

Střední průmyslová škola Třebíč

Maturitní práce

Pexeso

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITA4

Školní rok: 2024/2025 Jakub Černý

Zadání práce

ABSTRAKT

Tématem této maturitní práce je vytvoření známe deskové hry pexeso. Hlavní účelem projektu je nabídnout zábavný zážitek prostřednictvím jednoduchého uživatelského rozhraní. Cílem hry je najít nejvíce dvojic stejných obrázků. Práce popisuje proces tvorby této hry pomocí programovacího jazyku C# a systému GitHub.

KLÍČOVÁ SLOVA

maturitní práce, pexeso, C#, GitHub, hra

ABSTRACT

The topic of this graduation thesis is the creation of the well-known board game Pexeso. The main function of the game is to provide the users with fun and enjoyable experience through simple graphics. The goal of the game is to find the most pairs of matching images. The work describes the development process of the project using C# programming language and GitHub.

KEYWORDS

graduation thesis, pairs, memory, C#, GitHub, game

PODĚKOVÁNÍ

Děkuji Mgr. Andrei Odehnalové za užitečné rady při vypracovávání maturitní práce.

V Třebíči dne 16. 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 16. března 2025

podpis autora

Obsah

[Úvod 8](#_Toc192504780)

[1 Teoretická část 9](#_Toc192504781)

[1.1 Vývoj a vliv deskových her 9](#_Toc192504782)

[1.2 Benefity deskových her 11](#_Toc192504783)

[1.3 Historie pexesa 13](#_Toc192504784)

[1.4 Pravidla hry pexeso 14](#_Toc192504785)

[1.5 Téma mého pexesa 15](#_Toc192504786)

[1.6 Microsoft 15](#_Toc192504787)

[1.7 Visual Studio 16](#_Toc192504788)

[1.8 .NET Framework 17](#_Toc192504789)

[1.9 C# 18](#_Toc192504790)

[1.10 Windows Forms 19](#_Toc192504791)

[1.11 GitHub a Git 20](#_Toc192504792)

[2 Praktická část 21](#_Toc192504793)

[2.1 Třídy 21](#_Toc192504794)

[2.2 Třída GameBoard (Herní pole) 22](#_Toc192504795)

[2.2.1 Konstruktor Třídy 22](#_Toc192504796)

[2.2.2 LoadImages (Načíst obrázky) 22](#_Toc192504797)

[2.2.3 SetupLayout (Sestavit herní pole) 23](#_Toc192504798)

[2.2.4 CreateCardLabel (Vytvořit hrací kartu) 24](#_Toc192504799)

[2.2.5 PlaceCards (Rozmístit karty) 24](#_Toc192504800)

[2.2.6 InitializeBoard (Inicializovat herní pole) 25](#_Toc192504801)

[2.2.7 GetCardImage (Získat obrázek karty) 25](#_Toc192504802)

[2.2.8 FlipCardFront (Otočit kartu vzhůru) 25](#_Toc192504803)

[2.2.9 GetBackImage (Získat zadní obrázek karty) 25](#_Toc192504804)

[2.2.10 FlipCardBack (Otočit kartu zpátky) 26](#_Toc192504805)

[2.3 Třída GameLogic (Herní logika) 26](#_Toc192504806)

[2.3.1 Konstruktor GameLogic 26](#_Toc192504807)

[2.3.2 Enum GameState (Výčtový typ) 27](#_Toc192504808)

[2.3.3 OnCardClicked (Při kliknutí na kartu) 27](#_Toc192504809)

[2.3.4 OnTimerTick (Při tikání časovače) 29](#_Toc192504810)

[2.3.5 GetRight (Získat správně) 29](#_Toc192504811)

[2.3.6 ComputerTurn (Tah počítače) 30](#_Toc192504812)

[2.3.7 WinnerCheck (Kontrola výherce) 32](#_Toc192504813)

[2.4 Třída GameScoreManager (Správce herního skóre) 32](#_Toc192504814)

[2.4.1 Add, Get a Set (Přidat, získat a nastavit) 32](#_Toc192504815)

[2.4.2 GetSortedScores a EndScore (Získat seřazené skóre a Konečné skóre) 33](#_Toc192504816)

[2.5 Třída SoundManager (Správce zvuku) 33](#_Toc192504817)

[2.5.1 Konstruktor 33](#_Toc192504818)

[2.5.2 LoadAudio (Načíst zvuky) 34](#_Toc192504819)

[2.5.3 PlayAudio (Spustit zvuk) 34](#_Toc192504820)

[2.5.4 StopAudio a Dispose (Zastavit zvuk a Uvolnit) 34](#_Toc192504821)

[2.5.5 Play Metody (Spuštění samostatných zvuků) 34](#_Toc192504822)

[2.6 Třída GameSave (Uložená hra) 34](#_Toc192504823)

[2.7 Třída GameSaveManager (Správce uložení hry) 35](#_Toc192504824)

[2.7.1 SaveGame a LoadGame (Uložení hry a Načtení hry) 35](#_Toc192504825)

[2.8 Třída ScoreData (Informace o Skóre) 36](#_Toc192504826)

[2.9 Třída GameScoreSaveManager (Správce uložení skóre hry) 36](#_Toc192504827)

[2.10 Formuláře 37](#_Toc192504828)

[2.11 StartingMenu (Hlavní menu) 37](#_Toc192504829)

[2.12 GameSettings (Nastavení hry) 38](#_Toc192504830)

[2.13 NewGame (Nová hra) 39](#_Toc192504831)

[2.13.1 Konstruktor 39](#_Toc192504832)

[2.13.2 GetNames (Získat jména) 39](#_Toc192504833)

[2.13.3 ShowScore (Ukázat skóre) 40](#_Toc192504834)

[2.13.4 SaveGame (Uložit hru) 40](#_Toc192504835)

[2.13.5 LoadGame (Načíst hru) 41](#_Toc192504836)

[2.13.6 RestoreFromGameSave (Obnovit ze zálohy hry) 42](#_Toc192504837)

[2.13.7 EndGame (Konec hry) 42](#_Toc192504838)

[2.14 Score (Skóre) 43](#_Toc192504839)

[2.14.1 Konstruktor 43](#_Toc192504840)

[2.14.2 InitializeDataGridView a InitializeComboBox (Inicializace DGV a ComboBoxu) 44](#_Toc192504841)

[2.14.3 LoadScoreData a DisplayData (Načíst informace o skóre a Zobrazit informace) 44](#_Toc192504842)

[2.14.4 buttonFilter\_Click a buttonClear\_Click (Tlačítko filtrování a vymazání) 44](#_Toc192504843)

[2.15 Psaní úvodu 47](#_Toc192504844)

[2.16 Struktura odstavců 47](#_Toc192504845)

[2.17 Obrázky, tabulky a rovnice 48](#_Toc192504846)

[2.18 Řazení a struktura kapitol 50](#_Toc192504847)

[2.19 Závěr 50](#_Toc192504848)

[2.20 Seznam použitých zdrojů 50](#_Toc192504849)

[2.21 Seznam použitých symbolů a zkratek 50](#_Toc192504850)

[2.22 Seznamy použitých obrázků a tabulek 51](#_Toc192504851)

[2.23 Seznam příloh 51](#_Toc192504852)

[Závěr 52](#_Toc192504853)

[Seznam použitých zdrojů 53](#_Toc192504854)

[Seznam použitých symbolů a zkratek 55](#_Toc192504855)

[Seznam obrázků 56](#_Toc192504856)

[Seznam tabulek 57](#_Toc192504857)

[Seznam příloh 58](#_Toc192504858)

Úvod

Už v mladém věku jsem si deskové hry velice oblíbil. Hrával jsem je převážně s ostatními členy své rodiny. A přesto, že mě vždy nechávali vyhrát, abych se nevztekal, tak mi láska k deskovým hrám stále zůstala.

Ve svých deseti letech jsem vyměnil deskové hry a knížky za počítač, kde se díky videohrám zrodila moje vášeň k programování. Můj nástup na SPŠT moji zálibu ještě více prohloubil.

Z programování mě nejvíce zaujal vývoj počítačových her, a to především kvůli nostalgické lásce k nim. Her nemám vytvořených mnoho, ale jejich vývoj mě neskutečně bavil a přinesl mi spoustu nových a cenných zkušeností. S třemi spolužáky jsme v rámci řízení projektů založili amatérské herní studio, kde jsme vytvořili dvě hry. Díky této velice obohacující zkušenosti jsem se naučil používat programování v praxi.

Práce byla zadána ve stejném programovacím jazyce, ve kterém jsme vyvíjeli naše menší hry.

Přesně z těchto několika důvodů jsem si vybral tuto maturitní práci.

Hra pexeso bude tvořena pomocí programovacího jazyka C# v prostředí Visual Studio a verzovacího prostředí GitHub.

# Teoretická část

Tato část maturitní práce je určena k představení použitých technologií. Také se v ní nachází stručný popis deskových her, pexesa, jeho pravidel a zvoleného tématu.

## Vývoj a vliv deskových her

Dnešní trh s deskovými hrami nabyl největší popularitu za celou dobu svojí existence.[1] Může tomu vděčit za dnešní možnost vše přetvořit do digitální podoby. Dnes každý nápad nebo vylepšení je možno předělat do aplikace nebo webové stránky a díky tomu je výběr mezi hrami tak neskutečně velký, že máme možnost hrát každý den novou hru. Obrovskou výhodou online světa je, že nepotřebujeme být se spoluhráči v jedné místnosti, ale můžeme hrát na dálku, klidně i přes půl světa. Spoluhráče dokonce nemusíme znát, stačí jednoduše kliknout na tlačítko a na hrací ploše se objeví neznámí hráči nebo umělá inteligence.

Nebylo tomu tak však vždy. Naši dávní historičtí předkové neměli takovou škálu výběru a informací. Vystačili si s jednoduchými deskovými hrami, které sami po domácku vyrobili. Každá země má své specifické hry, ať už originální nebo převzaté a upravené od jiných kultur. Spousta se jich dochovala v hrobkách nebo v záznamech, díky čemuž lze nahlédnout do dávné historie. Za první prototyp deskové hry jsou považovány hrací kostky starší než samotné písmo, které byly nalezeny archeology na pohřebišti v jihovýchodním Turecku. Dané pohřebiště bylo vytvořeno přibližně v roce pět tisíc před Kristem, což se zdá neuvěřitelné. Na místě bylo nalezeno asi padesát vyřezaných a namalovaných kamínků, které sloužily jako kousky hry. Podobné nálezy se vyskytují po celém středním východě, což naznačuje, že deskové hry jako takové, se pravděpodobně začaly hrát právě tady. [1]

Deskové hry hráli i starověcí Egypťané. První záznamy o jejich populární hře byly datovány do roku 3 500 př.n.l. Hra jménem Senet připomíná předchůdce šachů. Cílem hry je přejít kamennými figurkami herní pole a vyhnout se jeho nástrahám. Obyvatelé dávného Egypta brali hru nejen jako zábavu, ale také jako spirituální a náboženský nástroj, který simuloval cestu do světa mrtvých. Pro Egypťany byl Senet možností, jak ovlivnit ještě před smrtí svůj posmrtný život. Senet byl součástí pohřební výbavy mnoha faraonů a byl nalezen v několika slavných hrobkách. [1] [2]

<https://www.cestyarcheologie.cz/single-post/hry-v-prubehu-veku-4-senet>

Obsah obrázku Maketa, modrá, interiér

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.1 Hrací plocha hry Senet

https://commons.wikimedia.org/wiki/File:SenetBoard-InscribedWithNameOfAmunhotepIII\_BrooklynMuseum.png

Nejstarší deskovou hrou je populární hra Go, která je stará přibližně 4 000 let. Hru i dnes pravidelně hraje několik desítek miliónů hráčů, a to převážně v Asii, odkud hra také pochází. V Japonsku byla tak vážená, že v sedmnáctém století vláda založila čtyři školy určené k výuce této hry. Hraní Go bylo v této době uznáváno jako povolání. Hra byla také populární v Číně a Koreji. Kvůli druhé světové válce se později rozšířila do celého světa.[3]

Šachy, nejikoničtější a nejznámější desková hra světa pochází z šestého století našeho letopočtu. Hra byla nejdříve hraná v Indii, poté se se díky cestám muslimů dostala i do západního světa. Kolem patnáctého století se v Itálii a Španělsku zrodila pravidla hry tak, jak je známe. Začaly se psát knihy o pravidlech a umění šachu.[4] https://pxhere.com/en/photo/1521263

Obsah obrázku Sálové hry a sporty, desková hra, Stolní hra, Hry

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.2 Šachovnice

Moderní turnaje a soutěže započaly v devatenáctém století, kdy se také odehrál první světový turnaj, který je i v dnešní době pravidelně pořádán. Později vznikla mezinárodní organizace, která spravuje světové turnaje a další soutěže. [4]

Od začátku roku 2 000 začaly být populární počítačové analýzy, které naprosto změnily přístup ke hře. Hráči mohli zdokonalovat své taktiky a získat tím výhodu nad protihráči. Online prostředí šachů je dnes také velice populární, každý den hrají šachy miliony hráčů. [4]

## Benefity deskových her

Z historie deskových lze vyvodit, že byly nedílnou součástí životů našich předků. Byla to jedna z mála možností jejich zábavy. Podmínky byly omezené, což vybízelo k vysoké míře kreativity a tvorbě nových originálních her. Hry můžeme dělit do různých žánrů – například podle tématu, materiálu nebo počtu hráčů.

Drtivá většina deskových her se hraje ve více hráčích, ale samozřejmě existuje několik výjimek. Je to skvělý způsob, jak strávit večer se svými blízkými – ať už s malými sourozenci, partnerem nebo celou rodinou. Hry mohou být dobrým rozptýlením a zábavou. [5]

Schopnost spolupráce a komunikace, které jsou často potřebné k dosažení vítězství, mohou být výhodou pro každodenní i profesní život. Deskové večery jsou také velice dobrým způsobem, jak navázat nová přátelství, zocelit ta stará, nebo posílit rodinné pouto. [5]

Nutnost pamatovat si komplexní pravidla, svoje i protihráčovi tahy a cíl hry, je velice náročné na paměť. Některé deskové hry se na záměrné procvičení paměti soustředí. Jejich hraní se může stát v životě velikou výhodou, převážně když si nebudete schopni vzpomenout, kde jste nechali ležet klíče. Hráč musí také umět vymýšlet nové strategie, díky kterým bude schopen přelstít ostatní protihráče, případně samotnou hru. Tohle strategické a kritické myšlení zapojuje části mozku, které je nutné pravidelně trénovat. [5]

Různorodost konečného výsledku na konci hry, drtivá výhra až suverénní prohra jsou cennou lekcí pro naše emoce. Udržet při prohře kontrolu nad svými emocemi je složitým úkolem převážně pro soutěživé povahy a malé děti, které chtějí za každou cenu vyhrát.[5]

Nátlak z každodenního života je velice stresující záležitost, která se týká každého z nás. Deskové hry jsou vědecky potvrzeným způsobem, jak zklidnit mysl a odprosit se na chvíli od stresujícího života. Hra vyžaduje plnou soustředěnost, bez které není možné vyhrát. Pokud se budeme věnovat pouze hře, dostaví se pocit uvolnění od zbytku světa a našich problémů. Úspěšná výhra vytvoří pocit úspěchu, který dokáže zlepšit náladu a snížit míru stresu. [5]

Velice zajímavým benefitem deskových her je, že dokáže zabránit nebo alespoň zpomalit spoustě vážných nemocí, jako jsou Alzheimerova choroba a demence. Zapojení mozku při hře trénuje části mozku starající se o kognitivní funkce, které se se zvyšujícím věkem zhoršují a jejich cvičení může být skvělým pomocníkem. [5]

## Historie pexesa

Desková hra pexeso je v Česku a na Slovensku populární již více než padesát let. Jejímu vzniku vděčíme již zesnulému Zdeňkovi Princovi, který údajně dostal nápad na sestrojení této legendární hry přímo v katedrále sv. Víta na Pražském hradě. [6]

Autor se v této době podílel na tvoření mozaiky Kristova křtu, na které spolupracoval s mnoha malíři a sochaři. V tomto kreativním prostředí se mu dostala i spousta času na jeho projekt, který měl nejdříve nést název po jeho projektu – obrázková mozaika. [6]

Zdeněk Princ nastoupil do nakladatelství Pressfoto, kde mu byla vystavena nabídka vytvoření hry, jejímž stvořením měl pomoci zvětšit výnosy a činnost nakladatelství. Návrh obrázkové mozaiky složené z 32 párů byl nakladatelstvím přijat kladně, název však byl problémem. Pan Princ tedy zkrátil název „Pekelně se soustřeď“ na námi známý akronym „Pexeso“, se kterým vedení nakladatelství bylo už spokojené. Autor se inspiroval československou televizí, kde se spojení Pekelně se soustřeď objevil jako název televizní soutěže. [6]

První vydání deskové hry sklidilo obrovský úspěch. Vydání mělo ilustraci hrdinů z filmů o Vinnetouovi. Kombinace jednoduchých pravidel a ilustrace slavných hrdinů určitě pomohla slávě a popularitě hry mezi všemi věkovými skupinami. Hra se brzy stala nutnou součástí každé domácnosti. [6]

Obsah obrázku oblečení, území, venku, osoba

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.3 Fotka hrdiny Vinettou

https://commons.wikimedia.org/wiki/File:Karl-May-Spiele\_Winnetou\_I\_Blutsbr%C3%BCder\_%282%29\_16.jpg

## Pravidla hry pexeso

Hra pexeso je převážně doporučena pro dva hráče, člověk ji však může hrát také sám nebo až v šesti hráčích. Proto je hra skvělou aktivitou pro volný čas s přáteli, rodinnou nebo dětmi.

Hrací pole tvoří několik párů karet se stejným obrázkem, základní počet karet bývá třicet dva, ale variací je více. Hráč, který je na řadě otočí dvě karty lícem nahoru podle jeho volby. Pokud jsou karty stejné, karty si odebere k sobě a dostává bod. Pokud karty se obrázky liší, otáčí je zpět lícem dolu. Hráč s nejvíce body získává titul výherce.

## Téma mého pexesa

Témata pexes, která můžete zakoupit v hračkářství nebo v jakémkoliv jiném obchodě se velice liší. Témat na výběr je spousty – od barev přes pohádkové bytosti až po mnoho dalšího. Zvoleným tématem tohoto pexesa je ovoce.

Návyky, na které si zvykneme v dětství, nám v dospělosti setrvávají, a to platí i o stravování. V dnešní době je velice těžké se stravovat správně a vyhýbat se nezdravému jídlu.

Ovoce patří do kategorie zdravého jídla a obsahuje mnoho vitamínů, vody a vlákniny, které v dnešní průměrné stravě chybí. Ovoce díky svému nízkému počtu kalorií taky pomáhá s hubnutím a brání nástupu obezity. [7]

S dodržováním doporučené dávky ovoce, která činí přibližně 400 gramů, má velká část společnosti problém. [8]

Existuje studie, kterou provedli nizozemští vědci, která zjistila, že u dětí se po hraní paměťové hry s ovocem drasticky zvýšila chuť na něj a jeho příjem.[9]

Přesně proto je tématem této hry ovoce. Třeba se díky ní někomu trochu zlepší stravovací návyky.

## Visual Studio

Visual Studio je ve vlastnictví společnosti Microsoft, kterou bylo vyvinuto a do dnešního dne je i spravováno a aktualizováno. Patří k nejoblíbenějším vývojovým prostředím mezi programátory, a to především díky své přehlednosti a variabilitě. V tomto prostředí lze vyvíjet spoustu různých druhů projektů, ať už webové aplikace, konzolové aplikace, desktopové aplikace a mnoho dalších. Vývojové prostředí je možno používat na operačních systémech Windows a macOS. [10]

Ani možnost volby programovacího jazyka není omezená, studio podporuje oblíbené jazyky pro back-end i kompletní vývojáře. Mezi ně patří HTML, CSS, JavaScript, JSON, LESS, SASS, PHP, Python nebo jazyk C# s technologií ASP.NET. [11]

Přesný počet podporovaných jazyků je třicet šest. [12]

Visual studio se dělí na tři edice. Při zpracování této maturitní práce byla použita verze Community. Tato edice byla vydána v roce 2014. Jedná se o jedinou edici dostupnou zdarma. Pro společnosti s vyšším počtem zaměstnanců je určitým způsobem omezená, pro individuální vývojáře však nikoli. Její hlavní funkcí je zprostředkovávání přístupu k několika tisícům knihoven a rozšíření, zároveň také poskytuje plnou podporu populárních jazyků. [13]

Obsah obrázku snímek obrazovky, software, Multimediální software, Grafický software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.4 Snímek z prostředí Visual Studia

## .NET Framework

.NET Framework, vyvinut a spravován společností Microsoft, je softwarová běhová platforma určená pro budovaní webových, desktopových a mobilních aplikací. Platformu tvoří dvě hlavní časti. Common Language Runtime, který spravuje paměť a stará se o spuštěné aplikace. Další částí je knihovna tříd, která vývojářům poskytuje nepřeberné množství opakovatelného kódu, který mohou zakomponovat do svých aplikací. [14]

Největší výhodou .NET Frameworku je jeho podpora velké škály výběru programovacích jazyků. Vývojář si může vybrat mezi programovacími jazyky tak, aby jeho volba nejlépe seděla danému problému, který se snaží vyřešit. Na volbě nesejde, vývojář stále bude moci využít stejné funkce a nástroje, které .NET Framework podporuje. [15]

Typů projektů na výběr je velká škála, což je pro vývojáře pracující na různých projektech značnou výhodou. Framework také obsahuje podporu prvků, které zlepšují bezpečnost, výkon a spolehlivost aplikací. Jelikož je .NET Framework vyvíjen společností Microsoft, je navržen tak, aby byl kompatibilní s dalšími nástroji, jako jsou nástroje SQL Server, SharePoint, Office, které jsou taktéž vyvinuty společností Microsoft.[15]

## C#

C# je moderní programovací jazyk z rodiny jazyků C, což znamená, že je podobný ostatním jazykům z rodiny, jako jsou C++ a Java. Byl vyvinut společností Microsoft v roce 2000 a běží na platformě .NET Framework. C# je jedna z nejlepších voleb ze všech programovacích jazyků na celém světě, což také odpovídá jeho popularitě. Je ideální pro vývoj Windows aplikací, vývoj her za pomocí Unity, mobilních i webových aplikací a služeb. Od verze vydaní verze .NET Core, lze C# aplikaci spustit na operačních systémech macOS a Linux. [16]

Obsah obrázku text, snímek obrazovky, software, Písmo

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.5 Ukázka kódu v jazyce C#

## Windows Forms

Windows formuláře neboli „WinForms“, jsou součástí knihovny pro tvorbu jednoduchého uživatelského rozhraní převážně pro stolní počítače. Windows formuláře poskytují přístup k velkému množství jednoduchých grafických prvků, které mohou vývojáři používat. Prvky jsou vytvořeny tak, aby vývojář mohl velice jednoduše pomocí vlastností a událostí nastavit jejich vzhled a chovaní. Visual Studio poskytuje pro tuto knihovnu vizuálního návrháře, který vývoj ještě více svojí jednoduchostí a Drag and Drop systému zjednodušuje. Mezi základní prvky patří například tlačítka, textová okna, ListBoxy a mnoho dalšího. [17]

Obsah obrázku text, snímek obrazovky, software, počítač

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 1.6 Ukázka grafického vývojáře ve Visual Studiu

## GitHub a Git

Git je verzovací systém, který zaznamenává změny ve složkách. Je převážně používán při práci v týmu, a to zejména vývojářském. Umožňuje více vývojářům najednou pracovat na stejných souborech. To je umožněno tak, že si vývojář udělá „pull“, tedy kopii z hlavního uložiště, kterému se říká „master branch“, na své lokální uložiště. Následně provede změny na svém lokálním uložišti a ty vloží do uložiště hlavního, čemuž se říká „commit“. Díky Gitu je velice jednoduché změny provádět a případně vrátit. [18]

GitHub je cloudová platforma a její hlavní funkcí je sdílení a ukládaní projektů. Projekty se ukládají v repositářích, což umožňuje několik výhod. Lze si tímto způsobem vytvořit portfolio a předvést ostatním svoji práci. V GitHubu je také velice jednoduché se orientovat a spravovat zdrojový kód. Další vývojáři mohou zhodnotit projekty ostatních a navrhnout zlepšení. Nejdůležitější výhodou je práce na společném projektu bez obavy zničení práce ostatních.[19]

# Praktická část

Cílem části této dokumentace je přiblížit **praktickou strukturu** a **implementaci hry** Pexesa. Projekt je rozdělen do několika částí, a to pro **větší přehlednost** a **omezení duplicity kódu**. Samotný kód je rozdělen do **osmi tříd** a **čtyř formulářů**. Výhodou toho rozdělení je, že při nefunkčnosti, opravě či optimalizaci jedné částí kódu bude zbytek projektu fungovat.

Obsah obrázku text, snímek obrazovky, software, Počítačová ikona

Popis byl vytvořen automaticky

Obr. 2.1 Rozdělení souborů hry

## Třídy

Každá třída nese název podle toho, co k formulářům přidává. Například třída, která je **správcem herního pole**, nese jméno spojené s ním. Třídy rozšiřují formuláře o svůj kód a zároveň zanechávají **čistotu kódu**. Obsah obrázku text, snímek obrazovky, software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.2 Příkladná ukázka kódu Třídy

## Třída GameBoard (Herní pole)

Třída *GameBoard* má na starosti **inicializaci herního pole** a **otáčení** **karet** v průběhu hry.

### Konstruktor Třídy

Díky konstruktoru se předávají dané proměnné, které třída potřebuje. *Konstruktor* si vyžádá celočíselný počet karet, jenž si hráč vybere v nastavení hry. Dané číslo určuje velikost **hracího pole** a **množství karet**. Další vyžádanou proměnnou je *TableLayoutPanel*, který je součástí knihovny **Windows Formulářů**. Je to prvek, který svůj obsah uchovává v řádcích a sloupcích. Počet mřížek se může dynamicky měnit a není pevně daný. [20]

Obsah obrázku text, snímek obrazovky, řada/pruh, diagram

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.3 Ukázka TableLayoutPanelu

### LoadImages (Načíst obrázky)

*LoadImages* je prostředníkem mezi herním polem a soubory s obrázky.

Vytvoří se nový *List* obrázků, do nějž se budou ukládat **obrázky** karet. Proměnná *backImageId*, která určuje **id obrázku** u karty otočené **lícem dolů**, se nastaví na **mínus prvou**. Hra pomocí třídy *Assembly* projede **všechny soubory**, které jsou uloženy v předem **staticky nastavené složce** projektu.

Vybere **jenom** soubory s příponou *png*, což jsou **obrázky karet**. **Každý obrázek** se otevře pomocí *proudu* *dat* *Stream*. Pokud se čtení proudu **povedlo** a je v něm obsažen **správný** **soubor**, dojde ke kontrole, zda obrázek **není** **zadní** **stranou** karty. Pokud **je**, tak se obrázek **uloží** do **globální** **proměnné** *backImage*. V případě, že je tomu **naopak**, tak se **uloží** do **seznamu** **obrázků**.

Při ukládání do seznamu, se zároveň do **dalšího** **seznamu** s názvem cardImagesIds **uloží** **číslo**. Tento seznam obsahuje čísla, která **představují** **odkaz** na **položky** v **seznamu** **obrázků**. Každý obrázek má svoje **identifikační** **číslo**. V logice hry se bude pracovat právě s tímto *Listem* **čísel**, kvůli **rychlejšímu** **běhu** **kódu** než při verzi s obrázky. Metoda je ošetřena **podmínkou**, zda je počet obrázků menší než osmnáct, což je jejich maximální počet. Podmínka zjistí, zda se načetly všechny obrázky.

Obsah obrázku ovoce, snímek obrazovky, Multimediální software, Grafický software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.4 Jeden z obrázků hry

### SetupLayout (Sestavit herní pole)

Tato metoda nastavuje už zmíněný mřížkovaný prvek *TableLayoutPanel*, který představuje **herní** **pole**. Nastaví se počet sloupců a řádků na převzatý **počet** **karet** z *konstruktoru* třídy. Následně se vymaže veškerý obsah a styly pole, což umožní cyklem *for* nastylovat prvek. Každý sloupec a řádek dostane **procentuální** **velikost** z celkové velikosti panelu. Například pokud má mřížka 4 sloupce, tak každý sloupec bude mít 25 % celkové velikosti pole.

### CreateCardLabel (Vytvořit hrací kartu)

Cílem této metody je vytvořit novou **kartu** neboli prvek *Label*. Metoda přijímá parametr celého čísla *tagValue*, ten se uloží do vlastnosti prvku jménem *Tag*. Vlastnost umožňuje velice jednoduše rozpoznat karty od sebe, každá má totiž *Tag* jiný.

Obrázek *Labelu* se na začátku hry nastaví na **obrázek zadní strany karty**, což vytváří efekt **otočení** všech karet. Také se určí jeho velikost, styly a vyplnění celého prostoru mřížky. Připojí se událost *Click*, což znamená, že hráč **kliknutím** na kartu spustí **část** **kódu** v této události. Událost vyvolává *asynchronního* *delegáta* *CardClicked*, který svoji implementaci má ve **třídě** s **herní logikou**. Následně metoda **vrací** **vytvořenou** **kartu**. Obsah obrázku počítač, snímek obrazovky, software, Operační systém

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.5 Ukázka Labelu ve Formuláři

### PlaceCards (Rozmístit karty)

Metoda *PlaceCards* **rozmístí** vytvořené **prvky** *Labelů* po **hracím** **poli**. Pracuje se s parametrem *isLoading*, který značí, zda se načítá už uložená hra, nebo se tvoří nová. Podle hodnoty v parametru se rozmístí karty **dvěma** různými způsoby.

V obou případech se nejdříve inicializuje **nový** **seznam** *hiddenLabels*, který je využíván pro uložení hodnoty *Tagů* karet, **které ještě nebyly otočeny**. S tímto seznamem se pracuje v herní logice.

Pokud se načítá již **uložená** hra, tak se cyklem *for* projedou všechny mřížky **hracího** **pole** a do každého se vytvoří **nová** **karta** s celočíselnou hodnotou *Tagu* **nula**. Následně se karta přidá do **hracího** **pole** a její *Tag* se uloží do **seznamu** **neotočených** karet.

V případě, že se načítá hra **nová**, tak je logika odlišná. Nejdříve se vytvoří nový pomocný *List* **icons** pro uložení **celých čísel**. Následně cyklem *for*, ve kterém je deklarována proměnná na celé číslo nula, projedeme **polovinu** všech vytvořených mřížek v **hracím poli**. Při každé mřížce přidáme do seznamu **dvě** hodnoty. Hodnotu celého čísla zvětšenou o **jedničku** a znovu tuto hodnotu, ale **vynásobenou** číslem **100**.

Hodnoty v seznamu reprezentují hodnoty, které budou uložené v *Tagu* karet. Na celém **hracím** **poli** budou vždy **dvě** karty, které k sobě patří a **tvoří** **pár**. Pár se pozná podle toho, že právě jedna karta v *Tagu* bude mít **číslo** a druhá jeho **násobek** **číslem** **100**.

Čísla obou karet **nemohou** **být** **stejná**, protože by to **narušovalo** zbytek **logiky** **hry**. Zároveň číslo sto je **nejmenší** možné číslo k násobení, protože se počet karet může pohybovat v **desítkách**.

Pomocí třídy *Random* a cyklu *while*, se hodnoty v seznamu **přeházejí** a vytvoří se tak **efekt** **zamíchání**. Cyklus *for* vytvoří ze zamíchaného seznamu pomocí metody *CreateCardLabel* **nové** karty, které přidá do **hracího** **pole** a jejich hodnotu *Tag* do **seznamu** **neotočených** karet.

### InitializeBoard (Inicializovat herní pole)

Funkcí této metody je **zavolání** **ostatních** **metod** **inicializace** **herního** **pole**. Místo toho, aby ve hlavním formuláři hry byly volány všechny metody samostatně, tak se zavolá tato metoda, která vyvolá ostatní. Jsou to *LoadImages*, *SetupLayout* a *PlaceCards* s parametrem *boolean* *IsLoading*. *InitializeBoard* také nastavuje správné odsazení mřížky **pole**.

Obsah obrázku text, snímek obrazovky, Písmo

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.6 Ukázka kódu z metody InitializeBoard

### GetCardImage (Získat obrázek karty)

Vrácenou hodnotou této metody je **obrázek**. Volá se v případě, že je potřeba otočit kartu **obrázkem** **nahoru**. Pracuje s celočíselnou hodnotou, která reprezentuje **identifikační** **číslo** obrázku. Pokud je větší než 100, je číslem **100** **vydělena**. Metoda je ošetřena podmínkou, zdali je větší nebo rovna nule a zároveň menší než počet obrázků. Podmínka zajišťuje, že vstup je **validní**. Metoda následně vrátí **obrázek** **ze** **seznamu**, který odpovídá **číslu**.

### FlipCardFront (Otočit kartu vzhůru)

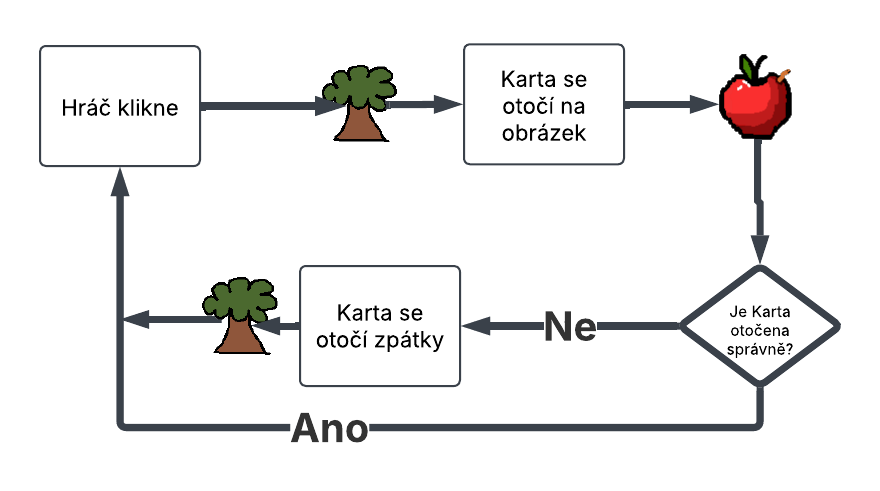
Při **otočení** **karty** je potřeba zobrazit její **obrázek**. To je funkcí této metody, která pracuje s **prvkem** *Label*. Metoda zkontroluje, zda uložená hodnota v *Tagu* je celé číslo, a poté je nastaven **obrázek** karty pomocí metody *GetCardImage*, které se hodnota **předá**.

### GetBackImage (Získat zadní obrázek karty)

Vrácenou hodnotou této metody je **obrázek** **zadní** **strany** *backImage*.

### FlipCardBack (Otočit kartu zpátky)

Metoda je volána při **špatném** **přiřazení** **dvou** **karet** a potřebě karty **otočit** zpět na **zadní** **stranu**. Pomocí *GetBackImage* se nastaví **obrázek** **prvku** *Label*.



Obr. 2.7 Diagram otočení karty

## Třída GameLogic (Herní logika)

Třída *GameLogic* spravuje veškerou **herní** **logiku** projektu. Zpracovává tah **hráče**, **počítače** a zároveň zjišťuje **vítěze**. V této třídě jsou velice důležité *Listy* *flippedLabels* a *hiddenLabels*, na kterých stojí veškerá **logika** **hry** **počítače**. V těchto seznamech jsou uložena celá **čísla**, která představují **identifikační** **čísla** **obrázků**.

### Konstruktor GameLogic

*Konstruktor* nastavuje dané proměnné, se kterými třída pracuje. Tyto proměnné spravují **herní** **prostředí** a **pravidla**. Jsou to **objekt** *gameBoard*, který odkazuje na **správu** **herní** **plochy**, a *instance správce skóre*, který **nastavuje** a **aktualizuje** **skóre** hráčů. *Konstruktor* také vyžaduje **počet** **hráčů** a **karet**, **zdali je zapnuta hra s počítačem, jeho případná obtížnost a nastavení zvuku**. Nastavení zvuku spravuje **objekt** *soundManager*. Inicializuje se také *slovník* *alreadyFlipped*, který zaznamenává pomocí **datového** **typu** *boolean* a jejich **id**, jaké karty byly otočeny. Při začátku nové hry jsou hodnoty ve *slovníku* nastaveny na *false*.

### Enum GameState (Výčtový typ – Stav hry)

*Enum* určuje různé **stavy** **hry**. Každý stav hry určuje **specifickou** **fázi** **tahu**, což zjednodušuje herní logiku.

Při čekání na tah, je *enum* nastaven na *Idle*. Otočení jedné z karet změní stav na *OneCardFlipped*. Správné kliknutí na druhou kartu hráčem nastaví *enum* na *Processing*. Při tomto stavu je **herní pole zablokováno**, aby se **nenarušila** herní logika. Stav *ProcessingForComputer* **zablokovává** **hrací** **pole** pro **hráče** při tahu počítače, aby hráč nemohl pokazit jeho tah.

Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.8 Změny stavů GameState

### OnCardClicked (Při kliknutí na kartu)

*OnCardClicked* je *asynchronní* *událost*, která zajišťuje správu **kliknutí** **hráče** na karty. Spravuje také logiku **otáčení** karet, **porovnávání** **párů**, **aktualizaci** **skóre** a stav *enumu* *gameState*. Událost pracuje se třemi *Labely*: *ClickedLabel*, což je karta, na kterou hráč **kliknul**, a s *Labely* first a second, kterým se při **správném** kliknutí přiřadí kliknutý *Label*.

Při kliknutí na **hrací** **pole** je nejdříve **ošetřen** **vstup**. Ten je ošetřen několika **podmínkami**. Vstup je zamítnut v několika situacích: pokud je *gameState* nastaven na *Processing* nebo *ProcessingForComputer*, neboli se **zpracovává jiný tah**; jestliže má kliknutá karta hodnotu *null*; nebo v případě, že hodnota *Tagu* kliknuté karty je nastavena na *backImageId*. Vstup bude také zamítnut, když je *gameState* nastaven na **stav** otočené **jedné** karty a kliknutá karta je **rovna** **otočené** kartě.

Hráč **kliknul** na kartu a vstup je **validní**. Jestliže je *gameState* nastaven na *Idle*, znamená to, že otočená karta je **první** **v** tomto **tahu**. *GameState* se nastaví na *oneCardFlipped* a začne **zpracovávání** logiky.

Do pomocné proměnné *first* se uloží *clickLabel*. *Objekt* *gameBoard* **otočí** metodou *FlipCardFront* kartu **obrázkem** **vzhůru**. V případě **zapnutého** zvuku *soundManager* zavolá metodu *PlayFlipCardSound*. Následně se pomocí **indexu** *Labelu* a **seznamu** *alreadyFlipped* zjistí, zda karta **byla** **už** **otočena**. Pokud **nebyla**, přidá se její hodnota *Tag* do seznamu již **otočených** karet *flippedLabels*. Hodnota *Tagu* se také **odebere** ze seznamu ještě **neotočených** karet *hiddenLabels*. Metoda se *asynchronně* na jednu vteřinu **pozastaví**, aby byl vytvořen dramatický **efekt** **otáčení** karty, a poté se vrátí.

V případě, že *gameState* je nastaven na *OneCardFlipped* a hráč **klikne** na **další** kartu, začne zpracovávání logiky **druhé** karty. *GameState* se nastaví na *Processing* a **do** pomocné **proměnné** se přiřadí **kliknutá** karta. Následuje stejná logika, jako při otočení **první** karty.

**Zjištění**, zda otočené karty **tvoří** **pár** stejných karet se provádí **porovnáváním** jejich **hodnot** v *Tagu* jednotlivých karet. Pokud **jeden** z *Tagů* **vynásobený** **100** je roven **druhému**, karty **tvoří** **pár**.

Jsou-li **zapnuté** zvuky, jejich **správce** *soundManager* spustí pomocí metody *PlayMatchedCorrect* zvuk **správně** **přiřazených** karet. Správce skóre zvedne hráči, který byl na řade **skóre** **o 1 bod** díky *AddScore* metodě a akci *ScoreUpdated*. Z *Listu* *flippedLabels* se **odeberou** *Tagy* **správně** **otočených** karet, aby nenarušovaly zbytek hry. Do obou *Tagů* *Labelů* se nastaví hodnota *backImageId*, tímto jsou karty mimo hru a už s nimi **nejde** hrát. Pomocné **proměnné** **se** **vynulují**, aby byly připravené na další tah. Zavoláním metody *WinnerCheck* se **zkontroluje**, zda hra **nemá** **vítěze**, a *gameState* se nastaví na stav *Idle*.

V případě, že karty **netvoří** **stejný** **pár** je na řadě jiná logika. Jsou-li zvuky **zapnuté**, spustí *soundManager* zvuk metodou *PlayMatchedWrong*. Metoda se asynchronně **pozastaví** na jednu vteřinu **pro** vizuální **efekt**. Proměnná *currentPlayer* udávající hráče, který je na řadě, přepne svoji hodnotu na **dalšího** hráče. Spustí se akce *ScoreUpdated* a metoda *OnTimerTick*.

Obsah obrázku text, diagram, snímek obrazovky, řada/pruh

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.9 Diagram tahu hráče

### OnTimerTick (Při tikání časovače)

Metoda *OnTimerTick* spravuje **otáčení**, **vynulování** **proměnných** a **přepínání** **hráčů** zároveň s vteřinovou odezvou. Po zkontrolování platnosti *Tagů* v podmínce se karty otočí **zadní** **stranou** **nahoru**. To provede objekt *gameBoard* za pomocí metody *FlipCardBack*. Následně se **vynulují** pomocné **proměnné** *first* a *second*, aby byly připravené pro **další** tah. *GameState* se ze stejného důvodu nastaví na *Idle*. Pokud je **zapnuta** hra **s** **počítačem** a v pořadí je **hráč** **druhý**, je na řade **počítač**. Tah počítače se zpracovává v metodě *ComputerTurn*.

### GetRight (Získat správně)

Tah počítače je závislý na **procentuální** **šanci**. Tato šance je předem určena **hráčem**, který si ji **v** **nastavení** hry **vybral**. Metoda obsahuje *switch*, který vrací tuto šanci v podobě **celého** **čísla**. Celé číslo představuje **procentuální** **šanci**, že počítač nebude otáčet karty **náhodně**, ale pokusí se najít **už** **otočený** **pár**.

Hráč si vybral mezi možnostmi **1, 2 a 3**, které představují dané **obtížnosti**. Pokud hráč zvolil možnost 1, vrátí mu switch celé číslo **30**. Možnost 2 představuje číslo **60** a možnost 3 představuje číslo **100**.

Obsah obrázku text, snímek obrazovky, Písmo

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.10 Ukázka kódu metody GetRight

### ComputerTurn (Tah počítače)

*Asynchronní* metoda *ComputerTurn* zajištuje logiku **tahu** **počítače**, pracuje s celými **čísly** uloženy v *Tagu* *Labelů*. **Nepracuje** přímo s **obrázky** v *Labelech*, protože řešení s čísly je daleko **rychlejší** a **optimalizovanější**.

Tah začíná nastavením *gameState* na *ProcessingForComputer*, a to kvůli **zablokování** interakce **hráče** s **hracím** **polem**. Poté se *asynchronně* **zastaví** na jednu vteřinu pro vytvoření **efektu**, že počítač „**přemýšlí**“.

Následně se inicializují dvě proměnné *indexFirstLabel* a *indexSecondLabel*. Do těchto proměnných se uloží hodnota *Tagu* **vybraných** karet. Také se určí, zda počítač **nevybere** karty **náhodně**, to díky proměnné *chance*, která **uchovává** **datový** **typ** *bool*. Do *chance* se uloží hodnota z podmínky, která zjišťuje, zda hodnota z metody *GetRight* je **větší** **nebo** **rovna** náhodně **vygenerovanému** **číslu** pomocí třídy *Random* z intervalu **0 až 100**.

Pokud *chance* je **pravda**, tak počítač **zkusí** **najít** pomocí dvou *for* cyklů dvě čísla, která tvoří **pár** karet. Počítač při srovnávání prochází pouze *List* už otočených karet *flippedLabels*. Jestliže najde pár, **uloží** si jejich hodnoty do **pomocných** **proměnných** a cyklus **předběžně** **ukončí** pomocnou proměnnou.

V případě, že **pár** stejných karet **nenašel**, nebo *chance* **nebyla pravda**, vybere z ještě **neotočených** karet *hiddenLabels* za pomocí třídy *Random* první **index**. Také ho **odebere** z tohoto *Listu* a přidá do již **otočených** karet.

V situaci, že *chance* je **pravda** a **první** **číslo** už bylo náhodně **vybráno**, projede počítač znovu cyklem *for* *List* už **otočených** karet. **Najde-li** kartu, která **tvoří** **pár**, zapamatuje si ji do **druhé** pomocné **proměnné** *indexSecondLabel*.

Nastane-li situace, že *chance* **není** **pravda** a bylo vybráno jenom **první** číslo, tak se za pomocí třídy *Random* **vybere** druhé. To se následně **odebere** z *Listu* **neotočených** karet a přidá se do *Listu* **otočených**. V případě, že v *Listu* **neotočených** karet není už žádná hodnota, **vybere** **se** **náhodná** hodnota z *Listu* už **otočených**.

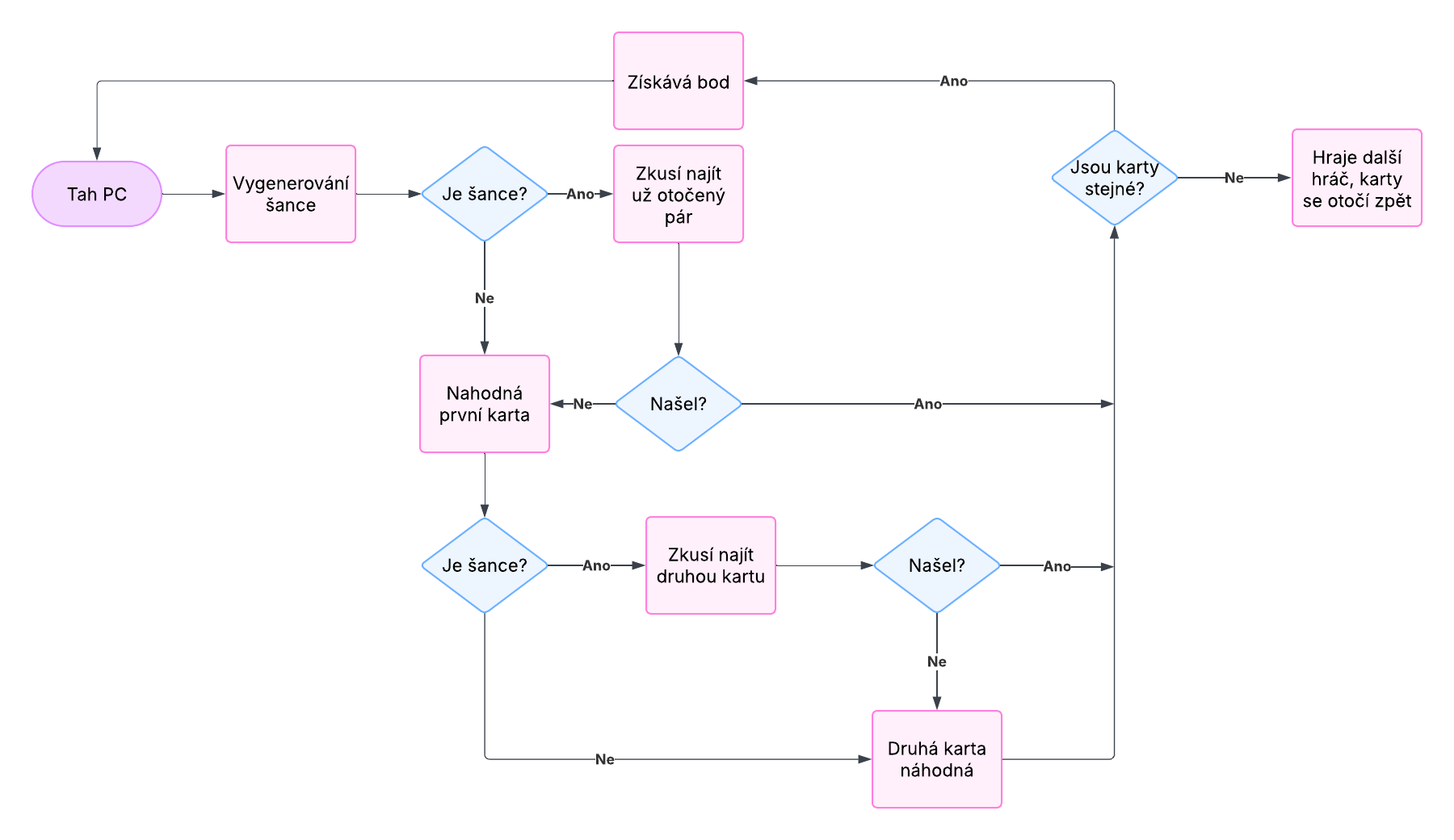
Následně se *asynchronně* metoda **pozastaví** pro dramatický **efekt** výběru karet. Inicializují se dvě nové pomocné proměnné *firstLabel* a *secondLabel*, obě **datového** **typu** *Label*. Cyklem *foreach* se zkontrolují všechny *Labely* uvnitř *TableLayoutPanelu*. Pokud hodnota *Tagu* uvnitř *Labelu* je rovna hodnotě v pomocné proměnné *indexFirstLabel* nebo *indexSecondLabel*, tak se *Label* **přiřadí** do dané proměnné přiřadí.

**Objekt** *gameBoard* otočí metodou *FlipCardFront* **obrázkem** **vzhůru** a pokud **je** hra se zvuky, tak jejich **správce** zavolá metodu *PlayFlipCardSound*. Mezi otočením obou karet je *asynchronní* pozastavení jedna vteřina pro dramatický **efekt** výběru.

Zjištění, zda karty tvoří **stejný** **pár**, probíhá **porovnáváním** pomocných proměnných celých **čísel**, ne samotných *Labelů*. Pokud jedno číslo je rovno druhému, násobenému 100, je to **pár** **stejných** karet.

V případě, že karty tvoří **stejný** **pár**, **odeberou** se jejich hodnoty *Listu* již **otočených** karet *flippedLabels*. Ve hře **se** **zvukem** se zavolá metoda *PlayMatchedCorrect*. **Správce** **skóre** přidá **počítači** jeden **bod** a zavolá **akci** *ScoreUpdated*. Vybraným *Labelům* se nastaví do hodnoty *Tag* *backImageId*, aby na ně hráč **nemohl** už kliknout a byly mimo hru. Následně se zavolá metoda *WinnerCheck*, aby zkontrolovala **vítěze**. Poté **hraje** počítač **znovu**.

Pokud karty **netvoří** **pár** **stejných** karet, tak se díky **objektu** *gameBoard* a metodě *FlipCardBack* **otočí** zpátky **zadní** **stranou** **nahoru**. Ve hře **se** **zvuky** se zavolá metoda *PlayMatchedWrong*. Proměnná *currentPlayer*, která udává, který **hráč** **je** **na** **řadě**, přepne svoji hodnotu na **dalšího** **hráče**. *GameState* se přepne na *Idle*, aby **mohl** hrát další hráč.



Obr. 2.11 Diagram tahu počítače

### WinnerCheck (Kontrola výherce)

Po správném **otočení** **dvou** karet se vždy zavolá tato metoda, která **zjistí**, zda není na **hracím** **poli** už **vše** **správně** **otočeno**. To zjistí za pomocí **součtu** **počtu** karet v *Listu* **neotočených** i **otočených**. Pokud je číslo **rovno** **nule**, hra skončí vyvoláním metody *EndScore* ze **správce** **skóre** a **akcí** *GameEnded*.

## Třída GameScoreManager (Správce herního skóre)

Třída *GameScoreManager* **spravuje** i **aktualizuje** **skóre** v průběhu hry a vypisuje **konečné** skóre. Zajištuje, aby správný hráč dostal **body**, které si zaslouží.

Třída pracuje se dvěma *poli*: *pole* **jmen** **hráčů** a jejich **skóre**. Na každém **indexu** v *poli* je jeden **hráč** a na **stejném** **indexu** v **druhém** *poli* je jeho **skóre**. Jelikož známe předem **počet** **hráčů**, je pro jejich uložení **rychlejší** využít *pole*, než *List* či jiné *kolekce*.

### Add, Get a Set (Přidat, získat a nastavit)

V třídě existuje několik metod, které **nastavují** nebo **vrací** **skóre** či **jméno**. Pracují s **indexem**, který **odkazuje** na hráče v *polích* **jmen** a **skóre**. Jsou ošetřeny podmínkou, zda je index **validní**. Například *AddScore* přidává hráči na určitém indexu v poli skóre 1 bod.

Obsah obrázku text, snímek obrazovky, software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.12 Ukázka metod ze třídy GameScoreManager

### GetSortedScores a EndScore (Získat seřazené skóre a Konečné skóre)

Tyto dvě metody zaručují **výstupní** **okno** **s** **finálním** **skóre**, a to v čase, kdy jsou na **herním** **poli** **správně** otočené **všechny** karty. *GetSortedScores* **přiřadí** a **seřadí** **skóre** a jména od **největšího** počtu bodů. *EndScore* seřazená jména **vypíše** pomocí *MessageBoxu*.

Výsledné okno je důležité proto, aby hráči znali svůj konečný počet bodů.

## Třída SoundManager (Správce zvuku)

Třída *SoundManager* slouží jako **správce** **zvukových** **efektů** ve hře. Načítá zvuky ze **souborů** hry a má na starosti jejich **přehrávání** a **uvolnění** po skončení hry.

### Konstruktor

*Konstruktor* pracuje s **parametrem** *bool* *isSound*, který určuje, zda si hráč přeje mít zvuky **povolené**. Pokud ano, *konstruktor* **inicializuje** přehrávač zvuku (*WaveOutEvent*) a *slovník* *audioStreams*, ve kterém jsou uložené zvuky. Následně se zavolá metoda *LoadAudio*, která **načte** zvuky **ze** **souborů** hry.

Obsah obrázku text, snímek obrazovky, Písmo, software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.13 Konstruktor třídy SoundManager

### LoadAudio (Načíst zvuky)

Metoda *LoadAudio* prochází **soubory** hry, konktrétně **složku** *Root*, která obsahuje **zvukové** **efekty** **s** **příponou** *mp3*. Správně nalezené soubory uloží do *slovníku* jako *paměťový* *proud* (*MemoryStream*) *s* *klíčem*, jenž je **názvem** souboru. Procházení souboru je ošetřeno výjimkou.

### PlayAudio (Spustit zvuk)

Metoda pracuje s **názvem** zvuku, který následně **přehraje**. Nejdříve pomocí metody *StopAudio* **zastaví** případně už přehrávající se zvuk a poté *Mp3FileReaderem* načte *paměťový* *proud* ze *slovníku*. Načtený zvuk **přehraje** pomocí **přehrávače** *WaveOutEvent*.

### StopAudio a Dispose (Zastavit zvuk a Uvolnit)

*StopAudio* **zastaví** přehrávání a **uvolní** data spojená se **čtením** **zvuků**. To provede metodou *Dispose*, která **zastaví** zvuky a **vyčistí** **kompletně** všechny *proudy*.

### Play Metody (Spuštění samostatných zvuků)

Pro usnadnění **volání** **zvuků** v logice hry jsou vytvořeny tři metody: *PlayFlipCardSound*, *PlayMatchedCorrect*, *PlayMatchedWrong*. Uvnitř nich je volána metoda *PlaySound*, která přehraje **statický** **název** **zvuku**, který **ladí** s **názvem** **metody**.

## Třída GameSave (Uložená hra)

Třída *GameSave* obsahuje všechny **důležité** **proměnné** **a** **informace** o stavu hry, kterou chce hráč **uložit**. **Atribut** *Serializable* usnadňuje jednoduché **uložení** do **libovolného** **souboru** a jeho následné **otevření** (*Deserializace*).

Obsah obrázku text, snímek obrazovky, software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.14 Proměnné ve třídě GameSave

## Třída GameSaveManager (Správce uložení hry)

**Statická** třída *GameSaveManager* má dva hlavní účely: **uložení** instance třídy *GameSave* do **binárního** souboru a jeho následné **načtení** do hry. Je zvoleno uložení do **binárního** souboru, protože běžný uživatel bude mít problém data **upravit**, jelikož jsou pro něj **nečitelná**. **Všechny** **uložené** soubory v tomto projektu používají **binární** **soubory** právě z tohoto důvodu.

### SaveGame a LoadGame (Uložení hry a Načtení hry)

Metoda *SaveGame* pracuje s herními daty z **rozehrané** hry a s **cestou** **k** **souboru**, do kterého chce hráč hru **uložit**. Třída *FileStream* s módem *Create* využije *BinaryFormatter* a do zvolené lokace převede do **binárních** **dat** ta **herní**.

Metoda *LoadGame* dělá přesný opak. Na zvolené lokaci pomocí třídy *FileStream* s módem *Open* a využitím *BinaryFormatter* **deserializuje** **binární** **data** a převede je na použitelná **herní** **data**. Obě metody jsou ošetřeny **výjimkou** v případě, že uživatel soubory **poškodil**. Všechna logika s **binárními** **soubory** je v tomto projektu ošetřena přesně z toho důvodu.

## Třída ScoreData (Informace o Skóre)

Třída *ScoreData* obsahuje **všechna** důležitá **data** **o** **hráči**, který **dohrál** hru. Mezi data patří jeho **jméno**, **počet** **výher**, **počet** **proher**, **počet** **nalezených** **karet** a **celkový** **počet** **karet** v jeho hrách. Tyto informace budou uloženy a použity do **tabulky** **skóre**. Třída obsahuje **atribut** *Serializable*.

Obsah obrázku text, snímek obrazovky, software

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.15 Proměnné ve třídě ScoreData

## Třída GameScoreSaveManager (Správce uložení skóre hry)

**Statická** třída *GameScoreSaveManager* spravuje **data** **instance** třídy *ScoreData*, a to díky **binární** serializaci. Může je **uložit**, **načíst** a **mazat**.

Metoda *SaveScoreData* pracuje s *Listem* **datového** **typu** *ScoreData*. **Uloží** **data** z *Listu* do **souboru** pomocí třídy *FileStream* a *BinaryFormatteru*. **Data** jsou **uložena** v **binární** podobě.

Metoda *LoadScoreData* je přesným opakem a převede **data** z **binární** **podoby** do *Listu* tak, aby byla **použitelná** v další implementaci.

Metoda *ClearScoreData* zkontroluje, zda **binární** **soubor** s **daty** existuje a případně ho **vymaže** za pomocí třídy *File* a její metody *Delete*.

## Formuláře

*Formuláře* představují **uživatelské** **rozhraní**, jehož cílem je dosáhnout jednoduché orientace ve hře, ale s přívětivým zážitkem. Vhodné části hry jsou proto rozděleny do více *nemodálních* *formulářů*. Takové části jsou **čtyři**. Každá část má svůj *designer* **soubor** a třídu formuláře. *Designer* soubor, vytvořen za pomocí *Drag* *and* *Drop* způsobu (Grafické rozhraní návrháře), nikoli za pomocí čistého kódování, tvoří **vzhled** **formuláře**. Z třídy *Form* dědí třída formuláře, v níž se určuje logika, která řídí, co formulář dělá a jak reaguje.

## StartingMenu (Hlavní menu)

Formulář *StartingMenu* je **první** okno, které hráč při spuštění uvidí. Slouží jako **křižovatka** mezi ostatními částmi hry. Mezi nimi se přechází za pomocí tlačítek.

Obsah obrázku text, ovoce, jahoda, snímek obrazovky

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.16 Vzhled formuláře StartingMenu

Při spuštění **nové** **hry** je nastavení měněno **proměnnými**, které formulář obsahuje. Tyto proměnné se dají změnit v **nastavení**. V tomto formuláři jsou nastaveny na **počáteční** **hodnotu**, aby nemusely být hráčem při každém spuštění měněny. Obsah obrázku text, snímek obrazovky, Písmo

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.17 Ukázka důležitých proměnných ze formuláře StartingMenu

Z formuláře lze také otevřít **uloženou** **hru** z **datového** **souboru**. Hráč soubor vybere pomocí *OpenFileDialogu* a hru načte do **nového** formuláře *NewGame* pomocí třídy *GameSaveManager* a její metody *LoadGame*. Tato metoda soubor otevře a převede z **datového** **souboru** do **použitelných** **dat**. Zbylou logiku načtení hry obsahuje formulář *NewGame*.

## GameSettings (Nastavení hry)

Formulář *GameSettings* pracuje s **proměnnými**, které předává **hlavnímu** **menu**. Hráč zde může měnit nastavení samotné hry a následně se pomocí tlačítka vrátit zpět.

Obsah obrázku text, ovoce, snímek obrazovky, jahoda

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.18 Vzhled formuláře GameSettings

Platný **počet** **hráčů** je ošetřen **podmínkou**. Pokud počet hráčů **není** **platný**, tlačítko **nepustí** hráče **do** **menu** a vyhodí **upozornění** v podobě *MessageBoxu*.

Zvolení velikosti **hracího** **pole** a **obtížnosti** **počítače** je hráčem provedeno pomocí *RadioButtonů*. Ty jsou rozděleny do dvou skupin *GroupBoxem*, proto lze vybrat vždy pouze **jednu** **možnost**, což je účelem, protože nelze mít více možností velikosti či obtížnosti naráz.

## NewGame (Nová hra)

Ve formuláři *NewGame* se nachází **hrací** **pole**. Zde hráč **hraje** samotnou **hru**. Obsah obrázku text, snímek obrazovky, kreslené, diagram

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.19 Vzhled formuláře NewGame

### Konstruktor

*Konstruktor* formuláře nastavuje **proměnné**, které získává z **hlavního** **menu**. Vytvoří se nová instance třídy *GameBoard*, která metodou *InitializeBoard* nastaví **herní** **pole**. Pokud je hra **nová** a nenačítá se z **uloženého** **souboru**, tak se zavolá metoda *GetNames*, a to proto, že v případě načítaní již **uložené** **hry** jsou **jména** už zadaná.

Vytvoří se nové instance tříd *ScoreManager* a *GameLogic*, které budou ve formuláři spravovat logiku hry a skóre.

*Konstruktor* také zajištuje **kliknutí** na karty. Když hráč **klikne** na kartu, spustí se *Event* *Handler*. Pokud je prvek v *Event* *Handleru* kartou, zavolá se *asynchronní* metoda *OnCardClicked* ze třídy *GameLogic*. Metoda zpracuje logiku tahu hráče. Po zpracování metody se zavolá metoda *ShowScore*, která případně **aktualizuje** **skóre**.

*Konstruktor* také připojuje k událostem ze třídy *GameLogic*, *ScoreUpdated* a *GameEnded* metody z formuláře *NewGame*. To zajistí správné **oddělení** a **volání** **kódu**, který pracuje s **prvky** *Windows* *Formulářů*.

### GetNames (Získat jména)

Metoda *GetNames* získává **jména** **hráčů**, se kterými pracují ostatní třídy. Při zapnutí nové hry vyskočí na hráče formulář, do kterého **může** jména zadat. Pokud žádné jméno nezadá, nastaví se jako jméno jeho **pořadí**. Například hráč, který je v pořadí třetí, bude mít jméno 3.

Ve hře, kde je jeden z hráčů počítač, je vždy jméno **druhého** **hráče** **PC**, protože na tomto místě v pořadí bude hrát.

Obsah obrázku text, snímek obrazovky, diagram, řada/pruh

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.20 Vzhled formuláře s jmény

### ShowScore (Ukázat skóre)

Tato metoda spravuje zobrazování **jména** **hráče**, jeho **pořadí** **a** **skóre** v *StatusStripu*, který se nachází pod **hracím** **polem**. Informace o hráči získává za pomocí třídy *GameLogic* a *ScoreManager*.

### SaveGame (Uložit hru)

Metoda *SaveGame* se spustí po **kliknutí** na tlačítko, které představuje **zelenou** **šipku** ve *ToolStripu*. Pomocí *SaveFileDialogu* hráč vybere, kam chce rozehranou hru **uložit**. Pokud hru uloží správně, začne zpracovávání uložení hry.

Do *Listu* celých čísel *cardPositions* se uloží všechny *Tagy* (identifikační čísla obrázků) *Labelů* (hracích karet) v *TableLayoutPanelu* (hracím poli). Do proměnné *indexFirstFlipped* a *indexSecondFlipped* se uloží **indexy** **otočených** karet, které se ještě **nestihly** zpracovat. Hra totiž může být hráčem **uložena** i **v** **půlce** **tahu**. Následně se vytvoří **nová** **instance** **proměnné** třídy *GameSave*, která obsahuje všechna **důležitá** **data** o hře.

Obsah obrázku text, snímek obrazovky

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.21 Ukázka kódu uložení hry

Zavoláním metody *SaveGame* v třídě *GameSaveManageru* se **data** na vybrané místo **uloží**.

### LoadGame (Načíst hru)

*LoadGame* dělá přesný opak než předchozí metoda. Při správném vybrání **souboru** hry *OpenFileDialogem*, začne logiku načítání uložené hry.

Vytvoří se **nová** **instance** třídy *GameSave*, do které se nahrají **data** pomocí třídy *GameSaveManager* a její metody *LoadGame*. Následuje předání **dat** z **proměnné** do aktuálního okna *NewGame*.

**Důležité** **proměnné**, které hráč může vybírat **v** **nastavení**, se předají jako **první**. **Instance** *ScoreManager* nastaví správně **jména** a **skóre** hráčů. Správce **herního** **pole** třídy *GameBoard* zavolá **inicializaci** **pole**. Po správné určení velikosti *TableLayoutPanelu* (herního pole) se cyklem *for* nahrají z *cardPositions* správné *Tagy* (id obrázků) *Labelů* (karet).

Třída *GameBoard* obsahuje také **proměnnou** *MatchedPairs*, což je *slovník*, který pracuje s **klíčem** **datového** **typu** *int* a hodnotou stejného typu. Slouží k zapamatování **indexu** v **hracím** **poli** a *Tagu* (id obrázku) **správně** **otočených** karet. Při každém správném otočení se informace o kartách **přiřadí** do *slovníku*.

Cyklem *foreach* se projede každá informace o kartě **ve** **slovníku**. Pokud jsou informace platné, do **indexu** v **hracím** **poli** se přiřadí správný *Tag*. **Správce** **herního** **pole** kartu otočí a *Tag* následně změní na *backImageId* (id zadní karty obrázku) proto, aby **nebylo** **možné** na kartu **kliknout**.

V případě, že **uložená** **proměnná** *IndexFirstFlipped* a *IndexSecondFlipped* v sobě mají platnou hodnotu, tak se v **hracím** **poli** otočí karta na jejich **indexu**. To zajistí, že karty otočené v průběhu **uložení** zůstanou v **tahu**.

Nakonec se nastaví proměnná *currentPlayer*, *flippedLabels*, *hiddenLabels* a *gameState*. Zavolá se i metoda *ShowScore*, která aktualizuje *StatusStrip* pod hracím polem.

Celá metoda je ošetřena **podmínkou**, protože hráč mohl poměnit **datový** **soubor**.

### RestoreFromGameSave (Obnovit ze zálohy hry)

Metoda obsahuje stejnou logiku jako *LoadGame*, akorát pracuje se **souborem**, který se vybere a předává z **hlavního** **menu**. Metody jsou oddělené právě proto, aby hráč **nemusel** začít hrát novou hru vždy, kdy chce otevřít hru již **uloženou**.

### EndGame (Konec hry)

*EndGame* má na starost zpracování **dat** o hráčích a jejich následné **předání** do tabulky **skóre**. Metoda se volá na **konci** hry.

Cyklem *for* projde **všechny** hráče a najde toho, kdo se stane **vítězem** (má nejvíce bodů) a **poraženým** (nejméně bodů).

Do *Listu* **datového** **typu** třídy *ScoreData* nahraje všechny uložené informace o hráčích v **datovém** **souboru**. To provede třídou *GameScoreSaveManager* a metodou *LoadScoreData*. Následně projede opět všechny hráče z **odehrané** hry. Pokud v *Listu* **neexistuje** **záznam** se **stejným** **jménem**, tak se vytvoří **nová** **instance** *ScoreData* a ta se přidá do *Listu*. V opačném případě se do *Listu* jenom přičtou hodnoty.

*List* se uloží do **datového** **souboru** třídou *GameScoreSaveManager* a její metodou *SaveScoreData*. Při **správném** **uložení** vyskočí **potvrzení** ve formě *MessageBoxu*. Na úplný konec této metody se otevře formulář *Score*, kde je **tabulka** se všemi záznamy hráčů. Formulář *NewGame* se zavře.

## Score (Skóre)

Formulář *Score* zobrazuje **tabulku** **skóre**, která obsahuje **všechny** **informace** o hráčích z **datového** **souboru**. Okno se zobrazí po každé **dohrané** **hře**, ale lze na něj přistoupit i z **hlavního** **menu**. Hlavní **proměnnou**, se kterou se pracuje, je *List* s **datovým** **typem** *ScoreData*. Tento *List* uchovává veškeré informace, které se následně zobrazí hráči v *DataGridView*.

Obsah obrázku text, snímek obrazovky, diagram, design

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obr. 2.22 Vzhled formuláře Score

### Konstruktor

*Konstruktor* tohoto formuláře volá metody, které **inicializují** *DataGridView* (tabulku) a *ComboBox* (výběr sloupců), načítají **skóre** do *Listu* a zobrazují ho do **tabulky**. *Konstruktory* existují dva, jeden uchovává pro **novou** **hru** proměnné, které si hráč vybral v **nastavení**. Uchovává je proto, aby při přechodu hráče z **menu** do **tabulky** a zpět, nebyly proměnné **ztraceny**. Druhý *konstruktor* slouží pro *Score* zobrazené z **dokončené** hry, ten proměnné uchovávat nemusí.

### InitializeDataGridView a InitializeComboBox (Inicializace DGV a ComboBoxu)

Metoda *InitializeDataGridView* **inicializuje** a **nastavuje** *DataGridView*. Nastavuje styly, sloupce a další vlastnosti.

*InitializeComboBox* nastavuje hodnoty položek na **jména** sloupců a počáteční vybraný **index**.

### LoadScoreData a DisplayData (Načíst informace o skóre a Zobrazit informace)

*LoadScoreData* načítá **data** za pomocí třídy *GameScoreSaveManager* a metody, která do *Listu* předá **data** z **datového** **souboru**. Metoda je ošetřena **výjimkou**, v případě, kdy soubor je **nepoužitelný**.

*DisplayData* **vymaže** veškeré řádky v tabulce a nahraje všechny **informace** z *Listu*.

### buttonFilter\_Click a buttonClear\_Click (Tlačítko filtrování a vymazání)

Metoda *buttonFilter*\_Click se spustí po **kliknutí** na tlačítko **Filtrovat**. Pracuje s textem z *TextBoxu*. Pokud je text prázdný, zobrazí se všechna data. V opačném případě se do pomocné proměnné přiřadí vybrána **položka** v *ComboBoxu*.

Vytvoří se také pomocný *List* **datového** **typu** *ScoreData*, do kterého se uloží všechny **data** z tabulky, která odpovídají **filtrování**. To se provede cyklem *foreach*, kterým se projdou všechny záznamy v původním *Listu*. V případě, že byla nalezena nějaká **data**, tak se zobrazí v **tabulce** místo původních, a to metodou *DisplayData*.

Při kliknutí na tlačítko **Vymazat** **Tabulku** se zobrazí *MessageBox*, který se ptá na **potvrzení**, zda chce hráč tabulku opravdu vymazat. Pokud je jeho odpověď **ano**, tak se zavolá metoda *ClearScoreData* ze třídy *GameScoreSaveManager*, která **vymaže** **datový** **soubor**. Následně se **vymažou** **data** v *Listu*, a ten se zobrazí metodou *DisplayData*. **Mazání** je ošetřeno **výjimkou**.



Obr. 2.23 Obsah

Text obsahu se píše od jednotné svislice (se zřetelem k nejdelšímu číselnému označení).

## Psaní úvodu

Úvod může být osobitějšího rázu. První odstavec by měl obsahovat motivaci či důvod, který autora přiměl k volbě daného tématu s ohledem na přínos práce. V dalších odstavcích může být popsán obsah jednotlivých kapitol.

## Struktura odstavců

V případě, že věta vychází se spojkou na konci řádku, je spojka vždy přesunuta pomocí Shift + Enter na následující řádek. Hodnoty s jednotkami musí mít mezi číslem a označením jednotky jednu mezeru. To lze uskutečnit pomocí Shift + Ctrl + Mezerník (tzv. pevná mezera).

Text práce by měl být napsán v neutrální formě, tj. ve 3. osobě v trpném rodě. V textu by se neměli objevovat slangové výrazy, citově zabarvená slova ani podmiňovací způsob (s výjimkou je-li, uvažujeme-li apod.). Autor by se měl vyhnout přílišnému opakování slov či užívání nadbytečných výrazů.

Forma textu:

* v neurčité formě (bylo zjištěno, navrhuje se…);
* v 1.  osobě jednotného čísla (zjistil jsem, navrhuji …) – zvláště vhodné v kapitolách, které jsou vlastní prací autora.

## Obrázky, tabulky a rovnice

Popisek obrázku a tabulky se vkládá kliknutím pravého tlačítka myši na objekt a výběrem možnosti *Vložit titulek*. Následně je vybrán typ objektu (*Tab.* nebo *Obr.*) a jeho poloha (obrázky pod objekt, tabulky nad objekt). Styl popisku je Times New Roman 11 kurzíva uprostřed. Seznamy obrázků a tabulek na konci dokumentu jsou automaticky vygenerovány. Obrázky a tabulky mají vždy i slovní popis a rovnice jsou bez slovního popisu. Za titulek obrázku nepatří tečka. [3] Číslování obrázků, tabulek a rovnic je provedeno dle **hlavní kapitoly**, v níž se vyskytují a jejich pořadí v této kapitole. Nástroj *Vložit Titulek* čísluje popisky obrázků a tabulek automaticky. V případě že dojde k editaci, odstranění nebo přesunu již existujícího popisku je nutné dokument aktualizovat. Pro samotné vkládání rovnic je užit nástroj *Rovnice* v záložce *Vložení* (rovnice jsou vždy psány kurzívou).

|  |  |
| --- | --- |
|  |  |

Vzorec pro měření entropie. Rovnice jsou bez slovního popisu.



Obr. 2.24 Příklad umístění legendy obrázku

Titulek ke grafu a obrázku se píše pod objekt.

Tab. 2.1 Legenda k tabulce

|  |  |  |  |
| --- | --- | --- | --- |
| Jméno | Příjmení | Body | Známka |
| Petr | Novák | 25 | 4 |
| Karel | Kolář | 36 | 3 |
| Martin | Pokorný | 89 | 1 |

Titulek tabulky se píše nad tabulku.

Vložený objekt musí být vždy okomentován. Je nutné napsat před a za objekt alespoň jeden odstavec textu.

## Řazení a struktura kapitol

Z hlediska přehlednosti každá nová kapitola (Nadpis 1) začíná na novém listu. První kapitola bývá zaměřena na rešeršní část, tedy definice pojmů, vymezení studované oblasti apod. Druhá a následující kapitoly jsou zaměřeny na řešení samotného problému.

## Závěr

Závěr obsahuje stručné shrnutí získaných poznatků, uvedení dalších možných postupů či řešení, hodnocení dostupné odborné literatury, ze které bylo čerpáno. Závěr by měl obsahovat kritické porovnání záměru práce a dosažených výsledků, srovnání dosažených výsledků s dosud známými poznatky a popis odlišností od doposud známých skutečností. Závěr může naznačit praktické uplatnění výsledků práce.

## Seznam použitých zdrojů

Účelem této části je zaznamenání publikací, jež byly využity v maturitní práci.

Citace se řídí normou ČSN ISO 690 Bibliografické citace. [1] V případě doslovné citace je potřebné citovaný text graficky odlišit od ostatního textu, ohraničit apostrofem, případně ještě kurzívou. Odkaz na citovaný zdroj, jehož plná citace bude uvedena v soupisu literatury na konci práce, se uvede přímo za doslovně citovaný text.

Citování zdrojů se provádí v hranatých závorkách, používá se metoda číselných odkazů. Tento seznam je vždy seřazen dle výskytu citací v textu práce. Necitování použité literatury je považováno za plagiátorství.

## Seznam použitých symbolů a zkratek

Seznam symbolů a zkratech je vytvořen pomocí tabulky. Zkratky a názvy veličin jsou psány stylem *Normální* a jednotky jsou vytvořeny pomocí nástroje *rovnice*. Pro přidání řádku je potřeba kliknout pravým tlačítkem myši do posledního řádku tabulky a z nabídky *vložit* vybrat možnost *vložit pod*. Pro odstranění pak obdobně vybrat možnost *odstranit celý řádek*.

## Seznamy použitých obrázků a tabulek

Tyto seznamy se vytvářejí automaticky dle použitých stylů. Seznamy je potřeba aktualizovat kliknutím pravého tlačítka myši na první položku seznamu a následným výběrem možnosti aktualizovat pole (nebo kliknutím na první položku seznamu a stiskem F9).

## Seznam příloh

Seznam příloh je nutné vyplnit ručně.

Závěr

Desková hra Pexeso zajištuje zábavný požitek z jednoduché paměťové hry. Uživatelské rozhraní tvořené pomocí Windows Formulářů obstarává přehlednost průchodu aplikací. Plynulý běh hry zaručuje optimalizovaný kód v programovacím jazyce C#. V průběhu vývoje bylo velice užitečné vývojové prostředí Visual Studio, především jeho grafický návrhář. Verzovací systém GitHubu usnadnil vrácení chyb v kódu.

Průběh vývoje probíhal bez problému a včas. Výsledná aplikace odpovídá požadovanému zadání. Hráč si může vybrat mezi hrou až se 6 hráči či hru s počítačem. Herní pole září obrázky ovoce, které hře dodávají barevnost a zábavu.

Rozdělení kódu do samostatných částí zajišťuje snadné rozšíření projektu o další funkce v budoucnu. V plánu je předělání desktopové aplikace do webové, což by umožnilo hru s více hráči na více zařízeních.

Realizace projektu mi prohloubila znalosti vývoje v C# a také mi pomohla uvědomit si, že se programování chci věnovat také po dokončení studia.

Seznam použitých zdrojů

1. RISTOV, Ivan. *The Complete History of Board Games*. Online. Dostupné z: <https://boardgamesland.com/the-complete-history-of-board-games/>. [cit. 2024-12-29].
2. BERGEROVÁ, Alžběta a VAŇKOVÁ, Lucie. *Hry v průběhu věků (4) Senet*. Online. CESTY ARCHEOLOGIE. 2021. Dostupné z: <https://www.cestyarcheologie.cz/single-post/hry-v-prubehu-veku-4-senet>/. [cit. 2025-03-17].
3. AUGUSTYN, Adam (ed.). *Go*. Online. Britannica. 1998, 7. února 2025. Dostupné z: <https://www.britannica.com/topic/go-game>. [cit. 2025-03-18].
4. HOUSE OF STAUTON. *History of Chess*. Online. Dostupné z: <https://www.houseofstaunton.com/history-of-chess?srsltid=AfmBOora16oAc_2gexbFihQdeaTroePGaTp_HuCxLu_sT3aTQ8rKefqN>. [cit. 2025-03-18].
5. BANDPASS DESIGN. What Are The Psychological Benefits Of Board Games for Adults & Children? Online. 2024. Dostupné z: [https://bandpassdesign.com/blogs/news/benefits-of-board-games?srsltid=AfmBOoob1w8gXQnjEmzTnUycI-i5v-mG7k9KFTV2P1m61fPBpzRpwjYh](https://bandpassdesign.com/blogs/news/benefits-of-board-games?srsltid=AfmBOoob1w8gXQnjEmzTnUycI-i5v-mG7k9KFTV2P1m61fPBpzRpwjYh%20) [cit. 2025-03-18].
6. MF DNES a HLAVÁČ, Jakub. *Pekelně se soustřeď. Autor pexesa přišel kvůli komunistům o slávu i peníze*. Online. IDNES. 2019. Dostupné z: <https://www.idnes.cz/hobby/domov/pexeso-hra-vznik-autor-zdenek-princ.A191015_151742_hobby-domov_mce>. [cit. 2025-03-18].
7. HARVARD T.H CHAN SCHOOL OF PUBLIC HEALTH. *Vegetables and Fruits*. Online. Dostupné z: <https://nutritionsource.hsph.harvard.edu/what-should-you-eat/vegetables-and-fruits/>. [cit. 2025-03-18].
8. KALMUS. *Doporučená denní dávka ovoce a zeleniny*. Online. FÉR Potravina. 2021. Dostupné z: <https://www.ferpotravina.cz/clanky/doporucena-denni-davka-ovoce-a-zeleniny>. [cit. 2025-03-18].
9. FOLKVORD, Frans; ANASTASIADOU, Dimitra Tatiana a ANSCHÜTZ, Doeschka. Memorizing fruit: The effect of a fruit memory-game on children's fruit intake. Online. *Preventive Medicine Reports*. 2017, vol. 5, s. 106-111. ISSN 2211-3355. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S2211335516301565>. [cit. 2025-03-18].
10. GEEKSFORGEEKS. *Introduction to Visual Studio*. Online. 22. září 2023. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-visual-studio/>. [cit. 2025-03-18].
11. MICROSOFT. *Webové jazyky*. Online. Dostupné z: <https://visualstudio.microsoft.com/cs/vs/features/web/languages/>. [cit. 2025-03-18].
12. KUMAR, Sunny. *Top Programming Languages Used in Visual Studio*. Online. Oxtrys. 2024. Dostupné z: <https://www.oxtrys.com/top-programming-languages-used-in-visual-studio>. [cit. 2025-03-18].
13. WADHWA, Mayank. *Visual Studio: Community, Professional, or Enterprise?* Online. Pangea. 2022, 8. srpna 2024. Dostupné z: <https://pangea.ai/resources/visual-studio-community-professional-or-enterprise>. [cit. 2025-03-18].
14. MICROSOFT. *Začínáme s rozhraním .NET Framework*. Online. 07. 03. 2025. Dostupné z: <https://learn.microsoft.com/cs-cz/dotnet/framework/get-started/#net-framework-for-developers>. [cit. 2025-03-18].
15. GEEKSFORGEEKS. *Introduction to .NET Framework*. Online. 31. leden 2025. Dostupné z: <https://learn.microsoft.com/cs-cz/dotnet/framework/get-started/#net-framework-for-developers>. [cit. 2025-03-18].
16. GEEKSFORGEEKS. *C# Tutorial*. Online. 16. prosince 2024. Dostupné z: <https://www.geeksforgeeks.org/csharp-programming-language/>. [cit. 2025-03-18].
17. GEEKSFORGEEKS. *Introduction to C# Windows Forms Applications*. Online. 4. května 2023. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-c-sharp-windows-forms-applications/>. [cit. 2025-03-18].
18. GITLAB. *What is Git version control?* Online. Dostupné z: <https://about.gitlab.com/topics/version-control/what-is-git-version-control/>. [cit. 2025-03-18].
19. GITHUB. *About GitHub and Git*. Online. Dostupné z: <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>. [cit. 2025-03-18].
20. MICROSOFT. *TableLayoutPanel – ovládací prvek (Windows Forms)*. Online. 2024. Dostupné z: <https://learn.microsoft.com/cs-cz/dotnet/desktop/winforms/controls/tablelayoutpanel-control-windows-forms?view=netframeworkdesktop-4.8>. [cit. 2025-03-18].

Seznam použitých symbolů a zkratek

|  |  |  |
| --- | --- | --- |
| Symbol | Veličina | Jednotka |
| S | Entropie |  |
| Q | Teplo |  |
| T | Termodynamická teplota |  |
| t | Čas |  |
|  |  |  |

Seznam obrázků

[Obr. 1.1 Hrací plocha hry Senet 10](#_Toc193047857)

[Obr. 1.2 Šachovnice 10](#_Toc193047858)

[Obr. 1.3 Fotka hrdiny Vinettou 13](#_Toc193047859)

[Obr. 1.4 Screenshot z Visual Studia 15](#_Toc193047860)

[Obr. 1.5 Ukázka kódu v jazyce C# 16](#_Toc193047861)

[Obr. 1.6 Ukázka grafického vývojáře ve Visual Studiu 17](#_Toc193047862)

[Obr. 2.1 Rozdělení souborů hry 19](#_Toc193047863)

[Obr. 2.2 Příkladná ukázka kódu Třídy 20](#_Toc193047864)

[Obr. 2.3 Ukázka TableLayoutPanelu 21](#_Toc193047865)

[Obr. 2.4 Jeden z obrázků hry 22](#_Toc193047866)

[Obr. 2.5 Ukázka Labelu ve Formuláři 23](#_Toc193047867)

[Obr. 2.6 Ukázka kódu z metody InitializeBoard 24](#_Toc193047868)

[Obr. 2.7 Diagram otočení karty 25](#_Toc193047869)

[Obr. 2.8 Změny stavů GameState 26](#_Toc193047870)

[Obr. 2.9 Diagram tahu hráče 28](#_Toc193047871)

[Obr. 2.10 Ukázka kódu metody GetRight 29](#_Toc193047872)

[Obr. 2.11 Diagram tahu počítače 31](#_Toc193047873)

[Obr. 2.12 Ukázka metod ze třídy GameScoreManager 32](#_Toc193047874)

[Obr. 2.13 Konstruktor třídy SoundManager 33](#_Toc193047875)

[Obr. 2.14 Proměnné ve třídě GameSave 34](#_Toc193047876)

[Obr. 2.15 Proměnné ve třídě ScoreData 35](#_Toc193047877)

[Obr. 2.16 Vzhled formuláře StartingMenu 37](#_Toc193047878)

[Obr. 2.17 Ukázka důležitých proměnných ze formuláře StartingMenu 37](#_Toc193047879)

[Obr. 2.18 Vzhled formuláře GameSettings 38](#_Toc193047880)

[Obr. 2.19 Vzhled formuláře NewGame 38](#_Toc193047881)

[Obr. 2.20 Vzhled formuláře s jmény 40](#_Toc193047882)

[Obr. 2.21 Ukázka kódu uložení hry 41](#_Toc193047883)

[Obr. 2.22 Vzhled formuláře Score 43](#_Toc193047884)

[Obr. 2.23 Obsah 47](#_Toc193047885)

[Obr. 2.24 Příklad umístění legendy obrázku 49](#_Toc193047886)

Seznam tabulek

[Tab. 2.1 Legenda k tabulce 12](#_Toc147493615)

Seznam příloh

Prázdná šablona maturitní práce