        }
    }
}
```

Obr. 15 – Metoda pro vykreslování symbolů

Pro vykreslování jsem nejprve využíval Graphics metody DrawLine a DrawEllipse, které pro křížek a kolečko byli dostačující, ale později jsem je nahradil metodou DrawString, která umožňuje vykreslování libovolných řetězců včetně emoji a usnadňuje nastavování vlastních herních symbolů pro uživatele.

2.4 Výpočty

O veškeré vnitřní prvky hry se stará třída Calculations. Počítá tahy počítače, kontroluje stav hry a uchovává vnitřní reprezentaci hry.

```
private int boardSize = 15; // Velikost desky
private GameSymbol[,] symbolsOnBoard; // Symboly na desce (x, y)
private short winLength = 5; // Počet symbolů v řadě potřebný k výhře
private short[,,,] symbolsInRow; // Počty symbolů v řadách ve všech směrech (x, y, směr, symbol)
private short[,,,] openEnds; // Počet volných konců (možná prodloužení řady) (x, y, směr, symbol)
private short[,] DirectionSigns; // Směrová znaménka pro horizontální, diagonální a vertikální (směr, souřadnice)
private int rowsLeftOnBoard; // Zbývající počet možných vítězných řad na desce
private int[,,] fieldValues; // Hodnoty polí používané při výběru tahů v heuristických přístupech (x, y, symbol)
private int[] Values; // Hodnoty na základě počtu symbolů v řadě (0, 0, 4, 20, 100, 500)
private short winRowsCount = 0; // Počet nalezených vítězných řad
private Random random = new Random(); // Instance pro generování náhodných čísel
```

Obr. 16 – Atributy třídy Calculations

Nejdůležitější proměnnou třídy je dvourozměrné pole typu GameSymbol, symbolsOnBoard, které pro každé pole hrací plochy obsahuje Symbol1, Symbol2 nebo Free (volné políčko). Jedná se tedy o vnitřní reprezentaci hrací plochy. Další velmi důležitá proměnná je čtyřrozměrné pole symbolsInRow, které pro oba symboly (tedy i pro oba hráče) uchovává počet symbolů v řadě ve všech směrech po celé desce. Proměnná esenciální pro hru s počítačem je trochrozměrné pole fieldValues, kde je uložena hodnota každého pole na hrací ploše pro každého hráče z pole Values. Na základě tohoto pole poté počítač zahraje tah na políčko s nejvyšší hodnotou.

```
/// <summary>
/// Vymaže pole s počty symbolů v řadě a volné konce a inicializuje počet možných vítězných řad.
/// </summary>
public void ClearSymbolsInRow()
{
    symbolsInRow = new short[boardSize, boardSize, (short)Direction.Diag2 + 1, (short)GameSymbol.Symbol2 + 1];
    openEnds = new short[boardSize, boardSize, (short)Direction.Diag2 + 1, (short)GameSymbol.Symbol2 + 1];
    for (int x = 0; x < boardSize; x++)
    {
        for (int y = 0; y < boardSize; y++)
        {
            foreach (Direction direction in Enum.GetValues(typeof(Direction)))
            {
                for (GameSymbol symbol = GameSymbol.Symbol1; symbol < GameSymbol.Free; symbol++)
                {
                    symbolsInRow[x, y, (short)direction, (short)symbol] = 0;
                    openEnds[x, y, (short)direction, (short)symbol] = 0;
                }
            }
        }
    }
    rowsLeftOnBoard = 4 * (2 * boardSize - (winLength - 1)) * (boardSize - (winLength - 1));
}
```

Obr. 17 – Metoda ClearSymbolsInRow

Všechny výše uvedené proměnné musíme inicializovat a také je vyresetovat po konci hry. K tomu máme metody ClearBoard, ClearSymbolsInRow a ClearFieldValues. Tyto metody inicializují pole a zároveň jej vždy vynulují a tím ho připraví na další hru.

V metodě ClearSymbolsInRow navíc inicializujeme počet možných výher na hrací ploše a pole s počtem otevřených konců pro každou řadu symbolů. Metoda ClearBoard jednoduše nastaví všechny hodnoty na Free a metoda ClearFieldValues na 0.

2.4.1 AddSymbol

Stěžejní metoda pro celou aplikaci je metoda AddSymbol, která kontroluje stav hry, přidává každý tah zahraný uživatelem do pole SymbolsOnBoard a zároveň přidá každému tahu hodnotu pro hráče na tahu i pro soupeře.

```
public GameResult AddSymbol(int x, int y, GameSymbol player, out List<List<Point>> winRows)
{
    winRows = new List<List<Point>>();
    GameResult result = GameResult.Continue;
    foreach (Direction dir in Enum.GetValues(typeof(Direction)))
    {
        short dirHor = DirectionSigns[(short)dir, (short)Coordinate.X];
        short dirVer = DirectionSigns[(short)dir, (short)Coordinate.Y];

        for (int i = 0; i < winLength; i++)
        {
            int posX = x + dirHor * i;
            int posY = y + dirVer * i;
            if (PositionWithinBounds(posX, posY, dirHor, dirVer))
            {
                result = IncludeDraw(ref symbolsInRow[posX, posY, (short)dir, (short)player]);
                if (result != GameResult.Continue)
                {
                    if (result == GameResult.Win)
                    {
                        SymbolsOnBoard[x, y] = player;
                        winRows = GetWinningRows(x, y, player);
                        winRowsCount = (short)winRows.Count;
                    }
                    break;
                }

                for (int j = 0; j < winLength; j++)
                {
                    short opponent = (short)GetOpponent(player);
                    int posXfield = posX - dirHor * j;
                    int posYfield = posY - dirVer * j;
                    RecalcValue(
                        symbolsInRow[posX, posY, (short)dir, (short)player],
                        symbolsInRow[posX, posY, (short)dir, opponent],
                        openEnds[posX, posY, (short)dir, (short)player],
                        openEnds[posX, posY, (short)dir, opponent],
                        ref FieldValues[posXfield, posYfield, (short)player],
                        ref FieldValues[posXfield, posYfield, opponent]);
                }
            }
        }
        if (result != GameResult.Continue)
        {
            break;
        }
    }

    SymbolsOnBoard[x, y] = player;
    if (result == GameResult.Continue && rowsLeftOnBoard <= 0)
```

Obr. 18 – Metoda AddSymbol

Jako první si musíme nastavit stav hry na Continue, což znamená, že hra bude pokračovat. Nyní budeme procházet všechny možné směry cyklem foreach. Poté se pro každý zvolený směr postupně posouváme od aktuální pozice tahu pomocí směrových vektorů získaných z pole DirectionSigns. Tyto vektory (s hodnotami 1, 0 a -1) určují posun podél os X a Y. V rámci každého směru se pro každý posun vynásobený indexem získají souřadnice polí, kde se pomocí metody PositionWithinBounds ověřuje, zda jsou v rámci desky. Pokud ano, zavolá se metoda IncludeDraw, která zvýší počet souvislých symbolů a zkontroluje, zda řada nedosáhla potřebné délky pro vítězství a také zkontroluje, jestli byla vytvořena nová řada. V případě že došlo k vytvoření nové řady, odečte se počet výherních řad na hrací ploše. Pokud se zjistí, že došlo k výhře, uloží se symbol do pole SymbolsOnBoard a získají se všechny vítězné řady pomocí metody GetWinningRows. Pokud výhra nenastane, vnitřním cyklem se pro každé pole v aktuální řadě rekalculují heuristické hodnoty (metoda RecalcValue) pro aktuálního hráče i soupeře. Nakonec se symbol vždy uloží do pole SymbolsOnBoard a pokud není již možné vytvořit žádnou vítěznou řadu, vyhodnotí se stav hry jako remíza.

2.5 Umělá inteligence

Jednou z nejzajímavějších částí aplikace je implementace hry proti počítači. I přes existenci různých umělých inteligencí, včetně těch, které se sami učí, jsem zvolil jednoduchý algoritmus cílený na rychlost a minimax algoritmus pro větší komplexitu tahů. V základním algoritmu je každému hracímu poli přiřazena hodnota skládající se ze součtu dvou hodnot určujících důležitost pole pro hráče i pro soupeře (hodnota hráče i soupeře je vynásobena koeficientem, který nám umožňuje měnit důležitost hráčových nebo soupeřových polí, díky čemuž můžeme nastavit agresivitu inteligence). Ze všech hracích polí je vybráno pole s maximální hodnotou. Pokud se

```
int bestValue = int.MinValue;

bestX = boardSize / 2;
bestY = boardSize / 2;
if (SymbolsOnBoard[bestX, bestY] == GameSymbol.Free)
{
    bestValue = 4; // Základní hodnota pro tah ve středu
}

for (int i = 0; i < boardSize; i++)
{
    for (int j = 0; j < boardSize; j++)
    {
        if (SymbolsOnBoard[i, j] == GameSymbol.Free)
        {
            int value =
                (FieldValues[i, j, (short)player] * 16)
                + 1
                + (FieldValues[i, j, (short)opponent] * 8);

            value += GetCenterBonus(i, j); // Bonus za blízkost ke středu

            if (value > bestValue)
            {
                bestValue = value;
                bestX = i;
                bestY = j;
            }
        }
    }
}
```

Obr. 19 – Základní algoritmus pro výpočet tahu Umělé inteligence

jedná o první tah hry, bude počítač vždy hrát doprostřed, díky prvotní podmínce, která středovému poli dává základně malou nenulovou hodnotu.

Aplikace disponuje třemi úrovněmi obtížnosti, z nichž lehká a střední obě vycházejí z algoritmu právě popsaného. Těžká obtížnost využívá komplikovanější algoritmus.

2.5.1 Lehká obtížnost

Počítač hraje většinou smysluplné tahy, avšak může zahrát i náhodně. V 80 % případů vybere ze 3 nejlepších svých tahů, které vyhodnocuje podle základního algoritmu. Ve zbylých 20 % nenabízí výrazné strategické varianty a hraje náhodně. Celkově tato obtížnost slouží spíše pro seznámení s hrou.

2.5.2 Střední obtížnost

Střední obtížnost nejprve prohledá hrací plochu a zkontroluje, zda nemá výhru v jednom tahu. Stejně prohledá hrací plochu, aby zjistil, zda soupeř nemá výhru v jeho následujícím tahu. Pokud výherní tah nalezne, nemusí proběhnout základní algoritmus a počítač zahraje výherní tah (v případě, že se jedná o výherní tah soupeře zahraje blokující tah). V ostatních případech projde základním algoritmem a zahraje tah vyhodnocený algoritmem jako nejoptimálnější.

```
public void GetBestMove_Medium(out int x, out int y, GameSymbol player)
{
    if (TryFindWinningMove(player, out x, out y))
        return;
    GameSymbol opponent = GetOpponent(player);
    if (TryFindWinningMove(opponent, out x, out y))
        return;
    PickMoveByFieldValue_Medium(out x, out y, player, opponent);
}
```

Obr. 20 – Metoda zjišťující nejlepší tah pro střední obtížnost Umělé inteligence

2.5.3 Těžká obtížnost

Stejně jako u střední obtížnosti se i u těžké obtížnosti nejprve zkontroluje rychlá výhra nebo prohra, avšak rozdíl je v tom, že tato obtížnost nevyužívá základní algoritmus, ale využívá komplikovanější algoritmus – Minimax.

```
public void GetBestMove_Hard(out int x, out int y, GameSymbol player)
{
    if (TryFindWinningMove(player, out x, out y))
        return;
    GameSymbol opponent = GetOpponent(player);
    if (TryFindWinningMove(opponent, out x, out y))
        return;

    (int bestX, int bestY) = FindBestMoveIterative(player, 200);
    x = bestX;
    y = bestY;
}
```

Obr. 21 – Metoda zjišťující nejlepší tah pro těžkou obtížnost Umělé inteligence

Jelikož se jedná o časově náročný algoritmus, potřebujeme pohlídat, že tahy nebudou trvat moc dlouho. To zajistíme metodou FindBestMoveIterative, která přijímá čas v ms jako parametr. V metodě samotné se poté spustí stopky a pokud stopky přesáhnou časový limit, metoda vrátí výsledek algoritmu.

```
private int Minimax(int depth, int maxDepth, GameSymbol currentPlayer,
                    GameSymbol maximizingPlayer, int alpha, int beta, out (int x, int y) bestMove)
{
    bestMove = (-1, -1);

    bool isFull = true;
    for (int i = 0; i < boardSize && isFull; i++)
    {
        for (int j = 0; j < boardSize; j++)
        {
            if (SymbolsOnBoard[i, j] == GameSymbol.Free)
            {
                isFull = false;
                break;
            }
        }
    }
    if (isFull) return EvaluateBoard(maximizingPlayer);
    if (depth == maxDepth) return EvaluateBoard(maximizingPlayer);

    List<(int x, int y)> candidates = GetCandidateMoves();
    if (candidates.Count == 0)
        return EvaluateBoard(maximizingPlayer);

    int bestScore;
```

Obr. 22 – Minimax algoritmus (1.část)

V algoritmu se nejprve ujistíme, že hrací plocha není zaplněná. Také se ujistíme, že hloubka nedosáhla maximální hloubky. Pokud některá z podmínek neprojde, dojde k zavolání metody EvaluateBoard, která pomocí metody EvaluateBoardAdvanced ohodnotí hráče i soupeře a vrátí rozdíl těchto hodnot. Vrácená hodnota rovnou slouží jako výstup Minimax algoritmu a ten je ukončen. Za ohodnocení tahů je zodpovědná metoda EvaluateLine, ta podle počtu symbolů v řadě, který by nastal po položení symbolu na aktuální herní pole přiřadí poli hodnotu 10,100,1000 nebo 10000. Také tato metoda bere v potaz, zda se jedná o otevřenou řadu z obou stran, z jedné strany, nebo o uzavřenou řadu.

```
if (currentPlayer == maximizingPlayer)
{
    bestScore = int.MinValue;
    foreach (var move in candidates)
    {
        SymbolsOnBoard[move.x, move.y] = currentPlayer;
        if (WouldThisMoveWin(move.x, move.y, currentPlayer))
        {
            SymbolsOnBoard[move.x, move.y] = GameSymbol.Free;
            bestMove = move;
            return 10000 - depth;
        }
        int score = Minimax(depth + 1, maxDepth, GetOpponent(currentPlayer),
            maximizingPlayer, alpha, beta, out _);
        SymbolsOnBoard[move.x, move.y] = GameSymbol.Free;
        if (score > bestScore)
        {
            bestScore = score;
            bestMove = move;
        }
        alpha = Math.Max(alpha, score);
        if (beta <= alpha)
            break;
    }
}
else
{
    bestScore = int.MaxValue;
    foreach (var move in candidates)
    {
        SymbolsOnBoard[move.x, move.y] = currentPlayer;
        if (WouldThisMoveWin(move.x, move.y, currentPlayer))
        {
            SymbolsOnBoard[move.x, move.y] = GameSymbol.Free;
            bestMove = move;
            return -10000 + depth;
        }
        int score = Minimax(depth + 1, maxDepth, GetOpponent(currentPlayer),
            maximizingPlayer, alpha, beta, out _);
        SymbolsOnBoard[move.x, move.y] = GameSymbol.Free;
        if (score < bestScore)
        {
            bestScore = score;
            bestMove = move;
        }
        beta = Math.Min(beta, score);
        if (beta <= alpha)
            break;
```

Obr. 23 – Minimax algoritmus (2.část – alfa-beta ořezávání)

Algoritmus podobně jako základní algoritmus prochází každé pole hry, avšak v minimax algoritmus má určenou hloubku procházení tahů a simuluje tahy právě do této nastavené hloubky. Algoritmus nejprve vytvoří list nejlepších tahů, a to metodou GetCandidateMoves. Metoda simuluje tahy na hrací ploše. Podle toho, v jaké fázi hry se nachází, zvolí, zda bude kontrolovat všechna pole, nebo pouze okolí posledního tahu. To zrychluje algoritmus ze začátku, aby nezvažoval nesmyslné tahy na kraji plochy. Když je hra pokročilejší, může si dovolit zamyslet se a projít tahy více do podrobna. Pokud nenalezne žádné ideální tahy, opět je vrácena hodnota zjištěna metodou EvaluateBoard. Z nejlepších tahů potom pomocí alfa-beta pruningu vyřadí slabší větve. Zbylé větve poté prochází, dokud nedojde do maximální nastavené hloubky, nebo dokud nedojde nastavený čas pro kalkulaci.

2.6 Konec hry

Ukončení hry v piškvorkách nastává ve chvíli, kdy některý z hráčů propojí požadovaný počet symbolů v řadě, a to horizontálně, vertikálně nebo diagonálně nebo ve chvíli, kdy hráč položí poslední možný symbol na hrací plochu a na hrací ploše již neexistuje žádný způsob, jak vyhrát hru.

2.6.1 Délka výherní řady

Standardně je požadováno spojení pěti symbolů (tzv. „pět v řadě“), nicméně aplikace umožňuje tuto hodnotu libovolně změnit, takže lze hrát jak klasické Piškvorky (5 v řadě), tak například hru „Tic Tac Toe“ (kde se hraje na hrací ploše 3x3 a výherní řada se skládá ze tří symbolů).

V rámci aplikace je tedy nutné po každém tahu zkontrolovat nově položený symbol (souřadnice X, Y) a spočítat, kolik symbolů stejného typu se nachází směrem doleva/doprava nebo nahoru/dolů, případně v diagonálních směrech. Pokud součet dosáhne požadované délky (5 a více), hráč vyhrává. V případě, že součet dosáhne požadované délky ve více směrech v jednom tahu, hráč obdrží bod za každou výherní řadu.

Při zaplnění hrací plochy bez výherce se vyhodnotí remíza. Pro kontrolu remízy se používá proměnná rowsLeftOnBoard inicializovaná na hodnotu podle vzorce.

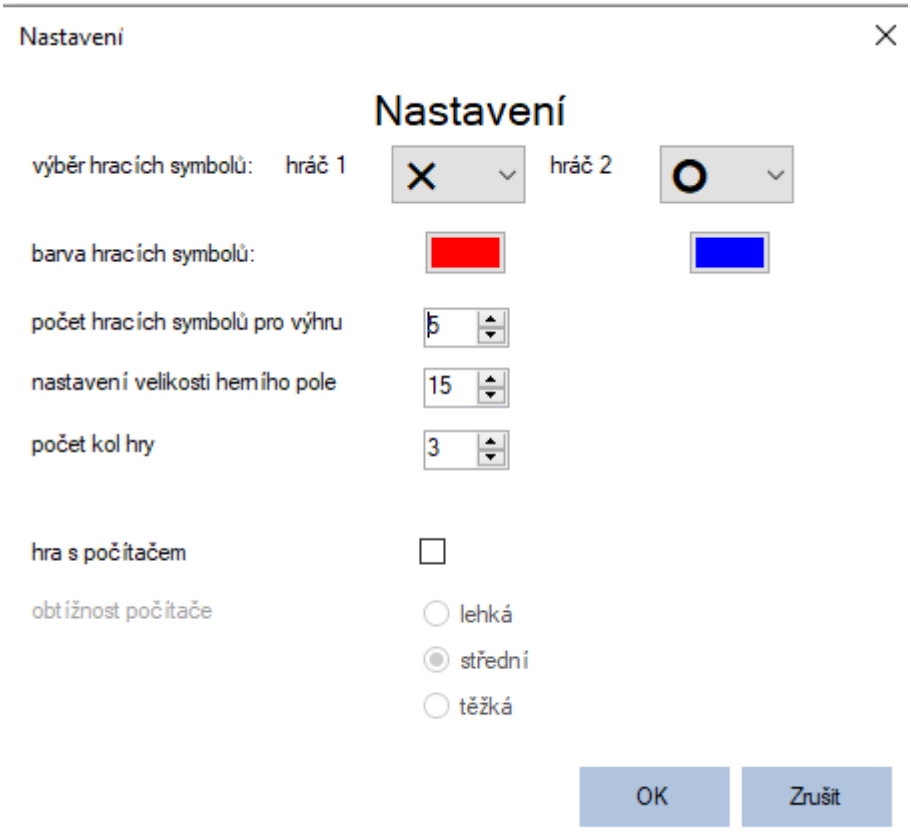
Proměnná značí počet možných výherních řad na hrací ploše. Vždy když dojde k zablokování výhry na hrací ploše, číslo snížíme. Když se počet výher dostane na nulu, nastavíme stav hry na remízu.

Po určení výsledku hra zobrazí hlášení o vítězi a pokud se jedná o hru proti počítači s těžkou obtížností je zapsán výsledek do Historie nejlepších.

Hráči mají volbu hrát větší počet her za sebou, přičemž vyhraje hráč s větším počtem bodů po všech odehraných hrách. V případě více her jsou individuální hry nazývány partiemi. Po odehrání všech partií se opět vypíše hlášení o vítězi.

2.7 Nastavení

Sekce nastavení je klíčovou součástí aplikace, která uživateli umožňuje přizpůsobit si hru podle vlastních preferencí. Ve FormSettings, což je formulář s veškerým nastavením aplikace může uživatel konfigurovat: Symboly hry a jejich barvu, velikost hrací plochy, délku výherní řady, počet kol hry, zda chce hrát proti počítači a obtížnost počítače.

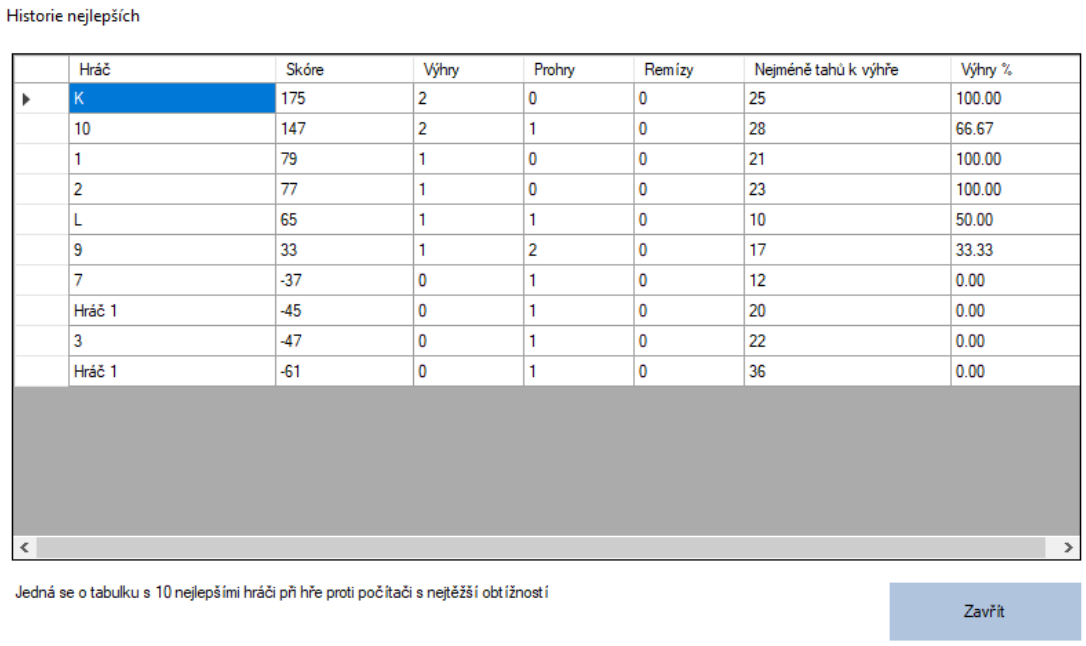


Obr. 24 – Formulář s nastavením hry

Z programátorského pohledu je sekce nastavení řešena jako dialogové okno formuláře, kde uživatel volby potvrdí tlačítkem „Ok,“ Nastavení se vždy zapíše do třídy GameSettings, která se skládá z veřejných vlastností, do kterých se veškerá data zapíší. Poté se ve formuláři FormBoard zapíší přes veřejné vlastnosti a metody do aktivní instance komponenty playingBoard, která se vykreslí se změněným nastavením.

2.8 Historie nejlepších

Aplikace obsahuje tabulku 10 nejlepších hráčů, která je motivací k neustálému zlepšování a dává hře další soutěžní rozměr. Záznamy se však uchovávají pouze z her proti počítači, a to ještě pouze pokud je nastaven na nejtěžší obtížnost, aby se předešlo podvádění nebo zahlcení tabulky jedním uživatelem. Tato tabulka se nazývá „Historie nejlepších“ a uchovává jméno hráče, výsledné skóre, výhry, prohry, remízy, výhry v % a nejkratší počet tahů k výhře.



Obr. 25 – Naplněná tabulka Historie nejlepších

Při přidání nového záznamu se tabulka setřídí podle v sestupném pořadí. Pokud je tabulka plná (10 záznamů) a nový záznam má vyšší skóre než nejnižší v tabulce, je nejnižší odstraněn.

Ukládání probíhá do textového souboru, v tomto případě do XML. Tento formát však není šifrovaný a umožňuje tak přepisování dat běžným uživatelem. K zabezpečení záznamů jsou data zapsána do souboru datového typu, který nejde běžně uživatelem zobrazit či upravit.

Uživatelům se tabulka zobrazuje v okně aplikace formou tabulky pomocí komponenty DataGridView, v níž lze proklikem nebo přehledným zobrazením sledovat, kdo momentálně vede žebříček.

2.9 Uložení a nahrání hry

Při implementaci funkce uložení a nahrání hry v C# Windows Forms využívám zabudované dialogy pro práci se soubory – tedy OpenFileDialog (pro načítání) a SaveFileDialog (pro ukládání). Tyto dialogy poskytují komfortní uživatelské rozhraní pro volbu názvu i umístění souboru, aniž by bylo nutné ručně psát cestu na disku.

SaveFileDialog se v aplikaci vyvolá v okamžiku, kdy uživatel potřebuje uložit rozehranou partii. Ve vybraném dialogu zvolí adresář a název souboru, do nějž se zapíšou všechna důležitá data o stavu hry (konkrétně rozložení symbolů na herní ploše, aktuální hráč, velikost hracího pole a další herní nastavení). Dialog je nastaven tak, aby se vždy otevíral s vybranou lokací ve složce “save“, která se zároveň s prvním uložením do této lokace vytvoří.

OpenFileDialog umožňuje uživateli vybrat soubor, z něhož se načte dříve uložená hra. Aplikace následně zvolený soubor otevře, přečte z něj potřebné hodnoty, vynuluje právě uložené hodnoty a zrekonstruuje herní pole i všechny ostatní parametry hry. Uživatel tak může plynule pokračovat v rozehraném duelu i pokud na hrací ploše rozehrál jinou hru.

Díky těmto dialogům je práce se soubory pro uživatele maximálně intuitivní, jedná se totiž o známé systémové okno, kde lze vybírat složky a soubory tak, jak je uživatel zvyklý z prohlížeče souborů. Volba formátu, do něhož se data ukládají je datový soubor s koncovkou .dat, tak aby běžný uživatel nemohl manuálně číst ani upravovat soubor.

Díky propojení SaveFileDialog a OpenFileDialog s uložením herního stavu do čitelného formátu získá uživatel přehled a volnost v tom, kam a jak často si hru ukládá.

Zároveň se jedná o lehké a rychlé řešení, které nevyžaduje žádnou nadbytečnou infrastrukturu (např. databáze). Jedná se tedy o optimální přístup jak pro uživatele, tak pro mě jako vývojáře.

2.10 Demo

Demo mód v tomto projektu slouží k automatickému přehrání hry, ve které se dva počítačoví hráči střídají v tazích. Hráč spustí hru kliknutím na hrací plochu a poté jen sleduje bez nutnosti zasahovat. Tento režim demonstruje schopnosti umělé inteligence a zároveň umožňuje vidět různé strategie v průběhu hry.

Po kliknutí na tlačítko Demo v hlavním menu se zobrazí formulář s běžnou hrací plochou, na které se však po kliknutí uživatele odehraje hra dvou počítačů nastavených na těžkou a střední obtížnost. Po každém tahu je nastavena krátká prodleva, aby byl průběh hry přirozený a vizuálně přehledný. Hra pokračuje automaticky, dokud nenastane výhra nebo remíza, po které uživatel může proces opakovat nebo se vrátit do menu.

Technicky je demo řízeno pomocí proměnné GameSettings.DemoMode, která se aktivuje při spuštění demo režimu. V průběhu hry je pak na základě této hodnoty automaticky prováděn tah AI pro oba hráče, aniž by se čekalo na vstup uživatele. Tahy jsou prováděny v metodě zpracovávající tahy hráčů, kde po tahu jednoho hráče následuje okamžitě tah druhého.

Hlavním cílem této funkce je demonstrace hry, ukázka možností umělé inteligence a vizuální prezentace pravidel hry.

Závěr

Aplikace disponuje hrou pro dva hráči i pro hráče proti počítače. V aplikaci lze nastavit všechny potřebné požadavky pro úplnou aplikaci (velikost hracího pole, počet symbolů pro výhru, symboly hráčů, obtížnost umělé inteligence, počet kol hry a barvy herních symbolů). Aplikace dále obsahuje historii deseti nejlepších hráčů ve hře proti nejtěžší obtížnosti počítače. Nechybí ani demo, kde po kliknutí uživatele proběhne hra odehraná dvěma počítači. Ve hře je samozřejmě i možnost nahrávat a ukládat rozehranou hru. Pro přívětivý zážitek nechybí responzivní herní plocha, zvýraznění posledního odehraného tahu a zvýraznění výherní řady.

Do budoucna zvažuji aplikaci předělat do Blazor MAUI, což je moderní technologie, jež je součástí .NET frameworku a umožňuje aplikaci spustit skrze webové rozhraní na více platformách včetně mobilních zařízení. Při práci na projektu jsem se ujistil v již získaných znalostech z mého studia, ale také jsem objevil mnoho dalších technologií a tříd v .NET prostředí. Znalosti získané v projektu mi budou platné i v budoucím studiu na MUNI v oboru zaměřeném na vývoj aplikací. Aplikace tedy splňuje zadání práce a zároveň je stále možné ji rozšiřovat a zlepšovat, což je také mým plánem do příštích měsíců.

Seznam použitých zdrojů

[1] WAGNER, Bill. *Prohlídka jazyka C#*. Online. 2024. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp/overview. [cit. 2025-03-22].

[2] IPSEN, Adam. *Top 10 programming languages for 2025*. Online. 7 Nov 2024. Dostupné z: https://www.pluralsight.com/resources/blog/upskilling/top-programming-languages-2025. [cit. 2025-03-22].

[3] MICHÁLEK, Ondřej. *Lekce 1 - Úvod do funkcionálního programování*. Online. 7. 11. 2019. Dostupné z: https://www.itnetwork.cz/programovani/haskell/uvod-do-funkcionalniho-programovani. [cit. 2025-03-24].

[4] KERI CORE ACADEMY. *Programovací jazyk C#: Vše co potřebujete vědět v roce 2024!*. Online. © 2025. Dostupné z: https://kericore.academy/programovaci-jazyk-c-sharp-potrebujete-vedet-2024/. [cit. 2025-03-26].

[5] MICROSOFT. *Visual Studio: Integrované vývojové prostředí (IDE) a editor kódu pro vývojáře softwaru a týmy*. Online. © 2025. Dostupné z: https://visualstudio.microsoft.com/cs/#vs-section. [cit. 2025-3-22].

[6] MICROSOFT. *Visual Studio: Integrované vývojové prostředí (IDE) a editor kódu pro vývojáře softwaru a týmy*. Online. © 2025. Dostupné z: https://visualstudio.microsoft.com/cs/vs/ [cit. 2025-3-26].

[7] GITHUB. *About GitHub and Git - GitHub Docs*. Online. GitHub. Dostupné z: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. [cit. 2025-03-26].

[8] MICROSOFT. *Přehled rozhraní .NET Framework*. Online. 2024. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/framework/get-started/overview. [cit. 2025-03-26].

[9] STUDENT CYBER GAMES. *O soutěži - pIšQworky*. Online. PIšQworky. © 2025. Dostupné z: https://pisqworky.cz/o-soutezi. [cit. 2025-03-26].

[10] *Piškvorky*. Online. In: Wikipedie. Stránka byla naposledy editována 17. 3. 2025 v 16:39. Dostupné z: https://cs.wikipedia.org/wiki/Piškvorky. [cit. 2025-03-26].

[11] ALLIS, Louis Victor. *Go-Moku and Threat-Space Search*. Disertace. Maastricht, Nizozemsko: University of Limburg, 1994. [cit. 2025-03-26]

[12] *Gomoku*. Online. In: Wikipedie. Stránka byla naposledy editována 30. 8. 2023 v 11:55. Dostupné z: https://cs.wikipedia.org/wiki/Gomoku. [cit. 2025-03-27].

[13] ČESKÁ FEDERACE PIŠKVOREK A RENJU. *O piškvorkách neboli gomoku*. Online. Česká federace piškvorek a renju. © 2025. Dostupné z: http://www.piskvorky.cz/federace/o-piskvorkach-neboli-gomoku-2/. [cit. 2025-03-27].

[14] ČESKÁ FEDERACE PIŠKVOREK A RENJU. *Historie gomoku a renju*. Online. Česká federace piškvorek a renju. © 2025. Dostupné z: http://www.piskvorky.cz/clanky/zajimavosti-ze-sveta-piskvorek-a-renju/historie-gomoku-a-renju/. [cit. 2025-03-27].

[15] *Go*. Online. In: Wikipedie. Stránka byla naposledy editována 19. 1. 2025 v 10:13. Dostupné z: https://cs.wikipedia.org/wiki/Go_(desková_hra). [cit. 2025-03-27].

[16] *Umělá inteligence*. Online. In: Wikipedie. Stránka byla naposledy editována 27. 3. 2025 v 06:23. Dostupné z: https://cs.wikipedia.org/wiki/Umělá_inteligence. [cit. 2025-03-28].

[17] ZAORAL, Karel. *Lekce 1 - Úvod do AI.* Online. 25. 4. 2024. Dostupné z: https://www.itnetwork.cz/ai/zaklady/uvod-do-ai. [cit. 2025-03-28].

[18] KÜHR, Tomáš. *Algoritmus Minimax*. Online. Dostupné z: https://www.inf.upol.cz/downloads/studium/PS/minimax.pdf. [cit. 2025-03-28].

[19] *Minimax (algoritmus)*. Online. In: Wikipedie. Stránka byla naposledy editována 7. 12. 2023 v 08:43. Dostupné z: https://cs.wikipedia.org/wiki/Minimax_(algoritmus). [cit. 2025-03-28].

[20] HNILICA, Jan. *Lekce 9 - Minimax – Piškvorky proti uživateli – Úvod*. 28. 3.

2024. Dostupné z: https://www.itnetwork.cz/algoritmy/rekurze/minimax-piskvorky-proti-uzivateli-uvod [cit. 2025-03-28].

[21] *Alfa-beta ořezávání*. Online. In: Wikipedie. Stránka byla naposledy editována 5. 10. 2023 v 02:29. Dostupné z: https://cs.wikipedia.org/wiki/Alfa-beta_ořezávání. [cit. 2025-03-28].

[22] *Desková hra*. Online. In: Wikipedie. Stránka byla naposledy editována 7. 3. 2025 v 14:00. Dostupné z: https://cs.wikipedia.org/wiki/Desková_hra. [cit. 2025-03-28].

[23] SLANINÁK, Adam, 2022. *Piškvorky hraju už šest let. Je to skvělý trénink na paměť, říká šampión XO21 Jakub Horák*. Online. 20.10.2022. Dostupné z: https://pisqworky.cz/novinky/2723-piskvorky-hraju-uz-sest-let-je-to-skvely-trenink-na-pamet-rika-sampion-xo21-jakub-horak. [cit. 2025-03-28].

[24] GOMOCUP,. *Download Gomoku Artificial Inteligence (AI)*. Online. Gomocup - the Gomoku AI Tournament. Dostupné z: https://gomocup.org/download-gomoku-ai/. [cit. 2025-03-28].

[25] APETÝT, 2023. *Hra, která vyžaduje trpělivost, vytrvalost i logické myšlení. Piškvorky rozvíjí mozek ve všech směrech*. Online. 8. listopadu 2023. Dostupné z: https://www.mujrozhlas.cz/apetyt/hra-ktera-vyzaduje-trpelivost-vytrvalost-i-logicke-mysleni-piskvorky-rozviji-mozek-ve-vsech. [cit. 2025-03-28].

Seznam obrázků

Seznam příloh

Prázdná šablona maturitní práce