Česká zemědělská univerzita v Praze

Technická fakulta

Katedra technologických zařízení staveb



Bakalářská práce

Návrh univerzální programové logiky pro vývoj her

Martin Novák

2022 ČZU v Praze





**Čestné prohlášení**

Prohlašuji, že svou bakalářskou práci "Návrh univerzální programové logiky pro vývoj her" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Poděkování**

Rád(a) bych touto cestou poděkoval(a) jméno vedoucího, případně dalších osob, a informace, za co děkujete.

Návrh univerzální programové logiky pro vývoj her

**Abstrakt**

Souhrn práce (cca 15 řádek textu).

**Klíčová slova:** herní enginy, uživatelská rozhraní, programovací jazyky, objektový model, vývoj herDesign of universal program logic for game development

**Abstract**

Anglický překlad českého souhrnu

**Keywords**: game engines, user interface, programing languages, object model, game development

**Obsah**

[1. Úvod 1](#_Toc101449580)

[2. Cíl práce a metodika 1](#_Toc101449581)

[2.1 Cíl práce 1](#_Toc101449582)

[2.2 Metodika 1](#_Toc101449583)

[3. Výběr vhodných programovacích jazyků pro vývoj her 1](#_Toc101449584)

[3.1 C++ 2](#_Toc101449585)

[3.2 Java 2](#_Toc101449586)

[3.3 C# 2](#_Toc101449587)

[3.4 výběr 2](#_Toc101449588)

[4. Výběr herních žánrů vhodných pro implementaci 2](#_Toc101449589)

[4.1 RPG 3](#_Toc101449590)

[4.2 akční 3](#_Toc101449591)

[4.3 strategie 3](#_Toc101449592)

[4.4 závodní 3](#_Toc101449593)

[5. Grafické výstupy aplikací 3](#_Toc101449594)

[5.1 konzolová aplikace 3](#_Toc101449595)

[5.2 okenní aplikace 3](#_Toc101449596)

[5.2.1 WinForm 3](#_Toc101449597)

[5.2.2 WPF 3](#_Toc101449598)

[6. Návrh aplikačního modelu 3](#_Toc101449599)

[7. Návrh vzorového řešení 3](#_Toc101449600)

[8. Zhodnocení realizace aplikace 4](#_Toc101449601)

[9. Závěr 4](#_Toc101449602)

[Seznam použitých zdrojů i](#_Toc101449603)

[Přílohy ii](#_Toc101449604)

**Seznam obrázků**

Odkazovaný seznam obrázků

**Seznam tabulek**

Odkazovaný seznam tabulek

**Seznam použitých zkratek**

Soupis a definování zkratek (vyskytuje-li se jich v textu velké množství)

# 1. Úvod

Text text text text text text text text text text text text text text text text text text text text text text text.

# 2. Cíl práce a metodika

## 2.1 Cíl práce

Cílem práce je popsat aktuální dostupné herní enginy, uživatelská rozhraní a programovací jazyky vhodné pro návrh vzorového řešení. Nejprve na návrhu aplikačního modelu popsat objektový model aplikace. Následně vytvořit návrh vzorového řešení, které bude univerzální v oblasti vývoje her typu RPG.

## 2.2 Metodika

Text text text

# 3. Výběr vhodných programovacích jazyků pro vývoj her

Programovací jazyky dělíme na dva základní skupiny. První jsou imperativní (např. C++), kam patří většina jazyků a jejich rysem je, že kód je sekvence instrukcí a je z něj čitelné co se v jaký okamžik bude provádět. Druhá skupina jsou deklarativní (např. HTML), které říkají jen co se musí vyřešit, ale ne konkrétní instrukce potřebné k provedení a z toho důvodu často nejsou považovány za programovací jazyky, ale používá se pro ně označení kódovací. Další skupina jsou funkcionální (např. Haskell), které ačkoliv se řadí mezi deklarativní mají znaky obou skupin a je možné jejich přístup použít i v imperativních jazycích. Na rozdíl od imperativních nevyužívají žádné globální proměnné a vše je prováděno uvnitř funkcí. Na Obr. 1 je porovnání sumy zapsané pomocí imperativního a funkcionálního jazyku.[1, 2]



Obr. imperativní vs. funkcionální jazyk [2]

Z popisu základních paradigmat je vidět, že jazyk bude vybírán z imperativních jazyků, které se dále dělí na dvě podskupiny. Procedurální[3] (např. C) pracují s funkcemi přijímajícími data pouze z parametrů nebo globálních proměnných. Pro svázání více souvisejících hodnot je možné použít strukturu, která je jako pole umožňující ukládat různé datové typy. Objektové (např. Java) mají třídy sloužící jako předlohy pro instance nazývané objekty, které stejně jako struktury mohou ukládat více hodnot různých typů, ale mají vlastní metody, a proto není potřeba všechna data předávat pomocí parametrů, protože si je může načíst z objektu kde se nachází. Objektově orientované programování (OOP) má čtyři základní principy: zapouzdření, abstrakce, dědičnost a polymorfismus. Zapouzdření umožňuje omezit viditelnost proměnných a metod mimo třídu, kontrolovat přístup k jejich hodnotám a ověřit, zda je zapisována platná hodnota. Abstrakce znamená, že pro práci s objektem není nutné znát vnitřní funkci jeho metod a při práci v týmu kolegovi stačí znát název, parametry a výstup metody. Použitím dědičnosti třída, která je potomek získá všechny proměnné a metody rodiče, ale je možné přidat nové, či změnit chování metody. Polymorfismus souvisí s dědičností, kde do proměnné typu rodič je možné vložit potomka, ale při volání metody se zavolá její přetížená verze, která má stejné jméno, typ a parametry, ale jiné tělo. Dále je možné přetěžovat metody změnou parametrů nebo návratové typu.[4] Na Obr. 2 je porovnání počítání obsahu čtverců a obdélníků napsané v procedurálním a objektovém jazyce (kvůli délce vynecháno zadávání hodnot). Je evidentní, že pro hry se nejvíce hodí objektové jazyky, a proto ty nejpoužívanější nyní budou probrány více do hloubky.



Obr. procedurální vs objektový jazyk-vlastní

## 3.1 C++

C++ je více paradigmatový jazyk[5] rozšiřující jazyk C o objekty, nová klíčová slova a datové typy. Byla snaha zachovat co největší zpětnou kompatibilitu, pro usnadnění přechodu z C na C++ umožňující tvorbu komplexnějších programů[6], ale některé kódy možné napsat v C jsou v C++ neplatné[7]. Se zpětnou kompatibilitou souvisí headery obsahující deklarace proměnných, struktur, tříd a jejich metod, které je potřeba používat i v jiných souborech[8], což sebou ale nese i nevýhodu, že přidání nových tříd a metod, či změny jejich hlaviček je nutno provádět na dvou místech.

### 3.1.1 Kompilace a hardware

Stejně jako jazyk C je kompilován pro konkrétní architekturu procesoru a operační systém, takže je nutno rozlišovat 32bitovou (označovanou jako x86) a 64bitovou verzi operačního systému (x86 dokáže běžet na x64 obráceně ne)[9, 10], ale existuje také C++/CLI, který je součást Microsoft .NET a je kompilován na bytecode (viz kap. 3.2), což umožňuje mít jednu verzi pro obě architektury a sestavit aplikaci z částí napsaných v různých .NET jazycích (viz Kap. 3.3)[11]. Tak jako C je i C++ díky své schopnosti pracovat přímo s pamětí a registry pomocí pointerů vhodný pro psaní ovladačů, operačních systémů a řízení jednočipových počítačů[12–14].

### 3.1.2 Novinky oproti C

Mezi novinky, které C++ přináší patří *namespace*, které umožňují kód organizovat do menších celků a je tak možné, aby se v projektu vyskytoval stejný název vícekrát. Jakožto objektový jazyk dovoluje přetěžování metod, ale oproti Javě a C# porovnává jen parametry, takže funkce s různým návratovým typem a stejnými parametry považuje za stejné a nepůjdou zkompilovat. Dále přibyli *Exceptions* sloužící jako zpráva o chybě ve volané metodě a umožňují tento problém vyřešit, aniž by došlo k pádu programu. Na rozdíl od Javy a C# se může jednat o libovolný datový typ[8].

### 3.1.3 Nevýhody

standardy C++ neobsahují Garbage Collector, takže se programátor musí starat o alokování a následné uvolňování paměti sám, ale je možné použít některý vytvořený třetí stranou[15]. C++ neobsahuje vlastní GUI a musíte proto použít některou z knihoven třetí strany[16].

## 3.2 Java

Java je objektový jazyk, který byl vyvinut s myšlenkou, aby bylo možné jeden program spustit na všech systémech. Architektura vychází z jazyků jako Eiffel, SmallTalk a Objective C. Pro snazší přechod programátorů z C++ byla snaha zachovat co nejpodobnější syntaxi, ale jeho funkcionality použity nebyli.[17] oproti C a C++ se v Javě nenachází funkce, které existují samy o sobě a nenáleží žádné třídě, ale jen metody, které jsou součástí objektu, nebo jsou statické[18, 19].

### 3.2.1 JIT (Just In Time)

Oproti C++ není kód kompilován přímo do strojového kódu, ale do vysokoúrovňového platformě nezávislého kódu nazývaného bytecode, který je spouštěn ve virtuálním stroji (Java Virtual Machine neboli JVM), což umožňuje, aby stejný program bylo možné spustit na všech operačních systémech v 32bitové i 64bitové verzi[9], ale ke spuštění programu musí být na zařízení naistalována odpovídající verze JVM. Nevýhodou bytecodu je jeho výpočetní náročnost, neboť je překládán do strojového kódu v momentě, kdy je spouštěn. Díky just in time (JIT) překladu je možné provést optimalizaci pro konkrétní CPU a tím dosáhnout vyšší rychlosti, než jaké dosahují programy napsané například v C nebo C++ a zkompilované na počítači, který je starší než ten, kde je spouštěn.[20].

### 3.2.2 Přístup k paměti a ovládání hardware

Java neumožňuje pracovat s pointery, neboť správu paměti zajišťuje run time[19]. Jelikož program nepřistupuje k paměti přímo je možné zajistit, že nebude zasahovat do paměti ostatních programů, což by mohlo způsobit pád systému či neoprávněný přístup k citlivým údajům[21]. Pomocí Java ME Embedded je možné ovládat i jednočipové počítače, ale je podporováno pouze Raspberry Pi Model B a dva čipy od STMicroelectronics[22].

### 3.2.3 Výhody

Na rozdíl od C++ Java nepoužívá headery a pro použití třídy v jiném souboru stačí, aby se nacházely ve stejném *namespace*, nebo na příslušný namespace přidat referenci. Oproti C++ má Java Garbage Collector, který se stará o uvolňování paměti mazáním objektů bez reference, čímž usnadňuje programátorovi práci, ovšem za cenu občasného zastavení běhu aplikace, což je možné vyřešit přidáním dalšího vlákna[23]. Doba potřebná ke smazání „mrtvých“ objektů zaleží na počtu „živých“ a velikosti paměti[24, 25]. Java má pro GUI dvě knihovny, jimiž jsou *awt* a odlehčený *swing*[26, 27].

### 3.2.4 Nevýhody

Stejně jako u C++ je zde možné využívat přetěžování metod, ale signaturu tvoří kromě parametrů i návratový typ, avšak oproti C++ a C# Java neumí přetěžovat operátory[19]. Další nevýhoda Javy je, že za generický typ, který se nejčastěji využívá u *Collection* (např. *ArrayList*) není možné dosadit primitivní datový typ, takže například pro přidání *int* do seznamu je třeba vytvořit nový objekt typu *Integer* s jeho hodnotou[28]. Java nemá datový typ pro bezznaménková celá čísla (*uint*)[19], takže je k dispozici pouze polovina rozsahu a pokud je potřeba zapsat hodnotu nad 2 miliardy (231) musí se použít *long* (64bitový).

## 3.3 C#

C# je plně objektový jazyk a hlavní zástupce rodiny Microsoft .NET, který spojuje to nejlepší z C++ a Javy. Ačkoliv vznikl původně pro Windows v posledních letech s přibývajícími frameworky postupně nahrazuje Javu ve vývoji mobilních aplikací (Xamarin a MAUI), PHP v back-endu (ASP .NET) a JavaScript na front-endu (Blazor) webových aplikací.

### 3.3.1 Microsoft .NET

Microsoft .NET je prostředí a rodina jazyků, které ho využívají. Tyto jazyky jsou vzájemně kompatibilní díky požadavkům na CTS (Common Type Specification), CLS (Common Language Specification), CLR (Common Language Runtime) a CLI (Common Language Infrastructure). Hlavní úlohou CLR je správa paměti a vláken. Mimo toho také kontroluje typovou bezpečnost. CTS zajišťují, že všechny jazyky mají stejnou definici datových typů a nemůže se tak stát, aby jednou byl *int* reprezentován třiceti dvěma bity a podruhé pouze šestnácti. Součástí těchto požadavků je, že veškeré referenční i hodnotové datové typy jsou potomky třídy *System.Object* a tím pádem jsou všechny .NET jazyky plně objektové. CLS zajišťuje, aby všechny jazyky byli kompilovatelné do bytecodu označovaného jako MSIL (Microsoft Intermediate Language), což umožňuje v jednom programu kombinovat knihovny napsané v C#, Visual Basic, F#, C++/CLI nebo jiném z více než dvaceti jazyků [11, 29, 30].

MSIL je objektový nízko úrovňový jazyk, který tak jako většinu bytecode je možné kompilovat v režimu JIT (Just In Time), ale navíc také podporuje AOT (Ahead Of Time), kdy se výsledný soubor chová podobně, jako v případě C++, a je tedy nutné ho sestavit pro každý systém a architekturu, kde chceme program spouštět[29]. Výhodou předem zkompilované aplikace je rychlejší start a pro složitější programy i výrazný nárůst výkonu, ovšem za cenu většího souboru, neboť obsahuje také MSIL, který je v některých případech potřeba[31]. při generování AOT jsou využívány nástroje NGen (Native Image Generator) pro .NET Framework a Crossgen2 pro .NET Core. Výstupy těchto nástrojů se nazývají nativní obrazy a jsou instalovány do NIC (Native Image Cache), kam jsou přidávány i závislosti, které je možno používat více obrazy, čímž se eliminuje duplicita. Kompilaci je možné spustit na počítači programátora, nebo až při instalaci programu. Vytvoření obrazu u uživatele má výhodu, že kód bude optimalizován pro jeho procesor a bude tak dosahovat nejvyššího možného výkonu. [11, 32, 33] Další výhodou AOT je, že není potřeba, aby byl překlad co nejrychlejší, takže má dost času provést optimalizace[34].

Velkou výhodou je, že .NET runtime je od Windows Vista součást operačního systému, takže je aktualizován společně se systémem[35], díky čemuž uživatel nemusí nic instalovat. Prostředí .NET bylo původně určeno pouze pro platformy Microsoftu (Windows a Xbox), což se změnilo až v roce 2014 vydáním .NET Core[29], ovšem s GUI pro ostatní systémy se vývojáři museli spoléhat na třetí strany. V roce 2022 bylo vydáno .NET MAUI umožňující vytvořit jednu aplikaci na Windows, Android, iOS a macOS s minimálními zásahy do kódu.[36]

### 3.3.2 Přístup k paměti a ovládání hardware

Na rozdíl od Javy je v C# možné využívat i pointery a obcházet tak správce paměti, což může vylepšit výkon, ale současně vést k bezpečnostním problémům a nestabilitě, kvůli čemuž není možné ověřit bezpečnost a takovýto kód musí být umístěn do bloku vyznačeného pomocí preprocesorů *unsafe*. Kód uvnitř toho bloku se podobá tomu, který by se napsal v C++ nebo C[37]. Ačkoliv C# oficiálně neumožňuje ovládání jednočipových počítačů, existují rozšíření třetích stran, jako například nanoFramework nebo placené visualmicro, které podporují čipy založené na ARM architektuře[38, 39].

### 3.3.3 Porovnání s Javou

Stejně jako u Javy je zde viditelnost tříd řízena pomocí *namespace*. Při přetěžování metod je signatura dána typem a pořadím parametrů, ale oproti Javě umí přetěžovat i operátory[40]. C# dokáže primitivní datové typy (např. *int*) automaticky měnit na objekty[30]. Tak jako Java i C# má Garbage Collector, který za programátora uvolňuje paměť. K jeho spuštění dochází při nedostatku paměti, nebo překročení stanoveného limitu[41, 42]. Ačkoliv C# v některých situacích vyžaduje oproti Javě další klíčová slova, čímž působí jako pomalejší na psaní, snižuje se tím množství chyb a urychluje orientaci v kódu, protože je na první pohled vidět přetěžování při dědičnosti a použité modifikátory.

### 3.3.4 Podobnosti s C++

Tak jako C++ má i C# struktury, které by se daly označit jako hodnotová verze objektu, ale mají omezené možnosti. Například nemohou mít hodnotu *null*, používat dědičnost a mít proměnné inicializované při deklaraci[37].

Podobně jako má C++ pointery na funkce, v C# jsou využíváni delegáti, kteří slouží k předávání metod v parametru, nebo umožňují dynamicky měnit volanou funkci. Delegáty je možné sloučit do *MulticastDelegate*, který obsahuje jejich seznam a při volání je postupně provádí[43–45]. Další jejich využití jsou eventy (např. kliknutí na tlačítko), kde metody, které na něj reagují, musí být typu *void* a mít parametry typu Object a *EventArgs* nebo jeho potomka. První parametr říká, jaký objekt event vyvolal a druhý obsahuje podrobnosti, jako například jaká je poloha kurzoru[46, 47].

### 3.3.5 Modifikátory parametrů metod

U parametru metody je možné použít klíčové slovo *out*, které ho změní na výstupní hodnotu, což umožňuje vracet více než jednu hodnotu bez nutnosti použít pole objektů, ze kterého by se poté postupně přiřazovaly do příslušných proměnných, nebo vracet bool, pokud metoda proběhla úspěšně, a tuto hodnotu předávat výstupním parametrem. Dále je možné využít modifikátory *ref*, který mění hodnotovou proměnnou na referenční, a *in*, který brání úpravám hodnoty[48].

### 3.3.6 nové funkce

Mezi novinky, které C# přináší patří *properties*, umožňující zabalit *get* a *set* pod jeden název, se kterým se při volání pracuje jako by se jednalo o proměnnou[49]. Další nová funkce je modifikátor *partial* umožňující rozdělit definici třídy, struktury nebo interface na více částí, které mohou být i ve více souborech. Pomocí této funkce se dá zvýšit přehlednost velkých tříd rozdělením na menší logické celky a zjednodušuje tak práci u týmových projektů, kde každý programátor může pracovat na své části, aniž by omezoval kolegu. Dále se této možnosti využívá při generování časti třídy, aniž by ovlivnila programátorův soubor. Příkladem je Windows Form, jehož grafická část je generována Visual Studiem[50].

## 3.4 výběr

Text…

# 4. Výběr herních žánrů vhodných pro implementaci

Text

## 4.1 RPG

Te

## 4.2 akční

te

## 4.3 strategie

te

## 4.4 závodní

te

5. Grafické výstupy aplikací

Te

## 5.1 konzolová aplikace

Te

## 5.2 okenní aplikace

Te

### 5.2.1 WinForm

Te

### 5.2.2 WPF

Te

6. Návrh aplikačního modelu

Te

7. Návrh vzorového řešení

Te

8. Zhodnocení realizace aplikace

Te

# 9. Závěr

Text…

# Seznam použitých zdrojů

[1] *imperative programming* [online]. [accessed. 2021-03-16]. Available at: https://whatis.techtarget.com/definition/imperative-programming

[2] COMPUTERPHILE. *Programming Paradigms - Computerphile* [online]. 2013 [accessed. 2021-03-29]. Available at: https://www.youtube.com/watch?v=sqV3pL5x8PI

[3] *procedural and object oriented programming* [online]. [accessed. 2021-03-29]. Available at: https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/

[4] FREECODECAMP.ORG. *Intro to Object Oriented Programming - Crash Course - YouTube* [online]. 2020 [accessed. 2021-07-04]. Available at: https://www.youtube.com/watch?v=SiBw7os-\_zI

# Přílohy

Odkazovaný seznam příloh