

UNIVERZITA OBRANY

FAKULTA VOJENSKÝCH TECHNOLOGIÍ

Studijní program: Vojenské technologie - elektrotechnické

Studijní obor: Informační technologie



DIPLOMOVÁ PRÁCE

Název: Aplikace vyhodnocující přesnost dotyku pilota na
dotykové obrazovce tabletu gyroskopického trenažéru

Zpracoval: rtm. Robert Juran

Vedoucí práce: plk. gšt. doc. Ing. Petr Františ, Ph.D.

BRNO 2024

Akademický rok: 2023/2024

Studijní program: Vojenské technologie - elektrotechnické

Forma studia: prezenční

Studijní obor/modul/specializace: vojenské technologie/Modul komunikační a informační systémy – IT

Katedra: Informatiky a kybernetických operací

Garant studijního programu: plk. gšt. doc. Ing. Josef Bajer, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

Hodnost, jméno a příjmení studenta: rtn. Robert JURAN

Téma (česky): Aplikace vyhodnocující přesnost dotyku pilota na dotykové obrazovce tabletu gyroskopického trenažéru

Téma (anglicky): Application evaluating the accuracy of the pilot's touch on the touch screen of the gyro trainer tablet


Vedoucí závěrečné práce: plk. gšt. doc. Ing. Petr FRANTIŠ, Ph.D.

Konzultanti: por. Ing. Petr GALLUS

Datum zadání: 1. 6. 2023

Termín odevzdání: 31. 5. 2024

Podpis studenta:



.....

I. Upřesnění podmínek zpracování závěrečné práce

V souvislosti se zpracováním závěrečné práce NEBUDE studentem nakládáno s utajovanými informacemi.

V souvislosti se zpracováním závěrečné práce NEBUDE studentem nakládáno s informacemi pro služební potřebu.

Závěrečná práce NEBUDE obsahovat utajené informace.

Závěrečná práce NEBUDE obsahovat informace pro služební potřebu.

Při zpracování závěrečné práce NEBUDOU zpracovávány osobní údaje.

II. Úkoly a pokyny pro zpracování závěrečné práce

Práce bude zpracována v českém jazyce.

Cílem této diplomové práce je navrhnout a implementovat aplikaci, která bude schopna vyhodnocovat přesnost dotyku pilota na dotykové obrazovce tabletu v rámci gyroskopického trenažéru. Aplikace bude sloužit k tréninku a hodnocení schopností pilota v ovládání a řízení letadla prostřednictvím dotykového rozhraní.

V práci:

- Proveďte analýzu možností implementace aplikace
- Rozeberte vhodné technologie a knihovny
- Na základě předchozí analýzy navrhnete možné řešení
- Implementujte aplikaci do tabletu gyroskopického trenažéru
- Proveďte vyhodnocování přesnosti dotyku na základě získaných dat z dotykové obrazovky
- Popište možné využití navrženého řešení a rozeberte jeho klady i zápory

III. Doporučená literatura

- MARSICANO, Kristin, Brian GARDNER, Bill PHILLIPS a Chris STEWART. *Android programming: the Big Nerd Ranch guide*. 4th edition. Atlanta, GA: Big Nerd Ranch, [2019]. ISBN 978-0135245125
- SMYTH, Neil. *Android Studio 4.0: development essentials : Kotlin edition*. [místo vydání není známo]: Neil Smyth / Payload Media, [2020]. ISBN 978-1-951442-20-0
- Choudhury, S., & Mondal, S. *Precision and accuracy evaluation of touch input for Android-based mobile devices*. International Journal of Advanced Research in Computer Science and Software Engineering [2015]. 5(11), 804-808
- Dokumentace k dotykové obrazovce tabletu gyroskopického trenažéru

V Brně dne 30.5.2023

V Brně dne 31.5.2023

V Brně dne 04.06.2023

Vedoucí závěrečné práce

Vedoucí odborné katedry

Garant studijního programu

PODĚKOVÁNÍ

Tímto bych rád vyjádřil své upřímné poděkování všem, kteří mi pomohli při tvorbě této diplomové práce.

Především děkuji svému konzultantovi, por. Ing. Petru Gallusovi, za jeho cenné rady při teoretické části i při návrhu a tvorbě aplikace a jeho odborné vedení během celého procesu zpracování této práce.

Dále děkuji mjr. Ing. Mgr. Ladislavu Göghovi za poskytnutí tabletu a za představení pilotního trenažéru, bez čehož by se práce neobešla.

Také bych rád poděkoval kpt. Ing. Ondřeji Machovi za jeho nápady při tvorbě aplikace a za jeho pomoc při komunikaci s firmou ad-libitum, která pilotní trenažéry vyrábí. Bez jeho přispění by nebylo možné dosáhnout výsledků, které jsou prezentovány v této práci.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem zadanou diplomovou práci na téma „Aplikace vyhodnocující přesnost dotyku pilota na dotykové obrazovce tabletu gyroskopického trenažeru“ vypracoval samostatně, pod odborným vedením vedoucího por. Ing. Petra Galluse a použil jsem pouze literární zdroje uvedené v práci. Při vypracování práce bylo využito nástrojů umělé inteligence pro korekturu textu a úpravu kódu.

Dále prohlašuji, že jsem seznámen s tím, že se na moji diplomovou práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména skutečnosti, že Univerzita obrany má právo na uzavření licenční smlouvy o užití této diplomové práce jako školního díla podle § 60 odst. 1 výše uvedeného zákona, a s tím, že pokud dojde k užití této diplomové práce mnou nebo bude poskytnuta licence o užití díla třetímu subjektu, je Univerzita obrany oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladu, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše. Souhlasím se zpřístupněním své diplomové práce pro prezenční studium v prostorách knihovny Univerzity obrany.

V Brně, dne 27. května 2024

.....

Jméno studenta

ABSTRAKT

Tato práce pojednává o mobilním operačním systému Android, jeho historii, architektuře a vývojových nástrojích používaných při vytváření aplikací. Důraz je poté kladen především na samotné aplikace, jejich strukturu, vývoj a programovací nástroje. Hlavní částí práce je proces vývoje samotné aplikace pro použití na pilotním gyroskopickém simulátoru a její testování. Praktická část práce je psaná s myšlenkou dalšího rozvoje aplikace a obsahuje vysvětlení všech jejích nejdůležitějších částí.

Klíčová slova: Android, aplikace, mobilní operační systém, pilotní gyroskopický simulátor, vývoj.

ABSTRACT

This thesis discusses the mobile operating system Android, its history, architecture, and development tools used in creating applications. The focus is then primarily on the applications themselves, their structure, development, and programming tools. The main part of the thesis is the development process of an application for use on a pilot gyroscopic simulator and its testing. The practical part of the thesis is written with the thought of further development of the application and contains explanations of all its most important parts.

Keywords: Android, application, development, mobile operating system, pilot gyroscopic simulator.

OBSAH

PODĚKOVÁNÍ	5
ČESTNÉ PROHLÁŠENÍ	6
ABSTRAKT	7
ABSTRACT	8
OBSAH	9
SEZNAM POUŽITÝCH ZKRATEK A ZNAČEK	11
SEZNAM OBRÁZKŮ A TABULEK	12
ÚVOD	14
1 Operační systém Android	15
1.1 Historie Android	15
1.2 Architektura Operačního Systému	18
1.3 Vývojové nástroje a prostředí	20
2 Základy Androidových aplikací	21
2.1 Programovací jazyky pro Android	21
2.2 Typy aplikací	23
2.3 Komponenty aplikací	24
2.3.1 Aktivita (Activities)	24
2.3.2 Služby (Services)	24
2.3.3 Přijímače vysílání (Broadcast receivers)	25
2.3.4 Poskytovatelé obsahu (Content providers)	25
2.3.5 Záměr (Intent)	26
2.3.6 Android Manifest	26
Android API	29
2.3.7 Úroveň API	29
3 Tvorba Aplikace	30

3.1	Návrh aplikace.....	30
3.2	Tvorba aplikace	33
3.3	activity_main.xml.....	38
3.4	activity_game.xml	41
3.5	MainActivity.kt	44
3.6	GameActivity.kt	46
4	Testování	55
5	Tvorba aplikace 2	58
5.1	activity_main.xml 2	58
5.2	activity_game.xml 2	59
5.3	MainActivity.kt 2	59
5.4	GameActivity.kt 2	62
5.5	activity_leaderboard.xml a item_leaderboard_header.xml	69
5.6	LeaderboardActivity.kt.....	72
6	Instalace a vyhodnocení.....	81
	ZÁVĚR.....	85
	SEZNAM POUŽITÉ LITERATURY	86
	SEZNAM PŘÍLOH	87

SEZNAM POUŽITÝCH ZKRATEK A ZNAČEK

AOSP	Android Open Source Project
API	Application Programming Interface
AWT	Abstract Window Toolkit
DVM	Dalvik Virtual Machine
HTML	HyperText Markup Language
ID	Identifier
IDE	Integrated Development Environment
IMAP4	Internet Message Access Protocol verze 4
iOS	iPhone Operating System
JVM	Java Virtual Machine
OS	Operating System
PC	Personal Computer
POP3	Post Office Protocol verze 3
RAM	Random Access Memory
SDK	Software Development Kit
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
USB	Universal Serial Bus
VPN	Virtual Private Network
WiFi	Wireless Fidelity
WVGA	Wide Video Graphics Array
XML	eXtensible Markup Language

SEZNAM OBRÁZKŮ A TABULEK

Obrázek 1: Pilotní simulátor [11]	31
Obrázek 2: Návrh aplikace	32
Obrázek 3: Android Studio – šablony	34
Obrázek 4: Android Studio - vytvoření aplikace	35
Obrázek 5: Android Studio - základní aktivita	36
Obrázek 6: Základní XML layout	38
Obrázek 7: activity_main.xml 1	39
Obrázek 8: Základní XML layout 2	41
Obrázek 9: activity_game.xml 1	42
Obrázek 10: MainActivity 1	45
Obrázek 11: GameActivity – class	47
Obrázek 12: GameActivity – createMatrix	49
Obrázek 13: GameActivity – lightRandomButton	50
Obrázek 14: GameActivity – onClick	51
Obrázek 15: GameActivity – endGame	53
Obrázek 16: GameActivity – returnToMenu	54
Obrázek 17: Testování – MainActivity	55
Obrázek 18: Testování – GameActivity	56
Obrázek 19: Testování - GameActivity 2	57
Obrázek 20: activity_game.xml 2	59
Obrázek 21: MainActivity 2 – onCreate	60
Obrázek 22: MainActivity 2 – startGame	61
Obrázek 23: MainActivity 2 – openLeaderboard	61

Obrázek 24: GameActivity 2 – onCreate.....	63
Obrázek 25: GameActivity 2 – buttons	64
Obrázek 26: GameActivity 2 – lightRandomButton	65
Obrázek 27: GameActivity 2 – onClick.....	65
Obrázek 28: GameActivity 2 – showReactionTimeChart	66
Obrázek 29: GameActivity 2 – endGame.....	67
Obrázek 30: GameActivity 2 – MyButton.....	68
Obrázek 31: GameActivity 2 – DataManager	68
Obrázek 32: activity_leaderboard – Spinner	69
Obrázek 33: activity_leaderboard – Buttons	70
Obrázek 34: activity_leaderboard – RecyclerView	71
Obrázek 35: LeaderboardActivity – onCreate	73
Obrázek 36: LeaderboardActivity – Buttons	74
Obrázek 37: LeaderboardActivity - loadUsers, clearResults.....	75
Obrázek 38: LeaderboardActivity – loadUsersFromStorage.....	76
Obrázek 39: LeaderboardActivity – LeaderboardAdapter	77
Obrázek 40: LeaderboardActivity – setupSpinner.....	78
Obrázek 41: LeaderboardActivity – sortLeaderboard	79
Obrázek 42: Gyroskopický trenažér [12].....	81
Obrázek 43: SWOT Analýza aplikace.....	82
Obrázek 44: Graf výsledků	83
Obrázek 45: Žebříček výsledků	83

ÚVOD

V dnešní moderní době, kdy letecký průmysl podstupuje nepřetržitý vývoj a inovace, nabývá výcvik pilotů klíčového významu. Pilotní simulátory a trenažéry se staly nepostradatelným prvkem ve výcviku pilotů, které především ve vojenském prostředí nabízejí bezpečnou možnost přípravy na válečný konflikt a zároveň dovolují vyhodnotit jejich schopnost ovládat stroje pod vlivy vznikajícími v extrémních rychlostech.

Pilotní simulátory jsou navrženy tak, aby co nejvíce simulovaly reálné letové podmínky. Stacionární simulátory poskytují statické prostředí, zatímco pohyblivé simulátory, umístěné na pohyblivých platformách, dokáží reprodukovat i pohyb letadla během letu. Tyto schopnosti simulátorů dokáží piloty seznámit a zdokonalit v jednotlivých fázích letu, od vzletu až po přistání, a naučit je reagovat na různé situace, jako jsou například turbulence.

Rozvoj technologií, které pronikají do každého aspektu našeho života, vede mimo jiné k otevření nových možností v oblasti simulací a výcviku. Jedním ze směrů, s nímž je tato práce provázána, je využití dotykových displejů a tabletů v leteckých simulátorech pro testování schopností pilota při simulovaném letu. Tato diplomová práce se zaměřuje na vývoj, optimalizaci a využití takové aplikace, která přináší jeden z možných způsobů měření. V práci bude provedena analýza možných implementací aplikace, návrh řešení a podrobní rozebrání částí vyvinuté aplikace. V závěru bude provedena už samotná implementace aplikace na příslušný tablet, její otestování a vyhodnocení navrhnutého řešení.

1 Operační systém Android

Android je mobilní operační systém na jádře Linuxu vyvíjený firmou Google pod hlavičkou Open Handset Alliance v rámci projektu AOSP. Poprvé byl zveřejněn v roce 2008 a nastartoval revoluci na poli mobilních operačních systémů. Několik let po vzniku nahradil postupem času mobilní operační systém Symbian, čímž se stal přímým konkurentem systému iOS. Android samotný zajišťuje pouze základní systémové funkce, které musí být pro dodatečné funkce zařízení doplněny o další proprietární software, neboli closed source software s licencí náležící určité firmě a většinou bez volně dostupného zdrojového kódu. Jedná se například o služby jako Google Play, YouTube nebo Amazon Appstore. [1]

Popularita operačního systému Android u všech skupin, od uživatelů po vývojáře, spočívá v hned několika jeho vlastnostech. Jedná se o open-source software, což znamená, že zdrojový kód Androidu je volně dostupný a šířitelný. Navíc se jedná o velmi stabilní a rychlý operační systém, což společně s jeho otevřeností vedlo k jeho oblíbení u mnoha vývojářů mobilních aplikací, kteří jej mohou využívat bez licenčních poplatků. Z uživatelského hlediska nabízí Android podporu u mnoha výrobců, což znamená rozmanitý výběr cenově dostupných zařízení s tímto operačním systémem. V neposlední řadě nabízí zařízení širokou nabídku aplikací mimo jiné s podporou integrace Google služeb, jako je Gmail, Google Drive, Google Maps a další. [1]

1.1 Historie Android

Operační systém Android byl původně vyvinut malou společností Android Inc., založenou roku 2003 a později odkoupenou v roce 2005 Googlem, kde pokračoval vývoj operačního systému v alianci několika dalších firem pod společným názvem Open Handset Alliance. První verze Androidu byla zveřejněna roku 2008 a poskytovala na tu dobu pokročilé funkce, jako internetový prohlížeč a integrované mapy. Prvním zařízením na trhu s operačním systémem Android byl mobilní telefon T-Mobile G1, který je považován za vůbec první skutečný smartphone. [1]

Po prvním oficiálním vydání Androidu 1.0 začal Google vydávat nové verze operačního systému, které od původní verze prošly mnoha změnami a vylepšeními

opravujícími zjištěné chyby a přidávajícími nové funkčnosti do systému. Jednotlivé vycházející verze dostaly název podle zákusků v abecedním pořadí. Každou z verzí rozebereme a uvedeme některé nové vlastnosti. [3]

1) Android 1.0

1. Jednalo se o první oficiální vydání operačního systému, zveřejněné 28. září 2008, a byl založen na jádře Linux 2.6.25.
2. Obsahoval webový prohlížeč pro prohlížení HTML a XHTML stránek s funkcí zobrazení více stránek ve formě karet.
3. Zařízení mělo fotoaparát bez jakýchkoli dalších funkcí.
4. Obsahoval Android Market – on-line katalog plný aplikací a her.
5. Podpora emailu s přístupem k emailovým serverům s podporou protokolů POP3, SMTP a IMAP4.
6. Google aplikace jako Gmail, Google Contacts, Google Calendar a Google Maps atd.

2) Android 1.5 (Cupcake)

1. Platforma vydaná 27. dubna 2009, založená na jádře Linux 2.6.27.
2. Přinesla zpřesnění všech dosavadních prvků uživatelského rozhraní.
3. Celkové zrychlení všech softwarových komponent.
4. Funkce otočení obrazovky při natočení zařízení.
5. Možnost ve fotoaparátu navíc nahrávat videa.
6. Přidání domovské obrazovky.
7. Rozhraní API a manifest prvky pro vývoj aplikací.

3) Android 1.6 (Donut)

1. Verze systému vydaná 15. září 2009 na Linuxovém jádře 2.6.29.
2. Zlepšení grafického rozhraní pro fotoaparát a galerii.
3. Přidání rychlého vyhledávání – možnost pro uživatele, jak z jednoho textového pole prohledávat více zdrojů (web, kontakty, aplikace, historii).
4. Panel pro konfiguraci VPN.
5. Nově podpora rozlišení obrazovky WVGA.
6. Hlasové vyhledávání.
7. Možnost zobrazení, které aplikace využívají kolik energie, na základě čehož má uživatel možnost manuálně aplikace ukončit pro úsporu.

4) Android 2.0/2.1 (Eclair)

1. Verze zveřejněná chvíli po předchozí, 26. října 2009, postavená na stejném jádře.
2. Vylepšení telefonního seznamu.
3. Podpora Microsoft Exchange.
4. Podpora více rozlišení displeje, což mimo jiné umožnilo uvést na trh jedny z prvních Android tabletů.
5. Podpora Bluetooth 2.1.
6. Přidání podpory HTML5 v prohlížeči.

5) Android 2.2 (Froyo)

1. Vyšla dne 20. května 2010 na Linuxovém jádře 2.6.32.
2. Přibyla možnost instalace aplikací na paměťovou kartu.
3. Vylepšení fotoaparátu a kamery.
4. Sdílení internetového připojení zapojením zařízení přes USB k PC.
5. Možnost využít zařízení jako WiFi hotspot a sdílet skrze zařízení internetové připojení.
6. JIT (Just-in-Time) kompilátor zrychlující celkový výkon systému.
7. Vylepšená správa paměti RAM.

6) Android 2.3/2.3 (Gingerbread)

1. Jedna z nejvyužívanějších verzí Androidu, zveřejněná 6. prosince 2010 na Linuxovém jádře 2.6.35.
2. Podpora video formátů WebM a HTML5.
3. Podpora IP telefonie v rámci protokolu SIP.

7) Android 3.0/3.1/3.2 (Honeycomb)

1. Verze určená pouze pro tablety s převážně grafickými úpravami.

8) Android 4.0/4.0.1/4.0.2 (Ice Cream Sandwich)

1. Verze vydaná v roce 2011 na Linuxovém jádře 3.0.1. Opět nasazená na mobilní telefony a přináší si ty nejlepší funkce z verzí Honeycomb.
2. Verze mají vylepšené rozpoznávání hlasu a možnost detekce obličeje.
3. Zdokonalení webového prohlížeče a dalších aplikací, které lze nyní ovládat pohybovými gesty.

1.2 Architektura Operačního Systému

Architektura OS Android se skládá z pěti vrstev, z nichž každá samostatně provádí v zařízení jiné operace. Toto oddělení vrstev je ale především jen teoretické, protože v praxi dochází ke spolupráci jednotlivých částí systému a vrstvy nejsou mezi sebou striktně odděleny.

Linux Kernel

Nejnižší vrstvou architektury je Linux Kernel nebo též jádro operačního systému. Jedná se o nejdůležitější část všech operačních systémů fungujících na Linuxu. Základní funkcí této vrstvy je zodpovědnost za low-level operace systému jako je správa disku a alokování paměti operacím a vytváří základní rozhraní mezi hardwarem počítače a jeho funkcemi. Při startu zařízení je jádro zavedené do operační paměti a je mu předáno řízení, které zahrnuje například kontrolu nad koordinací činnosti všech běžících procesů, už zmíněná správa paměti, správa sítí a mnoho dalších. [4]

Knihovny

Libraries (knihovny) jsou další vrstvou architektury Android. Obecně se jedná o sety instrukcí napsaných v programovacím jazyce. Obsahují většinou konfigurační data, dokumentaci, předepsaný kód atd. [4] Android nabízí mnohá rozhraní API pro vývoj aplikací. Pro představu uvedeme některé nejčastěji používané, které jsou dostupné pro všechna android zařízení:

1. Android.util – hlavní balíček knihoven obsahující utility jako manipulace data/času, base64 enkodéry a dekodéry a nástroje pro parsování XML
2. Android.os – knihovny s přístupem k základním službám operačního systému, jako je předávání zpráv a mezi-procesová komunikace
3. Android.graphics – knihovny pro zobrazování grafických prvků
4. Android.text – knihovny pro zpracování a zobrazení řetězců textu
5. Android.database – knihovny s třídami potřebnými pro práci s kurzory databází
6. Android.content – obsahují třídy pro přístup a publikování dat na zařízení [1]

Android Runtime

Prostředí Android Runtime je jednou z nedůležitějších částí operačního systému Android. Obsahuje komponenty jako jsou základní knihovny a Dalvik Virtual Machine. Především poskytuje základ pro application framework a pohání aplikace s pomocí základních knihoven. Základní knihovny nám umožňují implementovat aplikace pro Android pomocí standardních programovacích jazyků jako je Java nebo Kotlin. [4]

DVM byl exkluzivně pro Android vyvíjen od roku 2005 společností Google a je pro něj speciálně navržený a optimalizovaný tak, aby zajistil efektivní běh více instancí na jednom zařízení. [1] Podobně jako Java Virtual Machine má Dalvik registrově orientovanou architekturu a využívá základní vlastnosti Linuxového jádra, jako je nízko-úrovňová správa paměti, koordinace běžících procesů a práce s vlákny (threading). [2]

Důvodem ke vzniku nového virtuálního stroje byla skutečnost, že programátoři vyvíjející aplikace pro operační systém Android používali často právě jazyk Java. Knihovny tohoto jazyka jsou licencovány jako open source, ale virtuální stroj (JVM), který slouží k překladu programu do spustitelné formy, již není volně šiřitelný. Dalším důvodem vzniku virtuálního stroje Dalvik byla optimalizace virtuálního stroje pro potřeby mobilních zařízení, kde hrál klíčovou roli výkon a úspora energie. [4]

Obsah knihoven programovacího jazyka Java, které tato vrstva zahrnuje, lze srovnat s platformou Java SE (Standard Edition), která zahrnuje JVM, API knihovny základních funkcí a API knihovny pro vytváření klientských desktopových aplikací (AWT, Swing). Na Androidu byly ale AWT a Swing nahrazeny knihovnami pro tvorbu uživatelského rozhraní a přibýly knihovny Apache pro práci se sítí.

Aplikace pro Android jsou psány například v jazyce Java, poté jsou přeloženy do Java bytecode a nakonec do mezikódu pomocí Dalvik kompilátoru. Výsledný bytecode je spuštěn na DVM a každá aplikace funguje jako samostatný proces s vlastní instancí DVM. [1]

Application Framework

Aplikační rámec poskytuje několik důležitých tříd, které se používají k vytváření aplikací pro Android. Nabízí obecnou abstrakci pro přístup k hardwaru a také pomáhá spravovat uživatelské rozhraní pomocí zdrojů aplikace. Obvykle poskytuje služby, pomocí nichž můžeme vytvořit konkrétní třídu a udělat tuto třídu užitečnou pro vytváření aplikací. [2]

Pro vývojáře je to tedy jedna z nejdůležitější vrstev architektury, protože umožňuje přistupovat k různým službám, které se dají používat přímo ve vytvořených aplikacích, a tím následně umožňují například přistupovat na prvky graficko-uživatelského rozhraní, používat hardware zařízení, spouštět služby na pozadí nebo používat určité služby zařízení, jako jsou hodiny, budík apod. [1] Navíc tato vrstva obsahuje různé typy služeb, jako jsou správce aktivit, správce oznámení, systém zobrazení, správce balíčků atd., které jsou užitečné pro vývoj aplikací podle požadavků. [4]

Applications

Aplikace představují poslední a nejvyšší vrstvu Android architektury. Tato vrstva zahrnuje jak předinstalované aplikace, jako jsou domovská obrazovka, kontakty, fotoaparát, galerie a další, tak i aplikace třetích stran stažené z obchodu Play, nebo odjinud z internetu, jako jsou chatovací aplikace, hry apod. Veškeré aplikace, bez ohledu na jejich původ, jsou nainstalovány a spouštěny právě na této vrstvě. [4]

Spouštění aplikací v této vrstvě probíhá v rámci Android Runtime za pomoci tříd a služeb poskytovaných aplikačním rámcem. Tato integrace zajišťuje správné fungování a vzájemnou interakci aplikací a služeb v rámci Android ekosystému. [2]

1.3 Vývojové nástroje a prostředí

Vývoj aplikací pro operační systém Android probíhá ve vývojovém prostředí nazývaném *Host-Target*, což je druh vývojového procesu, při kterém je prostředí, kde je aplikace vyvíjena, a prostředí pro realizaci aplikace naprosto odlišné. Jednoduše řečeno je, například v případě vývoje androidových aplikací, aplikace vytvářena ve vývojovém prostředí na počítači za pomoci dalších nástrojů a testování zároveň s implementací aplikace probíhá na jiném zařízení, ať už mobilní telefon, tablet atd. [1]

Existuje samozřejmě možnost využití emulátorů a virtualizace, které umožňují spuštění androidovského operačního systému na počítači s například operačním systémem Windows, ale v takovém případě mohou nastat komplikace, jako je například špatná kompatibilita v případě starších verzí OS, omezený přístup k hardwaru a nedostačující výkonu a celkové zbytečné spotřeby zdrojů.

Obecně je tedy možné provádět základní testování a ladění aplikací za pomoci emulátorů, ale vždy je lepší a můžeme říct i nutné, testovat vytvořenou aplikaci na zařízení a v prostředí, pro které byla navržena. V našem případě tedy mobilní zařízení v podobě tabletu.

2 Základy Androidových aplikací

Před navrhováním a vývojem jakéhokoliv nástroje je nutné porozumět podstatě oboru a jeho kontextu. Jinak tomu není ani při vývoji aplikací. Bez hlubšího pochopení nelze efektivně identifikovat potřeby a požadavky uživatelů a je nutné pro vybrání klíčových funkcí a vlastností, které by do aplikace měly být začleněny. Dále proto nahlédneme na nejčastější programovací jazyky používané při vývoji androidových aplikací, na konstrukci aplikací, kterou je vhodné se řídit a na další komponenty s aplikacemi a vývojem svázané.

2.1 Programovací jazyky pro Android

Obecně se pro vývoj Android aplikací nejčastěji používají dva velmi známé a rozšířené programovací jazyky: Java a C++. Dále se běžně využívá XML pro popis uživatelského rozhraní, který ale ve své podstatě není přímo programovacím jazykem.

V posledních letech se také rozšířil programovací jazyk Kotlin, který je moderní a efektivní alternativou k Javě.

1. Java:

Java je univerzální objektově orientovaný programovací jazyk poprvé představený v roce 1995. Od svého vzniku si prošel mnoha vývojovými fázemi a stal se jedním z nejrozsáhlejších programovacích jazyků na světě. Mezi jeho vlastnosti patří už

zmíněná objektová orientace, což znamená, že veškerý kód v Java je organizován do objektů.

Další významnou vlastností je podpora multi-threadingu, který umožňuje paralelní a asynchronní zpracovávání dat. Ohromnou výhodou jazyka je platformní nezávislost, která umožňuje spouštět kód napsaný v tomto jazyce na různých platformách bez nutnosti jakýchkoli úprav; často označováno jako „Write Once, Run Anywhere“.

V neposlední řadě obsahuje Java také garbage collector, který se stará o správu paměti tím, že automaticky odstraňuje nepoužívané objekty, což usnadňuje proces vývoje a zabraňuje úniku paměti kvůli přetečení. [5]

2. C++:

C++ je další vysokoúrovňový programovací jazyk, který byl v roce 1979 vyvinut jako rozšíření jazyka C s přidáním objektově orientovaných prvků. Jako většina programovacích jazyků si také prošel řadou rozšíření, revizí a aktualizací. Podobně jako Java je i C++ objektově orientovaný a podporuje tedy principy jako dědičnost a třídy.

Jedním s významných rysů C++ je jeho vysoká výkonnost, jelikož umožňuje přímý přístup k paměti a zkušeným programátorům tak dovoluje přímou kontrolu nad hardwarovými prostředky. Tato schopnost byla při vývoji ostatních zmíněných jazyků záměrně odepřena především z důvodu bezpečnosti, protože nesprávně řízený přímý přístup k paměti může být pro nezkušeného programátora rizikovým. [6]

3. XML:

Jazyk eXtensible Markup Language není programovacím jazykem v pravém smyslu slova, ale jedná se o takzvaný značkovací jazyk, který slouží k značkování strukturovaných dat a je tedy používán k popisu struktury a označování informací. Značkování dat probíhá v XML vytvořením značek, které jsou ohraničeny závorkami následovně:

```
<osoba>
```

```
    <jmeno> Petr Novák </jmeno>
```

```
    <vek> 25 </vek>
```

```
    <povolani> Elektrikář </povolani>
```

```
</osoba>
```

Data jsou tímto způsobem organizována do hierarchické struktury a každá značka může obsahovat další podřazené značky a hodnoty. Standardizace formátu dat se v XML dosahuje pomocí XML schémat nebo Document Type Definitions. XML je platformě nezávislý, což z něj dělá často používaným nástrojem na výměnu dat mezi různými systémy. XML sám o sobě nemá vlastní programovací paradigma a pro manipulaci s daty se tak používají různé programy a knihovny. Jazyky jako například Java podporují nástroje pro práci s XML. [7]

4. Kotlin:

Kotlin je relativně nový, plně objektově orientovaný programovací jazyk vydaný v roce 2017 firmou JetBrains. Byl navržen tak, aby byl interoperabilní s jazykem Java. To znamená, že je možné integrovat kód z jazyku Java do kódu napsaného v Kotlinu a naopak. Mimo to obsahuje Kotlin například některé funkcionální prvky, jako jsou lambdové výrazy vyšších řádů atd. Dále obsahuje ochranu před velmi běžnými chybami spojenými s hodnotami typu null (neexistence hodnoty v proměnné nebo objektu) a umožňuje rozšiřitelnost. Kotlin se stal nástupcem jazyka Java jako oficiální programovací jazyk k vývoji aplikací pro platformu Android. [8]

2.2 Typy aplikací

Nezákladnější rozlišení aplikací pro operační systém Android je klasifikace podle toho, jak aplikace interagují s uživatelem a jak pracují ve vztahu k ostatním aplikacím běžícím na systému. Podle těchto vlastností se aplikace mohou dělit do třech základních kategorií:

1. Aplikace na popředí (Foreground apps): Jedná se o aplikace, které jsou momentálně aktivní a užitečné pouze v případě, že běží na popředí. Jejich činnost je pozastavena, nebo ukončena, pokud je aplikace dána na pozadí. Zobrazují se na hlavní obrazovce nebo jsou otevřeny v režimu prohlížeče. Tyto aplikace mají přímý přístup ke zdrojům, jako jsou procesor, paměť a síťové funkce. [1]
2. Aplikace na pozadí (Background apps): Tyto aplikace nemusí být aktivní přímo na popředí a nemají přímou interakci s uživatelem. Jejich hlavní činnost probíhá mimo momentální běh zařízení a mohou na pozadí provádět různé úkony, jako

je sledování polohy, aktualizace, sledování příchozích hovorů atd. Systém většinou ovlivňuje přístup těchto aplikací k některým zdrojům uvedeným výše, aby šetřil energii a uvolnil výpočetní výkon pro jiné aplikace. [1]

3. Aplikace s přerušovanou činností (Foreground services): Aplikace řadící se pod tuto kategorii jsou na popředí, ale mohou provádět různé operace i poté, co uživatel aplikaci opustí, a očekává se od nich komunikace s uživatelem skrze upozornění. Často tyto aplikace bývají spojeny se službami (services), které mohou fungovat na pozadí. Příkladem takovýchto aplikací je nejtýpčtěji přehrávač hudby, jako Soundcloud a Spotify, které na popředí přehrávají hudbu, i když je zároveň aktivní jiná aplikace. [1]

2.3 Komponenty aplikací

Každá aplikace se skládá z několika komponent, které jsou jejich základními stavebními bloky. Každá komponenta je vstupním bodem, skrze kterou může systém nebo uživatel do aplikace vstupovat a komunikovat s ní. Každý druh má jasný účel a životní cyklus, jenž určuje, jak je komponenta vytvořena a zničena. Některé komponenty mohou být závislé na jiných a vzájemně spolu komunikují. [9]

2.3.1 Aktivita (Activities)

Aktivity představují prezentační vrstvu a umožňují interakci s uživatelem. Jedná se o vizuální komponentu aktuální obrazovky s uživatelským rozhraním. Vysvětlení provedeme například na aplikaci Google Drive s několika aktivitami, z nichž jedna ukazuje náhled na soubory v disku, další umožňuje vytvoření nového souboru a další dovoluje přecházení mezi jinými náhledy v aplikaci. Typicky je jedna aktivita určena jako hlavní a zobrazuje se uživateli ihned po spuštění. Aktivity by měly být vytvořeny s myšlenkou intuitivnosti a snadnosti použití. Nabízejí také možnost spolupráce s jinými aplikacemi. Například v případě Google Drive možnost přímého nahrání obrázků z galerie přes menu sdílení. [9]

2.3.2 Služby (Services)

Jedná se o části aplikace, které nejsou součástí uživatelského rozhraní, ale provádějí operace na pozadí, a tedy neumožňují uživatelům přímé ovládání. Fungují jako vstupní

bod pro umožnění aplikaci pracovat v pozadí pro provádění dlouhotrvajících operací nebo pro práci s vzdálenými procesy. [9]

2.3.3 Přijímače vysílání (Broadcast receivers)

Přijímače jsou jedním ze vstupů do aplikací umožňující systému doručovat aplikacím události mimo klasický uživatelský tok a dovolují tak reagovat na důležitá systémová oznámení. Důležitou vlastností přijímačů je schopnost přistupovat k aplikaci, i když není právě spuštěna ani na pozadí, a provádět k nim vysílání. Této vlastnosti se hojně využívá u mnohých aplikací, přestože samy přijímače vykonávají minimální množství práce. Příkladem jejich použití je kalendářová aplikace, která jej používá k odesílání upozornění na naplánované události nebo k synchronizaci dat s online kalendářem, a to i v případě, kdy aplikace není spuštěna. [9]

2.3.4 Poskytovatelé obsahu (Content providers)

Poskytovatelé obsahu provádějí operace s daty a slouží k jejich sdílení mezi aplikacemi. Aplikace ukládají data do svého souborového systému, do databáze, na web nebo na jiné druhy úložišť. Jelikož neexistuje žádné běžné úložiště, do kterého by měly přístup všechny Android balíčky, tak této funkci napomáhají poskytovatelé obsahu. Pokud k tomu má aplikace dostatečné oprávnění, jejich prostřednictvím si může data vyžádat, nebo je modifikovat.

Unikátním prvkem Android aplikací je schopnost aplikace vyžádat si určité komponenty a funkce z jakékoliv jiné aplikace. Například může bankovní aplikace požádat uživatele, aby pořídil fotku občanského průkazu, nebo seznamovací aplikace může požádat o fotku obličeje. Většinou podobné funkcionality již existují v nějaké jiné aplikaci a nemusíte tedy vytvářet vlastní aktivity a kód pro focení nebo na ně odkazovat. Stačí pouze vytvořit aktivitu ve fotoaparátu pro pořízení fotografie, která se poté vrátí aplikaci, a ta s ní může dále pracovat. Pro uživatele se celý proces jeví, jako by fotoaparát byl přímo součástí aplikace.

Jelikož každá aplikace existuje v systému jako samostatný proces s vlastními oprávněními, nemůže aplikace sama přistoupit k obsahu aplikace jiné. O tyto požadavky se stará systém samotný. Aplikace si od něj zprávou se záměrem zažádá o přístup ke komponentě a systém jí reakcí daný komponent aktivuje. [9]

2.3.5 Záměr (Intent)

Záměr neboli Intent, je asynchronní zpráva aktivující tři ze čtyř uvedených komponent: aktivity, služby a přijímače vysílání. Záměry na sebe vážou jednotlivé komponenty a fungují jako prostředníci, kteří od nich vyžadují činnosti. Jejich vytváření je prováděno přes objekt Intent, který nadefinuje zprávu pro aktivaci konkrétní komponenty nebo pro aktivaci komponentu nějakého typu.

Záměry, jak bylo uvedeno u poskytovatelů obsahu a jak název napovídá, definují záměr aplikace k provedení nějaké akce, kterou může být třeba přístup k datům nebo k prvku nějaké jiné aplikace. U aktivit a služeb může záměr signalizovat požadavek k otevření fotoaparátu nebo webové aplikace. V případě přijímačů vysílání obvykle záměry značí požadavek k odvysílání nějakého upozornění, jako například upozornění z kalendáře nebo spuštění budíku.

Poskytovatelé obsahu fungují z bezpečnostních důvodů jinak a na rozdíl od aktivit, služeb a přijímačů jsou vyvolávány požadavkem Překladače obsahu (Content resolver), který zpracovává přímé transakce s poskytovateli obsahu. [9]

2.3.6 Android Manifest

AndroidManifest je XML soubor uložený v kořenovém adresáři aplikace (typicky [název aplikace]/manifests/AndroidManifest.xml), který obsahuje informace o různých parametrech aplikace, jako je název a verze API, jednotlivé komponenty, potřebná oprávnění, softwarové závislosti pro použití komponent jiných aplikací a další požadavky pro správný chod aplikace. [1]

Ukážeme si strukturu jednoduchého souboru AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".SecondActivity" />
    </application>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

Nyní si vysvětlíme jednotlivé části:

1. Manifest element:

- ‘<manifest>’ je kořenový element souboru AndroidManifest.xml.
- ‘xmlns:android=http://schemas.android.com/apk/res/android’ slouží pro definování jmenného prostoru Android a je povinné pro všechny prvky v manifestu.

2. Package:

- ‘package="com.example.myapplication"' definuje jméno balíčku aplikace.

3. Application:

- ‘<application>’ obsahuje globální informace o aplikaci a jejích komponentech (dříve zmíněné aktivity, služby, posluchače, poskytovatelé obsahu)
- ‘android:...’ jsou atributy pro nastavení vlastností aplikace.

4. Activity:

- ‘<activity>’ slouží pro definování jednotlivých aktivit aplikace.
- ‘android:name’ určuje název třídy aktivity.
- ‘<intent-filter>’ definuje typy záměrů pro každou aktivitu.
- “android.intent.action.MAIN” říká, že se jedná o hlavní (MAIN) aktivitu a tedy se při spuštění aplikace obslouží jako první a ostatní aktivity jsou na ni dále vázány. Například se může jednat o spuštění hlavní nabídky.
- ‘android.intent.category.LAUNCHER’ říká, že aktivita bude zobrazena v seznamu aplikací (ikonka).

5. Permissions:

- ‘<uses-permission>’ definuje, jaká povolení potřebuje aplikace k fungování. V tomto případě ‘INTERNET’ umožňuje aplikaci připojení k síti a ‘ACCESS_NETWORK_STATE’ umožňuje aplikaci získávat informace o stavu sítě.

Android API

Android API je základním stavebním kamenem pro vývoj aplikací pro operační systém Android. Tato sada programových nástrojů je klíčová pro interakci aplikací s jádrem systému a hardwarovými komponenty zařízení. Jedná se o rozhraní, které umožňuje vývojářům využívat širokou škálu funkcí a služeb poskytovaných Androidem, včetně práce s uživatelským rozhraním, sítěmi, databázemi, multimédií a dalšími.

Android API je nedílnou součástí Android SDK, což je balík nástrojů, které vývojáři potřebují k tvorbě aplikací pro Android. Tato knihovna poskytuje různé funkce a abstrakce, které umožňují vývojářům psát kód pro Android aplikace bez nutnosti znalosti detailů implementace samotného systému. [10]

S každou novou verzí operačního systému Android je aktualizována i verze Android API, což přináší nové funkce, vylepšení a opravy chyb. To vede k existenci několika nejčastěji používaných verzí API, které poskytují širokou škálu funkcí a jsou podporovány na velké části moderních zařízení s Androidem. [3]

2.3.7 Úroveň API

API je číselná hodnota nastavitelná v souboru AndroidManifest.xml, která určuje verzi Android SDK používanou aplikací. Android SDK a poskytované nástroje API jsou navrženy tak, aby byly obecně zpětně kompatibilní, takže vývojáři mohou používat novější verze k vytváření aplikací, které jsou spustitelné na starších verzích Androidu. Pro vývojáře je důležité zvážit aktuální trendy a potřeby uživatelů při výběru verze API pro své aplikace, aby mohli využít nejnovějších funkcí a zároveň zajistit maximální kompatibilitu se zařízeními uživatelů. [3]

3 Tvorba Aplikace

Prvním krokem při vytváření jakékoli aplikace je vybrat vhodné integrované vývojové prostředí (IDE), což je specializovaný program, který poskytuje vývojářům sadu nástrojů pro vývoj softwaru v jednom prostředí. Máme na výběr několik možností, jako je populární Visual Studio s použitím pluginu Xamarin, IntelliJ IDEA nebo Android Studio. Pro naše potřeby je Android Studio nejvhodnější volbou, protože poskytuje širokou škálu nástrojů pro usnadnění vývoje, včetně integrovaného asistenta AI ve svých nejnovějších verzích, a podporuje oba nejpoužívanější jazyky pro vývoj aplikací pro Android – Java a Kotlin. Začneme tedy instalací z oficiálních stránek na adrese <https://developer.android.com/studio>, kde máme na výběr z několika verzí IDE.

3.1 Návrh aplikace

Důležitým krokem před psaním samotné aplikace je rozhodnutí, jak by měla aplikace vůbec vypadat a co by měla dělat. Naše zadání je jasné – vytvořit aplikaci pro měření přesnosti doteku, ale i přes to nám zadání nechává velkou volnost v provedení. V této části si tedy projdeme všechny návrhy, které jsme zvážili, a zhodnotíme jejich výhody a možné problémy, které by mohly nastat během vývoje.

1. Simulátor ovládacího panelu letounu

První představa pro měření přesnosti doteku pro pilotní simulátor je samozřejmě věrná rekreace reálného ovládacího panelu, jaký se používá ve skutečných letadlech. Taková podoba aplikace by nejen v simulátoru působila věrohodně a profesionálně, ale také by piloty lépe připravila na situaci, kdy budou sedět za reálným ovládacím panelem. V tomto prvotním návrhu jsme narazili na několik problémů, které by vývoj takové aplikace nejen ztížily, ale pravděpodobně i znemožnily. Za prvé pracujeme s Androidovým tabletem malé velikosti, což by vedlo k nečitelnosti tlačítek a celý ovládací panel by mohl být nakonec nesrozumitelný a zbytečný. Na přiloženém obrázku lze vidět fotografii kokpitu pilotního simulátoru pro stíhačku Lockheed Martin F-35 Lightning II, z níž jsou uvedené problémy zjevné.



Obrázek 1: Pilotní simulátor [11]

Zároveň by takové provedení mohlo potenciálně posunout obsah diplomové práce na utajovaný, což by působilo jasný konflikt se zadáním.

2. „Hra“ s pohyblivým bodem

Jedním z navrhovaných řešení je vytvoření hry s pohyblivým bodem, na který by uživatel musel podle instrukcí klikat. Z toho by se následně vypočítala přesnost uživatele. Toto řešení by, spolu s rotací simulátoru, mohlo přinést praktickou a vypovídající aplikaci. Nicméně jako první problém tohoto řešení se může ukázat konstantní animace pohybujícího se bodu, která může vyvstávat kvůli hardwarovým limitacím tabletu.

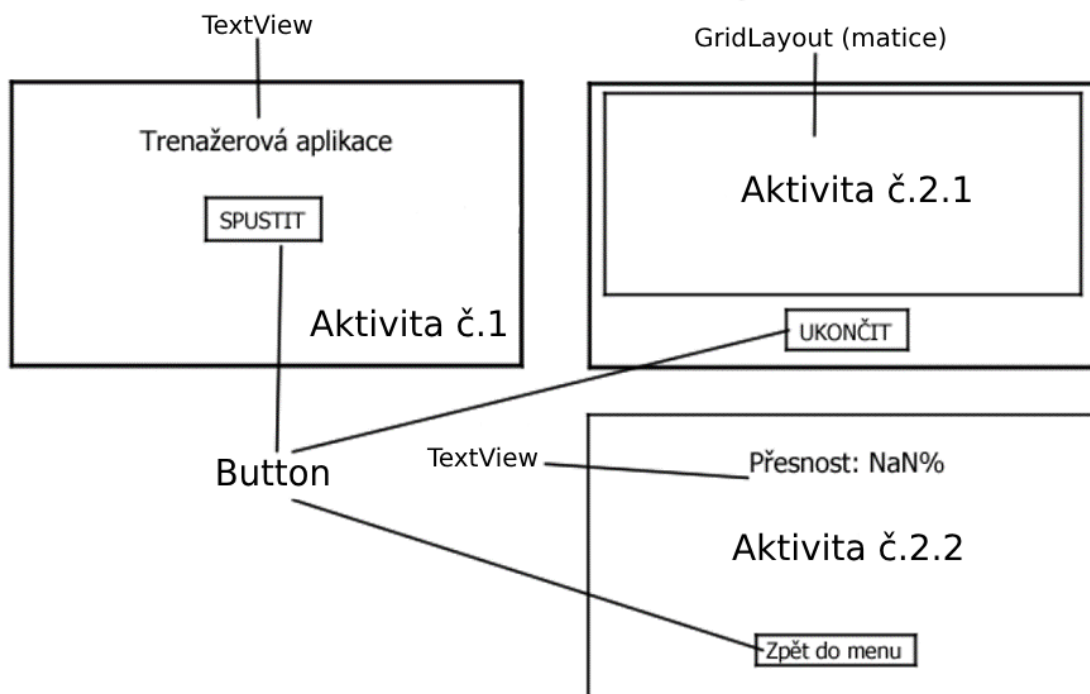
3. Maticová „hra“ s měřením přesnosti dotyku

Po konzultaci s Centrem Tělesné Výchovy a Sportu a s kpt. Ing. Ondřejem Machem, který má bohaté zkušenosti s používáním simulátoru jsme došli k několika závěrům:

- Aplikace by neměla být náročná – tablet musí být schopný aplikaci provozovat minimálně několik hodin v kuse bez potřeby nabíjení.
- Aplikace by měla být jednoduchá – bez složitějších instrukcí a návodů by měla být uživatelsky přívětivá, jak pro učitele, tak pro žáka.

Finálním návrhem naší aplikace bude jednoduché a nenáročné provedení. Aplikace bude prezentovat jednu matici postupně se rozsvěčujících tlačítek, na které musí uživatel kliknout v určeném pořadí. Po provedení předem určeného počtu kliknutí proběhne výpočet přesnosti, která se zobrazí na finální obrazovce jako procentuální hodnota úspěšnosti.

Po finálním výběru provedení je vhodné si vymyslet, jak na sebe budou navazovat jednotlivé aktivity aplikace a jak budou přibližně vypadat. K tomu nám postačí tužka a papír, nebo můžeme použít digitální prostředky. Tento proces je většinou prováděn dedikovaným designerem aplikace, ale pro vytvoření jednoduché aplikace si vystačíme sami.



Obrázek 2: Návrh aplikace
vlastní zdroj

Podle popisu výše jsme dospěli k následujícímu návrhu aplikace, ve kterém identifikujeme základní aktivity a prvky aplikace. První aktivitou je menu aplikace, které obsahuje úvodní text a tlačítko pro spuštění hry. Po kliknutí na toto tlačítko se přesuneme do druhé aktivity. V této aktivitě se automaticky vygeneruje herní pole ve formě matice tlačítek s předem určenými rozměry. Na spodní části této aktivity je

umístěno tlačítko, které umožňuje uživateli ukončit aktuální hru a vrátit se zpět do menu.

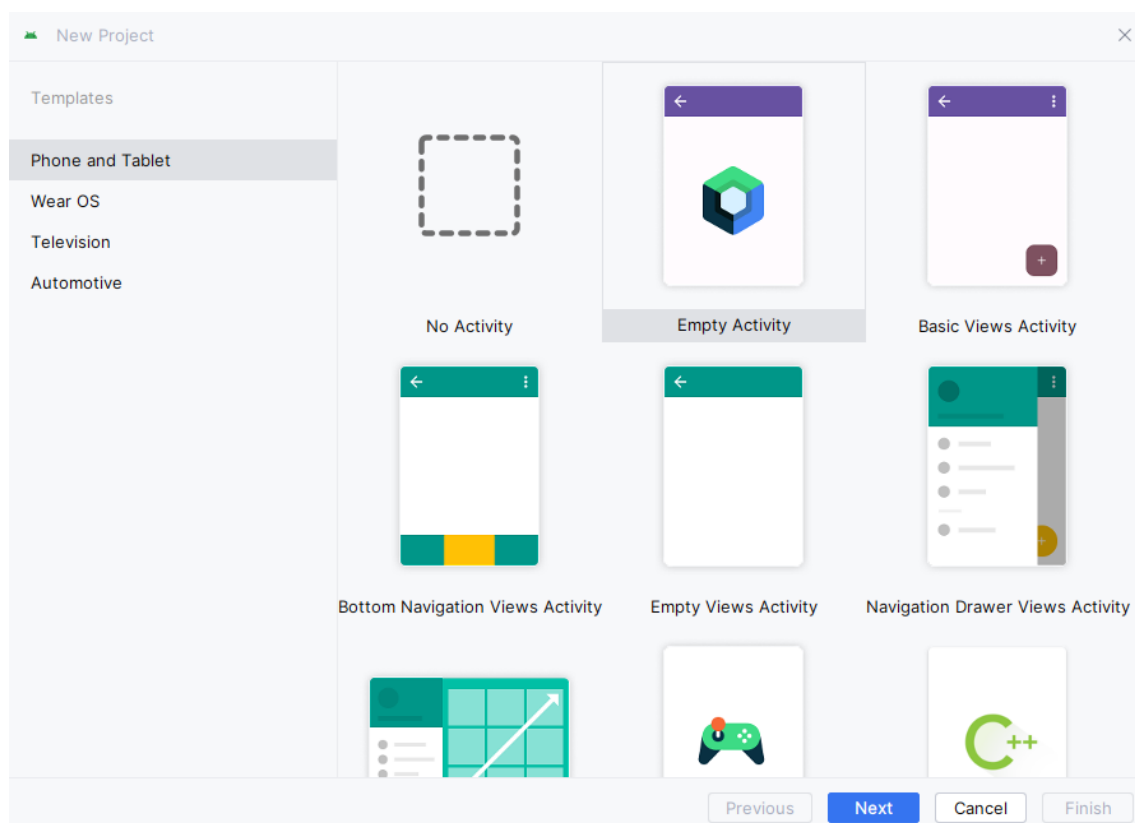
Po dosažení určitého cíle nebo podmínek pro ukončení hry dojde k vyhodnocení přesnosti výkonu hráče, která je vyjádřena v procentech. Kromě toho dojde ke změně textu na tlačítku pro ukončení hry, čímž se zlepší vizuální efekt a uživatelská zkušenost. Bylo by možné tuto druhou část aktivity oddělit do samostatné aktivity, avšak pro zachování přehlednosti a jednoduchosti celého kódu jsme se rozhodli ponechat tyto části společně.

3.2 Tvorba aplikace

Jak už bylo zmíněno, aplikaci budeme psát v IDE Android Studio. Posledním krokem před začátkem vývoje aplikace je nezbytné pečlivě vybrat vhodný programovací jazyk. V našem případě jsme se rozhodli pro Kotlin, který v současné době nabízí několik výhod oproti svému předchůdci, Javě. Kotlin získává stále více podpory a uznání ze strany vývojářské komunity díky své moderní syntaxi a řadě užitečných funkcí.

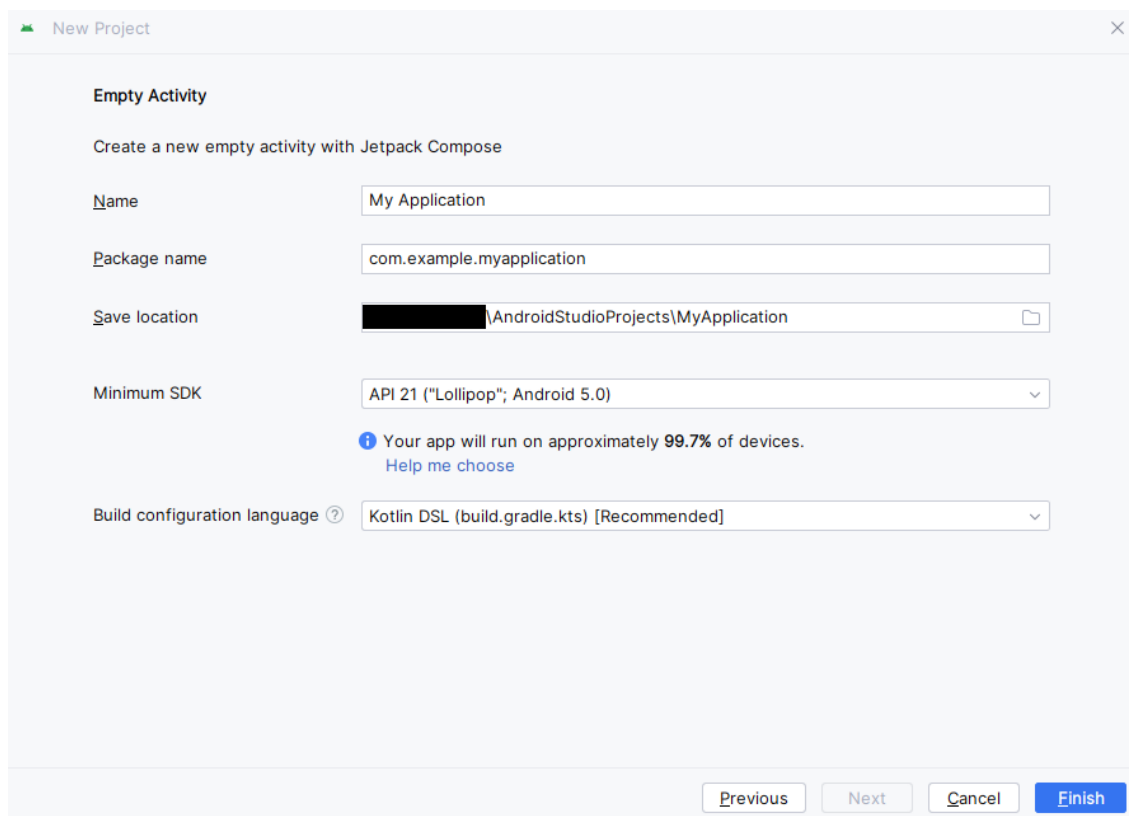
Pro úpravu vizuální stránky naší aplikace budeme využívat XML. Tento značkovací jazyk je běžně používán pro definici rozložení uživatelského rozhraní. S pomocí XML můžeme efektivně definovat vzhled naší aplikace a zajistit, aby byla atraktivní a uživatelsky přívětivá.

Nyní tedy přejdeme přímo k psaní aplikace a v Android Studio vytvoříme nový projekt.



*Obrázek 3: Android Studio – šablony
vlastní zdroj*

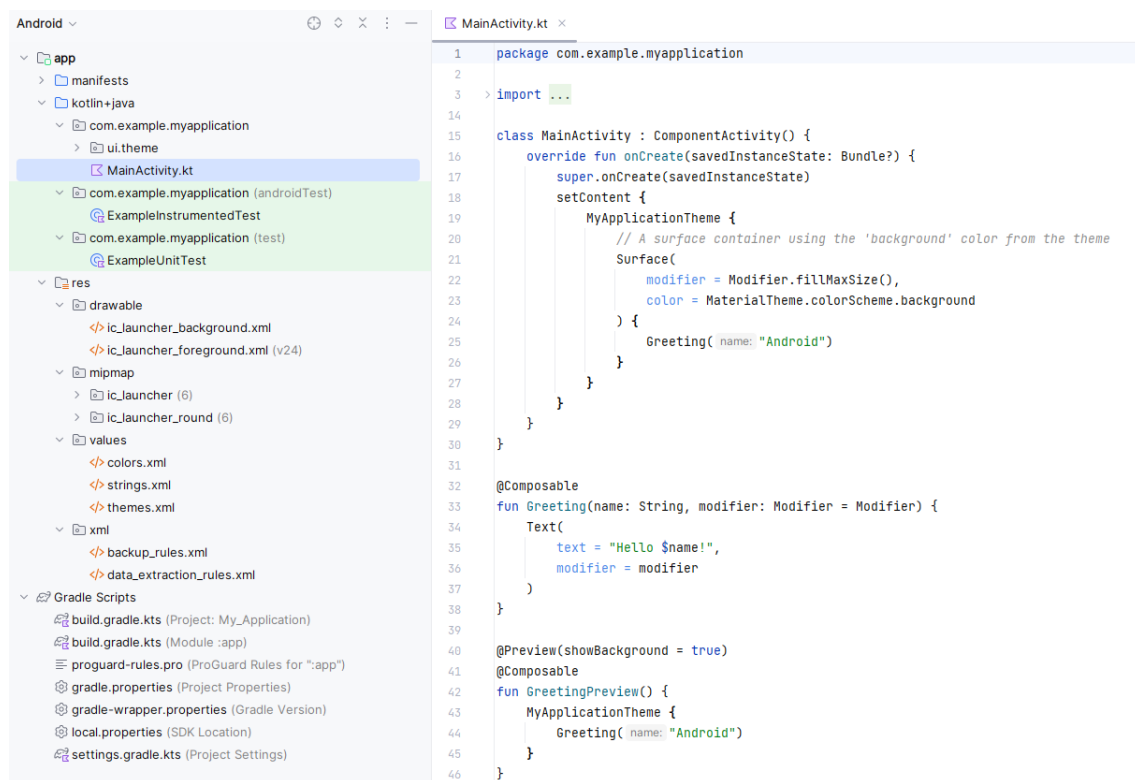
Dostaneme na výběr z několika šablon, což je jedna z velkých výhod Android Studio, která v mnohých případech zjednodušuje vývoj a nabízí vývojářům nemálo základu, na kterých mohou dále stavět svou aplikaci. V našem případě ale vytvoříme prázdnou aktivitu.



*Obrázek 4: Android Studio - vytvoření aplikace
vlastní zdroj*

V následující obrazovce aplikaci pojmenujeme, vybereme umístění a název balíčku a pravděpodobně nejdůležitější částí je zde verze API a s tím spojená verze Android OS. V našem případě vytváříme aplikaci pro jeden konkrétní tablet s verzí Androidu 10, ale jelikož jsme k němu dostali přístup až v pozdější fázi vývoje, vybereme verzi Androidu 5, která má nejširší podporu zařízení a stále si uchovává nejdůležitější funkce. Toto by pro nás nemělo být z důvodu jednoduchosti aplikace problémové.

Tímto se nám vytvoří nový projekt, který už bude mít před vytvořením některé části aplikace, jako je hlavní aktivita podle vybrané šablony (v našem případě tedy prázdná) a provede se první synchronizace Gradle a zbytku souborů aplikace.



Obrázek 5: Android Studio - základní aktivita vlastní zdroj

Jako první se nám po vytvoření projektu otevře hlavní aktivita, jak nám ji Android Studio navrhl. Pro nás důležité části se nacházejí v první části kódu, kde je deklarace balíčku *com.example.myapplication* a importy, které přináší různé třídy a funkce pro použití v kódu. V levé části můžeme vidět ostatní před vytvořenou adresáře a soubory:

1. **drawable:** Tento adresář obsahuje obrázky, ikony a jiné grafické soubory používané v aplikaci. V Android Studio je to běžný způsob, jak organizovat zdroje obrázků podle různých rozlišení displeje a jiných konfigurací.
2. **colors.xml:** Tento XML soubor obsahuje definice barev používaných v aplikaci. Může obsahovat hexadecimální kódy barev nebo odkazy na barvy definované jinde v aplikaci.

3. `strings.xml`: Tento XML soubor obsahuje řetězcové zdroje používané v aplikaci. Je to dobrá praxe oddělit řetězcové hodnoty od zdrojového kódu, což umožňuje jednodušší lokalizaci a správu textů ve vaší aplikaci.
4. `themes.xml`: Tento soubor obsahuje definice témat aplikace. Témata mohou obsahovat nastavení jako barvy pozadí, styly textu, velikosti písma atd. Tento soubor je klíčový pro nastavení vzhledu vaší aplikace.
5. `backup_rules.xml`: Tento soubor může obsahovat pravidla pro zálohování dat aplikace. Záloha dat je důležitým aspektem vývoje aplikací, zejména pokud aplikace ukládá důležitá data, která by uživatel neměl ztratit.
6. `data_extraction_rules.xml`: Tento soubor může obsahovat pravidla pro extrakci dat z různých zdrojů. To může být užitečné, pokud aplikace pracuje s daty ze zdrojů jako jsou webové stránky, soubory nebo databáze.
7. `build.gradle.kts`: Tento soubor obsahuje skripty pro sestavení projektu pomocí nástroje Gradle. Gradle je nástroj používaný vývojáři Android aplikací pro automatizaci procesu sestavení a správu závislostí projektu. Soubor `.kts` znamená, že se jedná o soubor napsaný v programovacím jazyce Kotlin pro lepší čitelnost a flexibilitu. Sestává se v základní konfiguraci projektů ze dvou částí – projektová a modulová. Do modulové části budeme přidávat některé implementace nutné pro správnou funkci aplikace.

3.3 activity_main.xml

Při psaní kódu si nejdříve pro každou část aplikace projdeme její základní podobu, popíšeme vzniklé problémy při vývoji s jejich řešením a co se změnilo ve finální podobě. Kompletní kód v jeho finální podobě lze nalézt na konci celé práce.

První věc, kterou musíme po vytvoření aplikace udělat, je vytvoření layoutu první obrazovky, což je v našem případě menu. Toho dosáhneme vytvořením adresáře layout pod adresářem res a následně vytvořením tzv. *Layout Resource File* pod adresářem layouts. Pojmenujeme ho pro přehlednost *activity_main*.

Tímto způsobem nám Android Studio vytvoří nový soubor, ve kterém předdefinuje displej s některými základními vlastnostmi.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</LinearLayout>
```

Obrázek 6: Základní XML layout
vlastní zdroj

První řádka definuje verzi XML a kódování dokumentu. Zbytek XML kódu popisuje rozložení uživatelského rozhraní v Android aplikaci pomocí prvku ‘LinearLayout’, který organizuje ostatní prvky ve vertikálním směru.

Velikou výhodou Android Studia je možnost grafického zobrazení designu aplikace. Momentálně je zobrazení prázdné, takže se na něj podíváme až vždy ve finální podobě.

Podle nákresu, který jsme si vytvořili při návrhu aplikace přidáme do *activity_main.xml* nadpis a tlačítko pro spuštění.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context="com.example.diplomka_test.MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:text="Trenažerová aplikace pro měření přesnosti doteku"
        android:layout_centerHorizontal="true"/>

    <Button
        android:id="@+id/startButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/textView"
        android:text="Spustit"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"/>

</RelativeLayout>

```

Obrázek 7: activity_main.xml 1
vlastní zdroj

Rozdílem oproti původnímu návrhu je naše použití `RelativeLayout`, který nám umožňuje definovat vztahy mezi prvky na základě jejich relativní polohy vůči sobě. Použili jsme vlastnosti `match_parent`, aby se šířka a výška rovnala rodičovskému kontejneru.

`TextView` je náš nadpis. Nejdříve mu vytvoříme identifikátor (`id`), který slouží pro jeho pozdější použití v kódu aplikace. Pro výšku a šířku jsme tentokrát použili vlastnosti `wrap_content`, což povede k přizpůsobení textového prvku podle jeho obsahu tzn. zvětší a zmenší se podle potřeby. Velikost písma jsme nastavili na 24sp (scale-independent pixels), což je prvotní hodnota, kterou později podle potřeby změníme.

Následně je text, který jsme nastavili podle našeho zadání. Nakonec zarovnáme celý text na střed pomocí `layout_centerHorizontal="true"`.

`Button` je naše tlačítko pro spuštění další aktivity s hrou. Identifikátorem tohoto tlačítka je `startButton`. Stejně jako u textu nastavíme výšku a šířku na `wrap_content`, takže při jakýchkoliv úpravách na textu tlačítka se nám obal přizpůsobí. Další vlastností je `layout_bellow="@id/textView"`, což nám jednoduše určuje, že tlačítko se bude vždy naházet pod nadpisem a vlastností `text="Spustit"` nastavíme samotný text tlačítka. Opět určíme tlačítku zarovnání na střed displeje a vlastností `layout_marginTop="32dp"` nastavíme odsazení tlačítka od nadpisu.

Opět musíme myslet na to, že toto rozložení je pouze prvotní a jako u ostatních částí kódu slouží pouze pro zprovoznění a testování. Většina prvků bude později změněna, aby byla přívětivější pro uživatele (velikost textu, velikost tlačítek apod.). Stejně tak i finální forma aplikace může být dále rozšířena a vytváření složitých animací, zvukových efektů a podobně nebudou součástí této práce.

3.4 activity_game.xml

Po vytvoření menu musíme vytvořit taktéž layout pro herní obrazovku. Stejně jako u menu vytváříme stále pouze soubor XML, takže se jedná jen o vzhled a logika se nachází v odděleném souboru. Opět vytvoříme Layout Resource File v adresáři layout a pojmenujeme ho *activity_game*. Již nebudeme rozebírat původní formu, jelikož je totožná s předchozí.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context="com.example.diplomka_test.GameActivity">
```

Obrázek 8: Základní XML layout 2
vlastní zdroj

Začátek souboru je téměř stejný jako v *activity_main.xml* s výjimkou `tools:context`, který v našem případě nedefinuje *MainActivity*, ale *GameActivity* a udává nám tím jiný kontext, ve kterém má být návrh rozložení zobrazen. Jinak řečeno, jedná se o rozložení displeje pro budoucí *GameActivity*, který bude obsahovat kód s logikou.

```

<TextView
    android:id="@+id/accuracyTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:text=""
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"/>
// Naplní se textem až v GameActivity

<GridLayout
    android:id="@+id/gridLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"/>

<Button
    android:id="@+id/endButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ukončit"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="16dp"/>

</RelativeLayout>

```

Obrázek 9: *activity_game.xml* 1
vlastní zdroj

V tomto souboru můžeme vidět podobné prvky, jako v předchozí části. `TextView` s id `accuracyTextView`, jak název napovídá, tentokrát neslouží jako nadpis obrazovky, ale pro zobrazení přesnosti uživatele po ukončení hry. Tento text jsme opět vycentrovali pomocí `layout_centerHorizontal`, nastavili `layout_marginTop` (prostor mezi prvkem a vrchní hranou obrazovky) na 16dp a ostatní vlastnosti jako v minulém případě. Můžeme si všimnout že `Text` je inicializován na prázdný řetězec. Pokud bychom vložili

text už zde, tak by se nám zobrazoval po celou dobu hry, což by samozřejmě působilo rušivě. Zároveň je tento TextView speciální případ, protože do něj budeme chtít zapsat text až po ukončení hry a vypočítání přesnosti, takže TextView budeme aktualizovat dynamicky v kódu.

Následující GridLayout je nový středový prvek, který je obecně kontejner umožňující uspořádat položky do řádků a sloupců. Náš prvek s id *gridLayout* bude po spuštění hry sloužit k uspořádání tlačítek do matice. Jeho šířka a výška jsou také nastaveny na *wrap_content* pro přizpůsobení se obsahu a *layout_centerInParent* slouží k umístění prvku přesně na střed obrazovky.

Tlačítko s id *endButton* slouží k okamžitému ukončení hry v jakékoliv fázi a vrácení se do menu. Pomocí *alignParentBottom* je umístěný na spodku obrazovky a přes *centerHorizontal* je zarovnán na střed. Nakonec má nastavenou mezeru od spodku obrazovky na 16dp.

3.5 MainActivity.kt

V následující části jsme vytvořili *MainActivity* s příponou souboru .kt, což indikuje, že se jedná soubor s kódem napsaným v jazyce Kotlin. Tento soubor obsahuje kód a definici chování pro hlavní obrazovku naší aplikace, jejíž rozložení jsme vytvořili v *activity_main*.

Soubory s kódem obsahují vždy importy nějakých knihoven dovolující fungování klíčových částí, a proto si vysvětlíme všechny námi použité:

4. ***android.content.Intent***: Knihovna, která je součástí Android SDK používaná k vytváření intentů a zahájení jiné aktivity nebo aplikace. Používáme k vytvoření intentu pro spuštění *GameActivity* (viz. následující podkapitola).
5. ***android.os.Bundle***: Datová struktura sloužící pro předávání dat mezi různými komponentami aplikace. V našem případě použita v metodě *onCreate* jako jeden z parametrů.
6. ***android.widget.Button***: Import sloužící pro manipulaci s tlačítky.
7. ***android.view.View***: Je základním stavebním blokem uživatelského rozhraní pro Android a poskytuje základní funkce pro manipulaci s uživatelskými prvky aplikace. Zde tento import konkrétně používáme pro skrytí status baru (vrchní lišta).
8. ***androidx.appcompat.app.AppCompatActivity***: Import pro třídu *AppCompatActivity*, která umožňuje podporu pro funkce moderního designu na starších verzích Androidu a používáme ji jako základ pro definici hlavní aktivity *MainActivity*.

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Skrytí "status baru"
        window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_FULLSCREEN
        // Skrytí "action baru"
        supportActionBar?.hide()

        setContentView(R.layout.activity_main)

        val startButton: Button = findViewById(R.id.startButton)
        startButton.setOnClickListener { it: View!
            startGame()
        }
    }

    private fun startGame() {
        val intent = Intent(packageContext: this, GameActivity::class.java)
        startActivity(intent)
    }
}

```

Obrázek 10: MainActivity 1
vlastní zdroj

Pod importy se nachází již zbytek kódu, jehož všechny části si vysvětlíme také.

Třída (class) *MainActivity* je odvozena od *AppCompatActivity*, což jí umožňuje dědit všechny vlastnosti a metody této třídy, čímž získává podporu pro novější designové prvky. Tímto způsobem mohou vývojáři obejít nedostatky starších verzí Android a rozšířit si možnosti.

Metoda *onCreate* je metoda životního cyklu volána při vytvoření aktivity. V této metodě je inicializováno uživatelské rozhraní a nastavení tlačítka Spustit na hlavní obrazovce.

Zavoláním `window.decorView.systemUiVisibility` a `supportActionBar?.hide` zajistíme, aby naše aplikace byla roztáhnuta na celou plochu tabletu. Nejdříve skryjeme

stavový panel na vrchní části displeje a posléze skryjeme akční panel, pokud je k dispozici.

Příkazem *setContentView* poté nastavíme rozložení pro tuto aktivitu na základě dříve vytvořeného zdroje *activity_main*. Z tohoto zdroje následně získáme id tlačítka „Spustit“ před val *startButton* a nastavíme posluchač *setOnClickListener* pro toto tlačítko, aby při zmáčknutí spustil metodu *startGame*.

V metodě *startGame* vytváříme nový intent, kterým se jednoduše spouští *GameActivity*. Aktivita je specifikována podle její třídy *GameActivity::class.java* (třída je Java, přestože pracujeme v jiném jazyce) a spouštěna pomocí metody *startActivity(intent)*.

Toto rozložení hlavní obrazovky je velice jednoduché, avšak pro naše účely plní všechny požadavky. Navrhování a práce na první části kódu byla prováděna s předpokladem pozdější úpravy a rozšíření. V procesu vývoje byla hlavní obrazovka několikrát rozšířena o nové elementy a funce.

3.6 GameActivity.kt

Na závěr vytvoříme soubor *GameActivity*, který obsahuje kód s logikou pro hlavní část celé aplikace a pracuje na základě zdroje z *activity_game*. Jedná se tentokrát o značně delší kód, takže bude rozpolcen do více obrázků s popisem.

GameActivity obsahuje importy knihoven totožné s *MainActivity* s několika přídávky:

1. ***android.graphics.drawable.ColorDrawable***: Knihovna umožňující pracovat s barevnými pozadími (drawables) u prvků uživatelského rozhraní.
2. ***android.widget.GridLayout***: Jedná se o layout manager, který dokáže organizovat své potomky do mřížky. Zde tento import samozřejmě používáme k vytvoření matice tlačítek.
3. ***android.widget.TextView***: Import widgetu pro zobrazení textu na obrazovce, který zde využíváme po ukončení hry na zobrazení textu s přesností.
4. ***kotlin.random.Random***: Často používaný import, kdykoliv je třeba generování náhodných čísel, který používáme pro náhodné zapínání tlačítek v matici.

```

class GameActivity : AppCompatActivity() {

    private companion object {
        const val MATRIX_SIZE = 5
        const val MAX_ATTEMPTS = 10
    }

    private lateinit var buttons: Array<Array<Button>>
    private var totalAttempts = 0
    private var successfulAttempts = 0
    private var litButtonRow: Int = -1
    private var litButtonCol: Int = -1

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_FULLSCREEN
        supportActionBar?.hide()
        setContentView(R.layout.activity_game)

        createMatrix()

        val endButton: Button = findViewById(R.id.endButton)
        endButton.setOnClickListener { it: View!
            endGame()
        }

        lightRandomButton()
    }
}

```

Obrázek 11: GameActivity – class
vlastní zdroj

Hlavní částí je třída *GameActivity*, která je opět odvozena od *AppCompatActivity* a má tedy stejné rozšířené vlastnosti a metody. Pod touto třídou jsou obsaženy prakticky všechny komponenty kódu s jednou výjimkou, ke které se dostaneme v následující kapitole.

Pod vytvořenou třídou jsme si definovali dvě konstanty *MATRIX_SIZE* a *MAX_ATTEMPTS*, které podle názvu definují hodnoty velikosti matice a počet pokusů, po kterých se hra ukončí. Pro zjednodušení má matice totožnou výšku i šířku a je nastavena na malý rozměr. Další definované proměnné jsou *buttons* v podobě dvourozměrného pole tlačítek (array) pro vytvoření herní plochy, *totalAttempts* pro počítání pokusů provedených hráčem a *successfulAttempts* pro počítání úspěšných pokusů, přičemž jsou na začátku obě nastavené na nula a aktualizované v průběhu hry a jako poslední proměnné *litButtonRow* a *litButtonCol*, které udržují informaci o tom, které tlačítko je aktuálně rozsvíceno.

Po definování proměnných následuje první metoda *onCreate*. Proměnná *savedInstanceState: Bundle?* slouží pro uchování stavu aktivity v případě, že je aktivita systémem zničena a poté opět vytvořena, což může nastat například při otočení obrazovky. V metodě opět skrýváme vrchní a spodní lištu pro roztáhnutí aplikace a nastavujeme layout aktivity na zdroj *activity_game*.

Vytváříme herní pole s maticí zavoláním metodou *createMatrix*, která je vysvětlena níže, nastavujeme posluchač na tlačítku „Ukončit“, které automaticky ukončí hru a zobrazí výsledky, a nakonec rozsvítíme náhodné tlačítko zavoláním metody *lightRandomButton*, která je opět rozebrána později.


```

private fun createMatrix() {
    val GridLayout: GridLayout = findViewById(R.id.gridLayout)
    GridLayout.removeAllViews()
    GridLayout.columnCount = MATRIX_SIZE
    GridLayout.rowCount = MATRIX_SIZE

    buttons = Array(MATRIX_SIZE) { row ->
        Array(MATRIX_SIZE) { col ->
            Button(context: this).apply { this: Button
                text = ""
                isEnabled = true
                setBackgroundColor(getColor(android.R.color.white))
                setOnClickListener { it: View!
                    onClick(row, col)
                }
            }
            val params = GridLayout.LayoutParams().apply { this: GridLayout.LayoutParams
                width = GridLayout.LayoutParams.WRAP_CONTENT
                height = GridLayout.LayoutParams.WRAP_CONTENT
                rowSpec = GridLayout.spec(row)
                columnSpec = GridLayout.spec(col)
            }
            GridLayout.addView(child: this, params)
        }
    }
}

```

Obrázek 12: *GameActivity* – *createMatrix*
vlastní zdroj

Následuje již jednou použitá funkce *createMatrix*, která je v třídě *GameActivity* zodpovědná za vytvoření herní matice, která obsahuje tlačítka, jež uživatel v průběhu hry mačká. Funkce nepřijímá žádné parametry a ani nevrací žádné hodnoty.

Na začátku funkce se pomocí *removeAllViews* odstraňují veškeré tlačítkové prvky z matice, čímž se zabrání překrývání stávajících prvků. Následně použijeme dříve nadefinovanou proměnnou *MATRIX_SIZE* pro nastavení rozměrů matice. Bylo by v tomto případě samozřejmě možné vytvořit dvě proměnné a podle nich nastavit odlišný počet sloupců a řádků.

Poté jsou pomocí vnořené smyčky vytvořena tlačítka v každém řádku a sloupci matice. Používá se dvou úrovní smyček: jedna pro iteraci přes řádky a druhá pro iteraci

přes sloupce. Každé tlačítko je inicializováno vnitřní smyčkou, která projde každý sloupec v daném řádku a vytváří tlačítka pomocí objektu třídy *Button*, nastavuje jejich vlastnosti a přiřazuje jim *onClick* posluchač. Po vytvoření tlačítka jsou každému přiřazeny nové parametry *params* a každé z nich je přiřazeno do mřížky pomocí objektu *GridLayout.LayoutParams*, který jim určí umístění. Pomocí metody *addView* jsou s použitím parametrů tlačítka vložena do mřížky.

```
private fun lightRandomButton() {
    val random = Random.Default
    val row = random.nextInt(MATRIX_SIZE)
    val col = random.nextInt(MATRIX_SIZE)

    val button = buttons[row][col]
    button.setBackgroundColor(getColor(android.R.color.holo_red_light))
    button.isEnabled = true
    litButtonRow = row
    litButtonCol = col
}
```

Obrázek 13: *GameActivity* – *lightRandomButton*
vlastní zdroj

Jednoduchá funkce *lightRandomButton* slouží k rozsvěcování náhodného tlačítka při spuštění hry a po zmáčknutí příslušného tlačítka. Definujeme si na začátku několik hodnot, které pomocí generátoru náhodných čísel náhodně určí sloupec a řadu tlačítka. Poté dojde ke zvýraznění tlačítka na námi vybranou hodnotu *holo_red_light* a nastavíme jeho stav na aktivní. Nakonec je pozice tlačítka *litButtonRow* a *litButtonCol* uložena do proměnných, což slouží k pozdějšímu ověření, zda uživatel stiskl správné tlačítko.

```

private fun onClick(row: Int, col: Int) {
    val button = buttons[row][col]
    totalAttempts++

    if (litButtonRow != row || litButtonCol != col) {
        if (totalAttempts >= MAX_ATTEMPTS) {
            endGame()
        }
        return
    }

    removeMatrix()
    lightRandomButton()
    createMatrix()

    if (button.isEnabled) {
        if (button.background is ColorDrawable) {
            val colorDrawable = button.background as ColorDrawable
            val color = colorDrawable.color
            if (color == getColor(android.R.color.holo_red_light)) {
                successfulAttempts++
            }
        }
        button.setBackgroundColor(getColor(android.R.color.white))
        button.isEnabled = false
        if (totalAttempts < MAX_ATTEMPTS) {
            lightRandomButton()
        } else {
            endGame()
        }
    }
}
}

```

Obrázek 14: GameActivity – onClick
vlastní zdroj

Následující funkce *onButtonClick* se volá po stisknutí tlačítka v herní matici a stará se o zpracování celé akce stisknutí. Ověřuje, zda bylo stisknuté správné tlačítko a aktualizuje podle toho stav hry. Funkce přijímá vstupní parametry *row* a *col*, které udávají pozici stisknutého tlačítka.

Hned na začátku funkce vidíme *totalAttempts++*, čímž po každém stisknutí tlačítka hned na začátku zvyšujeme počet provedených pokusů, které samozřejmě nejsou nijak závislé na správnosti stisknutí. Funkce dále porovnává pozici stisknutého tlačítka s pozicí tlačítka, které bylo předtím zvýrazněno v metodě *lightRandomButton*. Pokud se hodnoty shodují, uživatel zmáčkl správné tlačítko.

Při vytváření aplikace jsme narazili na problém, kdy se stav tlačítka po zmáčknutí neměnil a jeho barva zůstávala stejná a zároveň po stisknutí již nebylo možné tlačítko použít. Proto jsme přešli k radikálnějšímu přístupu a po každém zmáčknutí tlačítka kompletní matici odstraníme a opět vytvoříme. Při menší velikosti matici se to nejevilo jako problémové a nedocházelo ke zpomalování.

Pokud bylo stisknuto správné tlačítko, zvýší se následně počet úspěšných pokusů. Na konci funkce dochází ke kontrole pro ukončení hry. Pokud je počet provedených pokusů menší, než počet maximálních pokusů („*totalAttempts < MAX_ATTEMPTS*“), spustí se funkce od začátku a pokračuje se ve hře. Ve všech ostatních případech se spustí funkce *endGame* a dojde k ukončení hry.

```

private fun endGame() {
    val accuracy = if (successfulAttempts == 0 && totalAttempts >= MAX_ATTEMPTS)
        0.00 else (successfulAttempts.toDouble() / totalAttempts) * 100
    val accuracyTextView: TextView = findViewById(R.id.accuracyTextView)
    accuracyTextView.text = String.format("Přesnost: %.2f %%", accuracy)

    val endButton: Button = findViewById(R.id.endButton)
    endButton.text = "Zpět na menu"
    endButton.setOnClickListener { it: View!
        returnToMenu()
    }
    // Ukončí všechna tlačítka
    disableAllButtons()

    // Odstranit matici tlačítek
    removeMatrix()
}

```

Obrázek 15: *GameActivity* – *endGame*
vlastní zdroj

Funkce *endGame* je volána po překročení pokusů a poskytuje uživateli zpětnou vazbu na jeho výkon ve hře a umožňuje se přes tlačítko vrátit zpět do menu.

Nejprve je ve funkci proveden výpočet přesnosti uživatele. Pokud nebyl proveden ani jeden úspěšný pokus a zároveň byl proveden alespoň jeden pokus, přesnost je nastavena na 0,00 %. Pro ostatní případy je proveden výpočet vydělením úspěšných pokusů celkovým počtem pokusů krát 100, čímž dostaneme procentuální úspěšnost. Bylo by možné použít celá čísla, ale pro speciální případy a větší preciznost je počet desetinných míst nastaven na dvě.

Přesnost je pomocí *TextView* zobrazena jako string v textovém poli *accuracyTextView*

Tlačítku *endButton* pro ukončení hry je změněn text na „Zpět na menu“ a je mu aktualizován posluchač, aby při stisknutí zavolal metodu *returnToMenu*. Pro jasnost doteď tlačítko volalo metodu *endGame*.

Zároveň je s funkcí provedeno ukončení všech tlačítek a odstranění matice, aby se uvolnila plocha pro zobrazení výsledků.

```

private fun returnToMenu() {
    val intent = Intent(packageContext: this, MainActivity::class.java)
    startActivity(intent)
}

private fun disableAllButtons() {
    for (row in buttons.indices) {
        for (col in buttons[row].indices) {
            buttons[row][col].isEnabled = false
        }
    }
}

private fun removeMatrix() {
    val GridLayout: GridLayout = findViewById(R.id.gridLayout)
    GridLayout.removeAllViews()
}

```

Obrázek 16: *GameActivity* – *returnToMenu*
vlastní zdroj

Poslední částí celého kódu jsou tři jednoduché funkce.

Funkce *returnToMenu* je volána při stisknutí tlačítka pro ukončení hry a jejím úkolem je přesměrovat uživatele zpět na menu. Vytváří novou instanci *Intent* pro přechod na hlavní aktivitu *MainActivity* a zavolá metodu *startActivity*, která intent spustí.

Funkce *disableAllButtons* je volána při ukončení hry a deaktivují se s ní všechna tlačítka v matici. Pomocí vnořené smyčky projde kompletně všechna tlačítka matice a vypnou se nastavením vlastnosti *isEnabled* na *false*.

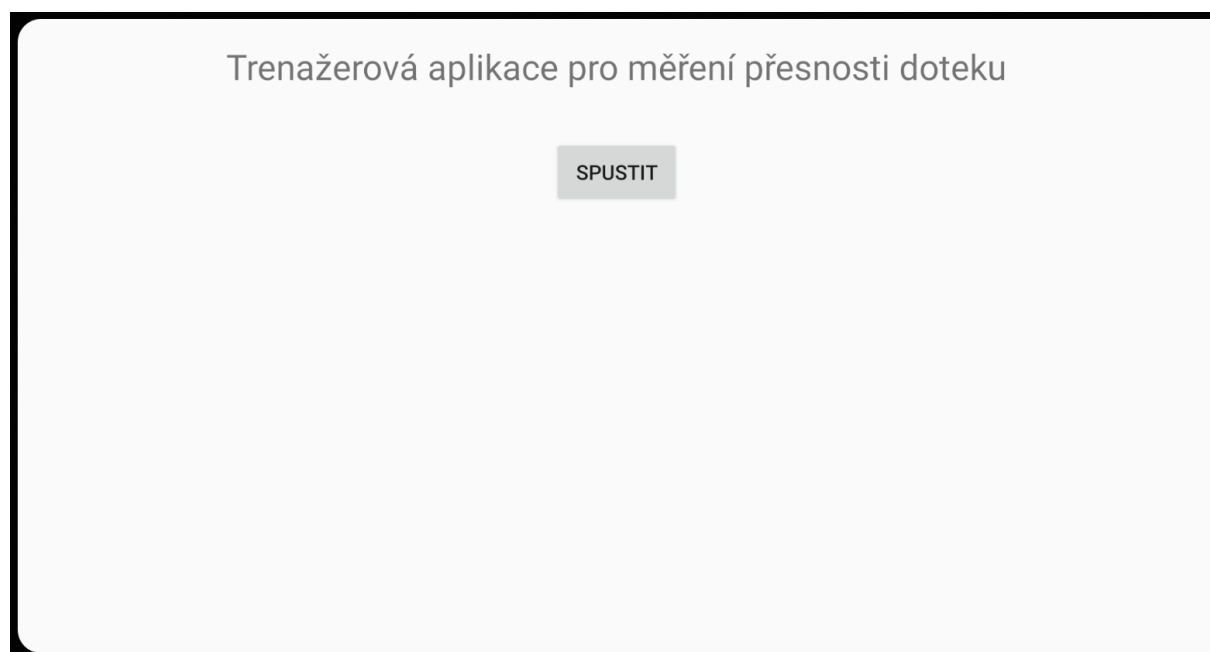
Poslední je funkce *removeMatrix*, která je také volána na konci hry a v této podobě kódu i při přetváření matice, a slouží k úplnému odstranění tlačítek matice. Najde si ID matice *gridLayout* a metodou *removeAllViews* odstraní všechny její prvky.

4 Testování

Součástí vývojového procesu každé aplikace je průběžné testování, při kterém ověřujeme správnou funkcionalitu, výkon a uživatelskou přívětivost aplikace. Naším cílem je při testování zajistit, že aplikace splňuje stanovené požadavky, očekávání a v našem případě i zadání.

Při vývoji aplikace bylo provedeno nespočet tzv. ručních testování na Androidovském emulátoru poskytovaném s Android Studiem, kde byla verze a rozměry zařízení vybrány co nejbližší skutečným parametrům tabletu gyroskopického simulátoru. Následně bylo provedeno testování přímo na tabletu, který byl propůjčen Centrem tělesné výchovy a sportu.

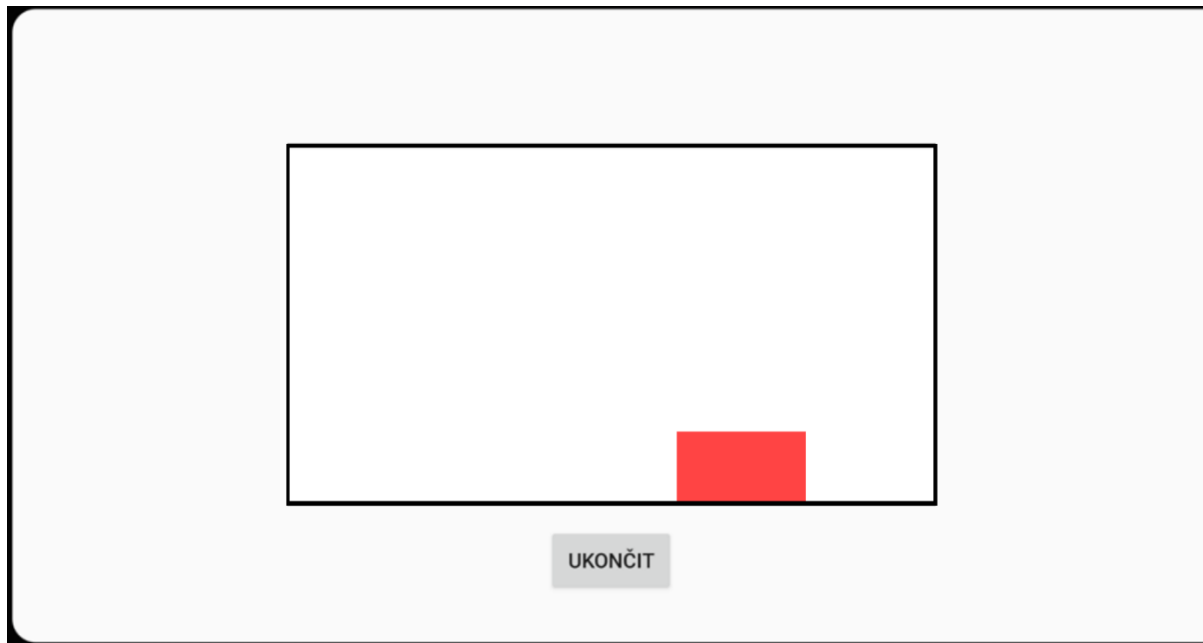
Praktické řešení všech problémů bude spolu s ostatními změnami rozvedeno v následující kapitole.



*Obrázek 17: Testování – MainActivity
vlastní zdroj*

První verze hlavní obrazovky je velmi jednoduchá a neobsahuje proto v podstatě žádné prvky pro testování, s výjimkou tlačítka pro spuštění, které funguje správně. Její hlavní a víceméně jedinou úlohou je přesměrovat uživatele na samotnou hru. Finální

verze hlavní obrazovky se bude odvíjet především od rozšiřování samotné herní části aplikace.

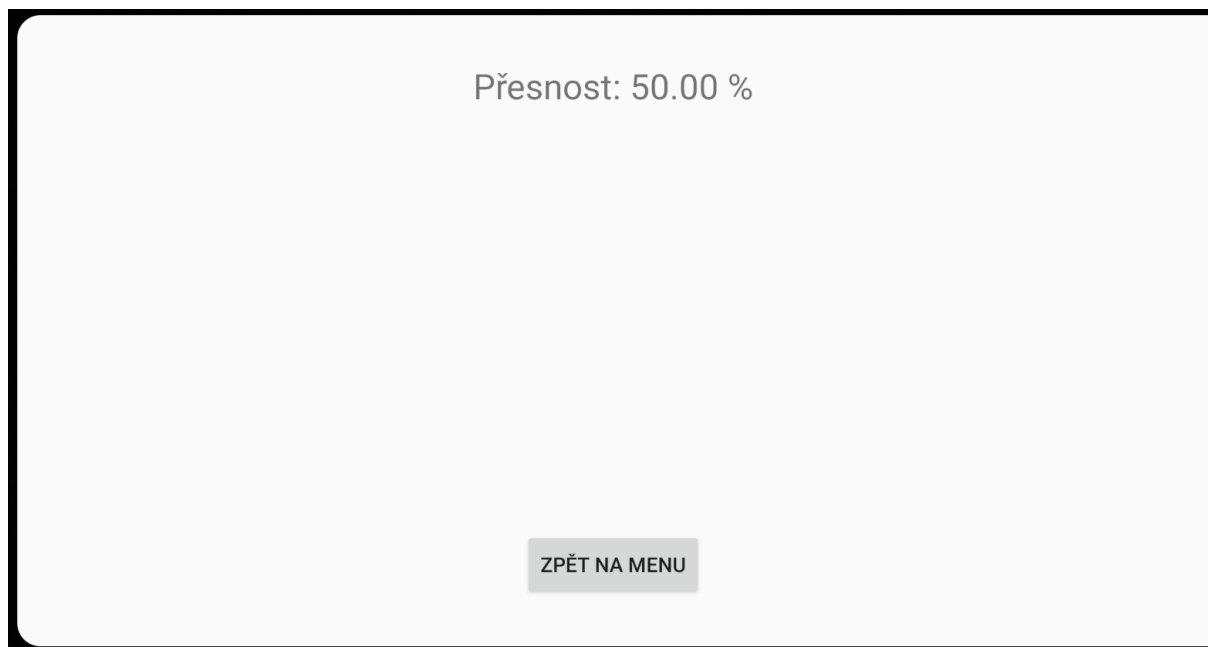


*Obrázek 18: Testování – GameActivity
vlastní zdroj*

Po stisknutí tlačítka pro spuštění dojde k odstartování samotné hry. Pro lepší čitelnost je matice tlačítek dodatečně černě zvýrazněna. Při vytváření herní matice se pro jednoduchost počítalo s malou maticí, aby byly všechny prvky aplikace snadno viditelné a nic se nepřekrývalo. Proto jsme při testování vyzkoušeli několik velikostí matice a došlo se k závěru, že nejlepší forma bude roztáhnutí na celý displej bez ohledu na počet tlačítek, čímž se dosáhne nejefektivnějšího využití herního prostoru. V závislosti na tomto závěru bude potřeba odstranit tlačítko pro ukončení pro odstranění rušících elementů. Ukončení hry může být nadále prováděno přes spodní lištu tabletu, takže nakonec nedojde k omezení možností.

Při otestování aplikace s různými velikostmi matice vyvstal problém při větších rozměrech, který nebyl ze začátku patrný. Neohrabané řešení dříve zmíněného problému s tlačítky matice vede při každém zmáčknutí aktivovaného tlačítka ke zpomalení aplikace, která musí pokaždé vytvořit novou matici. Celý systém generování tlačítek proto projde změnami.

Dalším dodatkem pro herní část bude možnost zvolení několika herních obtížností, které se budou odvíjet od počtu tlačítek v matici. Podle toho dojde i ke změně v hlavní nabídce, ve které se jedno tlačítko pro spuštění změny v několik tlačítek s nabídkou obtížností.



*Obrázek 19: Testování - GameActivity 2
vlastní zdroj*

Poslední částí aplikace je obrazovka se zobrazením přesnosti. Výpočet přesnosti a taktéž tlačítko pro návrat do menu fungují správně, a proto není třeba po technické stránce žádných oprav. Změny budou provedeny převážně pouze po estetické stránce, jako je velikost písma a tlačítka. Jediným vylepšením, které vyplní prázdný prostor obrazovky, a navíc rozšíří schopnosti měření aplikace bude přidání časovače, který změří časy mezi jednotlivými stisky uživatele, a nakonec zobrazí jeho průměrný čas mezi nimi.

5 Tvorba aplikace 2

Poslední kapitola práce se bude věnovat finální formě aplikace, jejímu porovnání s verzí první a řešení problémů zjištěných v předešlé kapitole. Většina změn vycházela z postupného vývoje a testování aplikace a jejich velká část tedy nebyla popsána v předešlé kapitole.

Od první verze byla aplikace rozšířena o několik obtížností, které jsou odlišné velikostí matice. Dále byla přidána možnost volného zadání počtu pokusů. Celkový vzhled herní části aplikace (rozšíření matice na celou obrazovku a automatické nastavení velikosti tlačítek) a vzhled ostatních aktivit vycházející z přidanych rozšíření.

Největší změnou ve finální verzi aplikace je možnost v menu „zaregistrování“ uživatele pod přezdívkou, pod kterou se následně ukládají jeho výsledky do textového souboru. Pro funkčnost tohoto vylepšení byla aplikace rozšířena o jednu aktivitu a tři XML soubory s jejich layoutem.

5.1 activity_main.xml 2

Největší změnou je v layoutu menu nahrazení jednoho tlačítka pro spuštění třemi tlačítky pro různé obtížnosti vnořenými do LinearLayout, který je řadí vedle sebe.

Dále byly přidány dva formuláře EditText, které slouží pro zadání přezdívky uživatele, používané dále pro zapsání výsledku pod zadaným jménem, a pro volné zadání počtu pokusů ve hře v číselné hodnotě.

Posledním důležitým přídatkem je nové tlačítko pro zobrazení žebříčku výsledků, který uživatele odkáže do nové aktivity *LeaderboardActivity*.

Na spodní část obrazovky bylo přidáno upozornění uživateli pro vypnutí automatického otáčení obrazovky, které by v pohybujícím se simulátoru mohlo působit komplikace při používání aplikace.

5.2 activity_game.xml 2

Do layoutu *activity_game* byl přidán element pro spojnicový graf, který se bude zobrazovat na konci hry.

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/reactionTimeChart"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:layout_above="@id/endButton"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:visibility="gone"/>
<!--Nastaven na gone, jinak v průběhu hlásí absenci dat-->
```

Obrázek 20: *activity_game.xml 2*
vlastní zdroj

Mimo to byly změněny velikosti textu a odstraněno tlačítko pro ukončení hry, které se nyní zobrazuje pouze na konci.

5.3 MainActivity.kt 2

V aktivitě pro menu byly provedeny změny spojené s přidáním elementy.

K *MainActivity* byla přidána knihovna *android.widget.EditText*, která aplikaci přidává funkce pro práci s textovými poli (*EditText*). Tím se umožňuje uživatelům zadávat textové vstupy, kterými jsou v tomto případě uživatelské jméno a počet pokusů.

```

class MainActivity : AppCompatActivity() {
    @robertjuran *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_FULLSCREEN
        supportActionBar?.hide()
        setContentView(R.layout.activity_main)

        // Tlačítka pro start obtížnosti
        val startButtonEasy: Button = findViewById(R.id.startButtonEasy)
        startButtonEasy.setOnClickListener { it: View!
            startGame( matrixSize: 3)
        }

        val startButtonMedium: Button = findViewById(R.id.startButtonMedium)
        startButtonMedium.setOnClickListener { it: View!
            startGame( matrixSize: 10)
        }

        val startButtonHard: Button = findViewById(R.id.startButtonHard)
        startButtonHard.setOnClickListener { it: View!
            startGame( matrixSize: 20)
        }
        val showLeaderboardButton: Button = findViewById(R.id.showLeaderboardButton)
        showLeaderboardButton.setOnClickListener { it: View!
            openLeaderboard()
        }
    }
}

```

Obrázek 21: MainActivity 2 – onCreate
vlastní zdroj

Místo jednoduchého spuštění hry pomocí *startGame* nyní máme tři tlačítka s vlastními posluchači, které předávají jednu ze tří hodnot *matrixSize* odpovídající vybrané obtížnosti. Tlačítko *startButtonEasy* předává hodnotu 3, *startButtonMedium* hodnotu 10 a *startButtonHard* hodnotu 20. Navíc byl přidán posluchač pro tlačítko spouštějící žebříček výsledků.

```

private fun startGame(matrixSize: Int) {
    val intent = Intent( packageContext: this, GameActivity::class.java)
    intent.putExtra( name: "MATRIX_SIZE", matrixSize)

    val nameEditText: EditText = findViewById(R.id.userNameEditText)
    val userName = nameEditText.text.toString()
    // Předej jméno uživatele do GameActivity
    intent.putExtra( name: "USER_NAME", userName)

    // Získání počtu pokusů z EditText
    val pokusyEditText: EditText = findViewById(R.id.pokusyEditText)
    val početPokusu = pokusyEditText.text.toString().toIntOrNull() ?: GameActivity.DEFAULT_MAX_ATTEMPTS
    intent.putExtra( name: "MAX_ATTEMPTS", početPokusu)

    startActivity(intent)
}

```

Obrázek 22: MainActivity 2 – startGame
vlastní zdroj

Funkce *startGame* pro spuštění hry byla značně obohacena o několik parametrů předávaných do *GameActivity*. Těmi jsou již zmíněná velikost matice *MATRIX_SIZE*, *USER_NAME* s uloženou přezdívkou uživatele a *MAX_ATTEMPTS* s číselnou hodnotou počtu pokusů.

```

private fun openLeaderboard() {
    val intent = Intent( packageContext: this, LeaderboardActivity::class.java)
    startActivity(intent)
}

```

Obrázek 23: MainActivity 2 – openLeaderboard
vlastní zdroj

Posledním dodatkem je funkce, která při aktivování příslušného posluchače spouští aktivitu pro zobrazení žebříčku.

5.4 `GameActivity.kt` 2

V současné verzi *GameActivity* bylo přidáno několik knihoven pro rozšíření funkcí aplikace:

1. ***android.content.Context***: Obecně slouží k poskytování přístupu k systémovým informacím a prostředkům aplikace. V této třídě poskytuje přístup k zdrojům, jako jsou rozměry displeje, umožňuje spouštění nových instancí *Intent* a je také využita v metodě pro ukládání výsledků do souboru.
2. ***android.graphics.Color***: Použita k manipulaci s barvami, konkrétně k nastavení barvy pozadí tlačítek.
3. ***android.os.SystemClock***: Knihovna poskytující metody pro práci s časem a hodinami. Umožňuje nám měření času mezi kliknutími, z kterého je následně uživateli vypočítána jeho průměrná reakční rychlost.
4. ***java.io.****: Zahrnuje několik knihoven (hvězdička značí všechny knihovny pod *java.io*) s nástroji pro práci se vstupem a výstupem v jazyce Java (Kotlin). Opět nám pomáhá při práci se soubory a zapisování do nich.
5. ***com.github.mikephil.charting***: Série několika tříd z knihovny *MPAndroidChart*, které poskytují komponenty pro kreslení grafů.

Dále budou uvedeny změny a vylepšení v poslední verzi *GameActivity*.

Dříve definovaná proměnná *MATRIX_SIZE* na začátku kódu je nyní posílána z *MainActivity*. Proměnné *litButtonRow* a *litButtonCol* pro pozdější vybrání umístění rozsvíceného tlačítka byly nahrazeny jedinou proměnnou *litButton* k uchování reference na aktuálně rozsvícené tlačítko. Přidány byly hodnoty *maxAttempts* a *userName* pro uchování hodnot pokusů a uživatelského jména a hodnoty *startTime*, *totalTime* a *clickCount* pro výpočet reakčního času uživatele. Poslední byla přidána proměnná *clickTimes* pro ukládání jednotlivých časů mezi kliknutími pro vykreslení grafu.

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // Skrytí "status baru"
    window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_FULLSCREEN
    // Skrytí "action baru"
    supportActionBar?.hide()
    setContentView(R.layout.activity_game)

    userName = intent.getStringExtra( name: "USER_NAME") ?: "Unknown"

    val matrixSize = intent.getIntExtra( name: "MATRIX_SIZE", defaultValue: 3)
    maxAttempts = intent.getIntExtra( name: "MAX_ATTEMPTS", DEFAULT_MAX_ATTEMPTS)

    createMatrix(matrixSize)

    val endButton: Button = findViewById(R.id.endButton)
    endButton.setOnClickListener { it: View!
        | endGame()
    }

    // Inicializace startTime při spuštění aktivitu
    startTime = SystemClock.elapsedRealtime()

    lightRandomButton()
}

```

Obrázek 24: GameActivity 2 – onCreate
vlastní zdroj

Metoda *onCreate* získává z *MainActivity* hodnoty *USER_NAME*, *MATRIX_SIZE* a *MAX_ATTEMPTS*, které si ukládá do dříve nadefinovaných proměnných. Metoda pro vytvoření matice tentokrát přijímá hodnotu *matrixSize*. Nakonec proměnná *startTime* získává hodnotu času pro pozdější výpočet.

```

// Získáme velikost displeje v pixelech
val screenWidth = resources.displayMetrics.widthPixels
val screenHeight = resources.displayMetrics.heightPixels

// Použijeme velikost displeje na velikost tlačítek (málo = velký, hodně = malý)
// - 40 jako rezerva
val buttonWidth = ((screenWidth - 40) / matrixSize )
val buttonHeight = ((screenHeight - 40) / matrixSize )

// Vytvoření tlačítek
buttons = Array(matrixSize) { row ->
    Array(matrixSize) { col ->
        MyButton( context: this, row, col).apply { this: MyButton
            layoutParams = GridLayout.LayoutParams().apply { this: GridLayout.LayoutParams
                width = buttonWidth
                height = buttonHeight
                rowSpec = GridLayout.spec(row)
                columnSpec = GridLayout.spec(col)
            }
            setOnClickListener { it: View!
                onClick( button: this)
            }
            GridLayout.addView( child: this)
        }
    }
}
}

```

Obrázek 25: *GameActivity 2 – buttons*
vlastní zdroj

V metodě *createMatrix* nově získáváme od systému rozměry displeje a z nich vypočítáváme rozměry tlačítek, aby nehledě na jejich počet správně vyplnily obrazovku. Pro rezervu je od výšky a šířky displeje odečítáno 40 pixelů.

Vytvoření tlačítek bylo lehce zjednodušeno a posluchač je nyní nastaven s odkazem na aktuální tlačítko (*this*).


```

private fun lightRandomButton() {
    val random = Random.Default
    val row = random.nextInt(buttons.size)
    val col = random.nextInt(buttons.size)

    litButton = buttons[row][col]
    litButton?.lightUp()
}

```

Obrázek 26: GameActivity 2 – lightRandomButton
vlastní zdroj

Metoda pro rozsvícení už nenastavuje sama barvu tlačítka, ale používá metodu *lightUp*, která je součástí třídy *MyButton*.

```

private fun onButtonClick(button: MyButton) {
    totalAttempts++

    if (button != litButton) {
        if (totalAttempts >= maxAttempts) {
            endGame()
        }
        return
    }

    successfulAttempts++
    button.turnOff()

    // Získání času mezi kliknutími a aktualizace celkového času a počtu kliknutí
    val currentTime = SystemClock.elapsedRealtime()
    if (clickCount > 0) {
        val reactionTime = currentTime - startTime
        totalTime += reactionTime
        clickTimes.add(reactionTime)
    }
    clickCount++
    startTime = currentTime

    if (totalAttempts < maxAttempts) {
        lightRandomButton()
    } else {
        endGame()
    }
}

```

Obrázek 27: GameActivity 2 – onButtonClick
vlastní zdroj

Metoda *onButtonClick* byla podstatně osekána a řeší problém se zpomalováním aplikace při větších rozměrech matice. Opakované mazání a přetváření matice proto bylo úplně odstraněno a nahrazeno metodou pro vypnutí jednoho tlačítka, matice tak po celou dobu od jejího vytvoření zůstává. Zároveň byla také upravena, aby přidávala časy mezi kliknutími do *clickTimes*. Vstupním parametrem pro metodu je namísto řádku a sloupce instance třídy *MyButton*.

```
private fun showReactionTimeChart() {  
    val chart: LineChart = findViewById(R.id.reactionTimeChart)  
    val entries = clickTimes.mapIndexed {index, time -> Entry(index.toFloat(), time.toFloat())}  
  
    val dataSet = LineDataSet(entries, label: "Reaction Times")  
    val lineData = LineData(dataSet)  
  
    chart.data = lineData  
    chart.invalidate() // Refresh grafu  
}
```

Obrázek 28: *GameActivity 2 – showReactionTimeChart*
vlastní zdroj

Pro vykreslování grafu byla přidána metoda *showReactionTimeChart*, která používá nové importy *MPAndroidChart*. Odkazuje se na element grafu v XML rozložení a vytváří seznam vstupů s hodnotami osy. Po vytvoření objektu s daty pro linkový graf jsou mu předávány hodnoty pro vykreslení, a nakonec je graf obnoven, aby se zobrazila nová data.

```

private fun endGame() {
    removeMatrix()
    val accuracy = calculateAccuracy()
    val accuracyTextView: TextView = findViewById(R.id.accuracyTextView)
    accuracyTextView.text = String.format("Přesnost: %.2f %%", accuracy)

    // Získání hodnoty matrixSize
    val matrixSize = intent.getIntExtra("MATRIX_SIZE", defaultValue: 3)

    // Výpočet průměrného času kliknutí a zobrazení na konci hry
    val averageTimePerClick = if (clickCount > 0) totalTime.toDouble() / clickCount else 0.0
    val averageTimeTextView: TextView = findViewById(R.id.averageTimeTextView)
    averageTimeTextView.text = String.format("Průměrný čas kliknutí: %.2f ms", averageTimePerClick)

    // Uložení výsledku do souboru
    val userResult = DataManager.UserResult(userName, matrixSize, totalAttempts, successfulAttempts, averageTimePerClick.toLong())
    DataManager.saveResultToFile(userResult, context: this) // Předáváme aktuální kontext

    // Zobrazí graf reakčních časů
    val reactionChart = findViewById<LineChart>(R.id.reactionTimeChart)
    reactionChart.visibility = View.VISIBLE
    showReactionTimeChart()
    // Nastavení popisku grafu
    reactionChart.getDescription().setText("Reakční čas/Pokusy");

    val endButton: Button = findViewById(R.id.endButton)
    endButton.text = "Zpět na menu"
    endButton.visibility = View.VISIBLE
    endButton.setOnClickListener { it: View!
        returnToMenu()
    }
}

```

Obrázek 29: *GameActivity 2 – endGame*
vlastní zdroj

Metoda *endGame* byla naopak viditelně rozšířena a obohacena o několik funkcí. Jediná instance pro mazání matice *removeMatrix* je zde umístěna na začátku metody. Výpočet přesnosti se už neprovádí v této metodě, ale byl přemístěn do vlastní metody *calculateAccuracy*, která je obsahově v podstatě totožná, takže nebude uvedena. Nově vypočítává metoda průměrný čas kliknutí *averageTimePerClick* a pro zobrazení jej převádí do textové formy. Největším rozšířením je ukládání výsledků uživatele a jejich zapisování pomocí třídy *DataManager*, a zobrazování grafu, který je nastaven na *visible* a je mu přidán popisek.

```

class MyButton(context: Context, val row: Int, val col: Int) : androidx.appcompat.widget.AppCompatButton(context) {

    @robertjuran
    init {
        setBackgroundColor(Color.WHITE)
    }

    @robertjuran
    fun lightUp() {
        setBackgroundColor(Color.RED)
    }

    @robertjuran
    fun turnOff() {
        setBackgroundColor(Color.WHITE)
    }
}

```

Obrázek 30: GameActivity 2 – MyButton
vlastní zdroj

Nově vytvořená třída *MyButton* slouží pro implementaci tlačítek a dědí ze třídy *AppCompatButton*, která obsahuje vlastnosti pro standardní tlačítko. Dále má konstruktor s parametry *Context*, *row* (řádek) a *col* (sloupec) tlačítka. Tato třída obsahuje metody pro inicializaci (*init*) s výchozí bílou barvou, rozsvícení (*lightUp*) s červenou barvou a zhasnutí (*turnOff*) tlačítka zpět na bílou.

```

object DataManager {
    private const val FILENAME = "user_results.txt"

    // Třída UserResult pro ukládání výsledků
    @robertjuran
    data class UserResult(
        val userName: String,
        val matrixSize: Int,
        val totalAttempts: Int,
        val successfulAttempts: Int,
        val averageSpeed: Long
    )

    @robertjuran
    fun saveResultToFile(result: UserResult, context: Context) {
        val line = "${result.userName},${result.matrixSize},${result.totalAttempts},${result.successfulAttempts},${result.averageSpeed}"
        try {
            context.openFileOutput(FILENAME, Context.MODE_APPEND).use { it: FileOutputStream!
                it.write((line + "\n").toByteArray())
            }
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }
}

```

Obrázek 31: GameActivity 2 – DataManager
vlastní zdroj

Poslední novou částí je singleton objekt *DataManager* sloužící k operacím se správou dat, konkrétně načítání a ukládání výsledků.

Konstanta *FILENAME* určuje název souboru, do kterého jsou data ukládána. Třída *UserResult* obsahuje reprezentace výsledků s atributy *userName*, *matrixSize*, atd.

Metoda *saveResultsToFile* otevírá soubor pro zápis a vrátí *OutputStream*, do kterého lze poté zapisovat. Data jsou v textovém formátu a oddělována čárkou. Pro vzniklé výjimky je použit blok try-catch, který vypíše trasovací informace výjimky.

5.5 activity_leaderboard.xml a item_leaderboard_header.xml

Při postupném rozšiřování aplikace vznikla potřeba přidání nové třídy s aktivitou pro obsluhování žebříčku s výsledky uživatelů. Stejně tak byly přidány layouty pro vzhled této aktivity, kterými jsou *activity_leaderboard* se samotným rozvržením a *item_leaderboard_header* sloužící pro upravení vzhledu tabulky s výsledky.

```
<!-- Drop-down pro výběr řazení -->
<Spinner
    android:id="@+id/sortBySpinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentTop="true"
    android:layout_marginEnd="16dp"
    android:layout_marginTop="16dp" />
```

Obrázek 32: *activity_leaderboard* – Spinner
vlastní zdroj

Layout obsahuje rozložení pro tzv. spinner, který definuje drop-down nabídku pro změnu řazení tabulky.

```

<LinearLayout
    android:id="@+id/buttonContainer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp">

    <Button
        android:id="@+id/easyButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Lehká" />

    <Button
        android:id="@+id/mediumButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Střední"
        android:layout_marginStart="8dp" />

    <Button
        android:id="@+id/hardButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Těžká"
        android:layout_marginStart="8dp" />

</LinearLayout>

```

Obrázek 33: *activity_leaderboard* – Buttons
vlastní zdroj

Problém s razením výsledků byla skutečnost, že od sebe nebyly odlišeny různé obtížnosti, takže se výsledky her „Jednoduchá“, „Střední“ i „Těžká“ řadily mezi sebe nehledě na použitou velikost matice. Tento problém byl vyřešen přidáním hodnoty velikosti matice do výsledků a vytvoření tlačítek, které zobrazí výsledky pouze z vybrané obtížnosti. Tlačítka jsou vložena do `LinearLayout`, takže jsou seřazena vedle sebe.

```

<!-- Tlačítko pro vymazání obsahu souboru -->
<Button
    android:id="@+id/clearButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Vymazat výsledky"
    android:layout_below="@+id/recyclerView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"/>

<!-- RecyclerView pro zobrazení žebříčku -->
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/buttonContainer"
    android:paddingTop="8dp"
    android:paddingBottom="8dp"
    android:clipToPadding="false"
    android:scrollbars="vertical" />

```

Obrázek 34: *activity_leaderboard* – *RecyclerView*
vlastní zdroj

Spodní část obrazovky obsahuje tlačítko pro vymazání výsledků, které po stisknutí otevírá textový soubor a maže jeho obsah.

První a nejdůležitější částí celého layoutu je tzv. *recyclerView*, který je v podstatě tabulkou, do které se vypisují výsledky z textového souboru podle vytvořené logiky. Prvek *recyclerView* automaticky obsahuje funkci scrollování, takže nehrozí zaplnění tabulky při větším počtu řádků.

Soubor *item_leaderboard_header* obsahuje pouze čtyři prvky *TextView* v *LinearLayout* s texty „Jméno“, „Počet Pokusů“, „Úspěšné Pokusy“ a „Průměrný Čas“, a slouží jako záhlaví pro celou tabulku.

5.6 LeaderboardActivity.kt

Úplně poslední částí aplikace je samotná aktivita pro obsluhování žebříčku, jejíž layout byl popsán výše. Podíváme se na několik nových knihoven, které v ní byly použity:

1. ***android.view.LayoutInflater***: Používá se v Androidu k naplnění XML layoutů do odpovídajících objektů v běžící aplikaci. Používáme ho v relaci s *activity_leaderboard*, kde v metodě *OnCreate* inicializujeme různé prvky uživatelského rozhraní.
2. ***android.view.ViewGroup***: Základní třída pro vytváření skupin prvků uživatelského rozhraní. V naší aktivitě je zodpovědná za zobrazení uživatelů v žebříčku.
3. ***androidx.recyclerview.widget.RecyclerView***: Nástroj pro zobrazování většího objemu dat v seznamu, mřížce apod. Používáme při zobrazování seznamu uživatelů.
4. ***androidx.recyclerview.widget.LinearLayoutManager***: Opět nástroj pro zobrazování dat v nějakém seznamu, konkrétně u nás je jeho instance správcem rozložení pro *RecyclerView*.
5. ***android.widget.Toast***: Umožňuje pro uživatele zobrazit krátkou zprávu jako pop-up. Zobrazuje nám zprávu při úspěšném vymazání výsledků ze souboru.
6. ***android.widget.AdapterView***: Adaptér pro převádění dat z datového zdroje na zobrazení položek v rozhraní. Používáme jej pro naplnění dat do spinneru pro výběr způsobu řazení žebříčku.
7. ***android.widget.AdapterView***: Poskytuje obecné metody pro práci s daty a jejich zobrazování v rozhraní. Opět používáme při naplňování tabulky v *RecyclerView*.
8. ***android.widget.Spinner***: Umožňuje použít drop-down menu (spinner), což je prvek uživatelského rozhraní, který dovoluje uživatelům vybírat hodnoty z rozbalovacího seznamu.

Aktivita žebříčku neplní pouze funkci zobrazování dat; musí mimo to být schopná číst data z otevřeného souboru, vložit je správně do tabulky, seřadit je podle pokynů uživatele a rozlišovat různé obtížnosti, podle kterých buď data do tabulky vloží, nebo je vynechá. Navíc aktivita obsahuje funkci kompletního smazání textového obsahu souboru.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_leaderboard)

    recyclerView = findViewById(R.id.recyclerView)
    layoutManager = LinearLayoutManager( context: this)
    recyclerView.layoutManager = layoutManager

    // Inicializace spinneru pro filtraci podle výsledku
    sortBySpinner = findViewById(R.id.sortBySpinner)

    // Nastavení posluchače pro spinner
    setupSpinner()
}
```

Obrázek 35: *LeaderboardActivity* – *onCreate*
vlastní zdroj

Na začátku třídy inicializujeme několik proměnných pro pozdější použití. Proměnná *recyclerView* slouží pro zobrazení kontejneru (seznamu) s výsledky. Správce rozložení *layoutManager* určuje, jak budou položky v tabulce uspořádány. Data o uživatelích uspořádává *leaderboardAdapter*, který je připravuje a podle pokynů zobrazuje. Poslední lateinit variable je *sortBySpinner*, který poskytuje funkčnost widgetu se spinnerem. Informace o velikosti matice, které se později zapisují do textového souboru jsou později ukládány do proměnné *currentMatrixSize*.

V metodě *onCreate* na začátku klasicky provádíme základní nastavení aktivity a nastavujeme layout podle souboru *activity_leaderboard*. Dále proběhne inicializace *RecyclerView* pomocí ID, vytváříme nový *LinearLayoutManager* a přiřazujeme layout manager k *RecyclerView*. Následně najdeme podle ID spinner a přiřadíme ho k příslušné proměnné a voláme metodu *setupSpinner* pro nastavení posluchačů.

```

val easyButton: Button = findViewById(R.id.easyButton)
val mediumButton: Button = findViewById(R.id.mediumButton)
val hardButton: Button = findViewById(R.id.hardButton)
val clearButton: Button = findViewById(R.id.clearButton)

// Tlačítka pro filtraci leaderboardu podle obtížnosti
easyButton.setOnClickListener { it: View!
    currentMatrixSize = 3
    loadUsersByMatrixSize( matrixSize: 3)
}

mediumButton.setOnClickListener { it: View!
    currentMatrixSize = 10
    loadUsersByMatrixSize( matrixSize: 10)
}

hardButton.setOnClickListener { it: View!
    currentMatrixSize = 20
    loadUsersByMatrixSize( matrixSize: 20)
}

// Tlačítka pro vymazání výsledků
clearButton.setOnClickListener { it: View!
    clearResults()
}

// Bez vybrání obtížnosti zobrazí všechny
loadAllUsers()
}

```

Obrázek 36: LeaderboardActivity – Buttons
vlastní zdroj

V druhé části třídy je provedeno nalezení tlačítek podle jejich ID v layoutu, jejich přiřazení proměnným a následné přiřazení posluchačů těmto tlačítkům. Tlačítka plní funkci filtrace výsledků a jejich smazání.

Poslední metoda obstará zobrazení všech výsledků, když není vybrána obtížnost.

```

// Fce pro zobrazení všech výsledků
private fun loadAllUsers() {
    val users = loadUsersFromStorage()
    leaderboardAdapter = LeaderboardAdapter(users)
    recyclerView.adapter = leaderboardAdapter
}

// Fce pro filtraci výsledků podle velikosti matice (obtížnosti)
private fun loadUsersByMatrixSize(matrixSize: Int) {
    val users = loadUsersFromStorage().filter { it.matrixSize == matrixSize }
    leaderboardAdapter = LeaderboardAdapter(users)
    recyclerView.adapter = leaderboardAdapter
}

// Fce pro vymazání obsahu .txt s výsledky
private fun clearResults() {
    val fileName = "user_results.txt"
    try {
        applicationContext.deleteFile(fileName)
        Toast.makeText( context: this, text: "Výsledky byly úspěšně smazány.", Toast.LENGTH_SHORT).show()
        loadAllUsers()
    } catch (e: Exception) {
        Toast.makeText( context: this, text: "Nastala chyba při mazání výsledků.", Toast.LENGTH_SHORT).show()
    }
}

```

Obrázek 37: LeaderboardActivity - loadUsers, clearResults
vlastní zdroj

Metoda *loadAllUsers* použitá právě po spuštění posluchačů načítá pomocí metody *loadUsersFromStorage* data výsledků, předává je do *LeaderboardAdapteru* a následně nastavuje adapter pro *RecyclerView* pro jejich zobrazení.

Další metoda *loadUsersByMatrixSize* načítá opět data s výsledky, které filtruje podle obtížnosti (velikosti matice). Dále funguje jako předešlá.

Metoda *clearResults* si otevírá soubor s výsledky a buď jej maže, přičemž se zobrazí toast zpráva „Výsledky byly úspěšně smazány“, nebo zachytí výjimku a zobrazí zpráva „Nastala chyba při mazání výsledků“.

```

private fun loadUsersFromStorage(): List<User> {
    val users = mutableListOf<User>()
    try {
        val fileName = "user_results.txt"
        applicationContext.openFileInput(fileName).bufferedReader().useLines { lines ->
            lines.forEach { line ->
                val parts = line.split( ...delimiters: ",")
                if (parts.size == 5) {
                    val userName = parts[0]
                    val matrixSize = parts[1].toInt()
                    val totalAttempts = parts[2].toInt()
                    val successfulAttempts = parts[3].toInt()
                    val averageSpeed = parts[4].toLong()
                    users.add(User(userName, matrixSize, totalAttempts, successfulAttempts, averageSpeed))
                }
            }
        }
    } catch (e: FileNotFoundException) {
        // Soubor neexistuje nebo je prázdný
    }
    return users
}

```

Obrázek 38: LeaderboardActivity – loadUsersFromStorage
vlastní zdroj

Již použitá metoda *loadUserFromStorage* má za úkol načítat data v podobě textu ze souboru a vracet je ve formě seznamu. Nejprve je vytvořen prázdný seznam *users* pro postupné zapisování dat. Poté je zbytek uzavřen do try-catch bloku, který bude postupně monitorovat proces načítání. V tomto bloku je otevřen soubor s výsledky a vrácený *InputStream* je zpracován přes třídu *BufferedReader*, která umožňuje postupné čtení po řádcích. Jednotlivé části řádku jsou vidět v kódu a rozděleny jsou čárkou.

V případě, že soubor neexistuje, vyvolá se výjimka, zachytí se do bloku catch a není provedena žádná operace. Nakonec je seznam s daty vrácen z metody.

```

class LeaderboardAdapter(private val users: List<User>) :
    RecyclerView.Adapter<LeaderboardAdapter.LeaderboardViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LeaderboardViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_leaderboard, parent, attachToRoot: false)
        return LeaderboardViewHolder(view)
    }

    override fun onBindViewHolder(holder: LeaderboardViewHolder, position: Int) {
        val user = users[position]
        holder.bind(user)
    }

    override fun getItemCount(): Int {
        return users.size
    }

    inner class LeaderboardViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        fun bind(user: User) {
            itemView.findViewById<TextView>(R.id.userNameTextView).text = user.name
            itemView.findViewById<TextView>(R.id.attemptsTextView).text = "Počet pokusů: ${user.totalAttempts}"
            itemView.findViewById<TextView>(R.id.successfulAttemptsTextView).text = "Úspěšné pokusy: ${user.successfulAttempts}"
            itemView.findViewById<TextView>(R.id.averageSpeedTextView).text = "Průměrný čas: ${user.averageSpeed} ms"
        }
    }
}

```

Obrázek 39: LeaderboardActivity – LeaderboardAdapter
vlastní zdroj

Třída *LeaderboardAdadapter* je adapter pro propojení seznamu uživatelů s RecyclerView, který jej zobrazuje.

Konstruktor přijímá seznam uživatelů, který má poté RecyclerView zobrazovat. Metoda *onCreateViewHolder* je volána, když RecyclerView potřebuje při operacích vytvořit nový ViewHolder, tedy buňku s jednou položkou ze seznamu. Metoda *onBindViewHolder* se volá při plnění nově vytvořeného ViewHolder. Přijímá vytvořený *LeaderboardViewHolder* a index položky v seznamu. Metoda *getItemCount* zjišťuje počet položek v seznamu a určuje, kolikrát bude volána metoda *onBindViewHolder*.

Vnitřní třída *LeaderboardViewHolder* obsahuje odkazy na jednotlivé položky v seznamu a metodu *bind*, která určuje obsah těchto prvků.

```

// Globální proměnná pro uchování aktuálně vybrané možnosti řazení
private var currentSortOption = SortByOption.USER_NAME

// V metodě setupSpinner() nastavení posluchače pro změnu výběru ve spinneru
private fun setupSpinner() {
    val sortByOptions = arrayOf("Jméno", "Počet Pokusů", "Úspěšné pokusy", "Průměrný čas")
    val adapter = ArrayAdapter(context = this, android.R.layout.simple_spinner_item, sortByOptions)
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
    sortBySpinner.adapter = adapter

    // Nastavení posluchače pro změnu výběru ve spinneru
    sortBySpinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
            val sortByOption = when (position) {
                0 -> SortByOption.USER_NAME
                1 -> SortByOption.TOTAL_ATTEMPTS
                2 -> SortByOption.SUCCESSFUL_ATTEMPTS
                3 -> SortByOption.AVERAGE_SPEED
                else -> SortByOption.USER_NAME
            }
            currentSortOption = sortByOption
            // Řazení žebříčku podle vybrané možnosti
            sortLeaderboard()
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {
            // Nepotřebujeme implementovat
        }
    }
}
}

```

Obrázek 40: LeaderboardActivity – setupSpinner
vlastní zdroj

Metoda *setupSpinner* slouží k inicializaci a nastavení spinneru, který slouží k řazení položek v tabulce s výsledky.

Globální proměnná *currentSortOption* uchovává aktuální vybranou možnost spinneru a její výchozí řazení je podle *USER_NAME*, neboli podle jmen uživatelů.

Samotná metoda začíná vytvořením pole řetězců s hodnotami Jméno, Počet Pokusů atd., které reprezentují možnosti řazení. *ArrayAdapter* vytváří adapter, který propojí pole se spinnerem. Vzhled jednotlivých položek spinneru je nastaven přes *setDropDownViewResource* na základní vzhled (*simple_spinner_dropdown_item*) a nakonec je kompletní adapter přiřazen.

Následuje nastavení posluchače událostí, které je v tomto případě o něco složitější, než jak tomu bylo u klasických tlačítek. Přes *sortBySpinner.onItemSelectedListener* je třeba nastavit posluchače pro každou položku ve spinneru metodou *onItemSelected*. Proměnná *sortByOption* rozlišuje, jaká z variant spinneru byla vybrána, následně ji ukládá do globální proměnné *currentSortOption* a metoda *sortLeaderboard* poté řadí žebříček podle vybrané možnosti. Metoda *onNothingSelected* v momentální verzi neplní žádnou úlohu, ale byla zanechána pro případné rozšíření a je volána, pokud není vybrána žádná z možností.

```
// Metoda pro řazení seznamu uživatelů podle aktuálního výběru ve spinneru
private fun sortLeaderboard() {
    val users = if (currentMatrixSize != 0) {
        loadUsersFromStorage().filter { it.matrixSize == currentMatrixSize }.toMutableList()
    } else {
        loadUsersFromStorage().toMutableList()
    }

    users.sortWith(compareBy<User> { it: User
        when (currentSortOption) {
            SortByOption.USER_NAME -> it.name ^compareBy
            SortByOption.TOTAL_ATTEMPTS -> -it.totalAttempts ^compareBy
            SortByOption.SUCCESSFUL_ATTEMPTS -> -it.successfulAttempts ^compareBy
            SortByOption.AVERAGE_SPEED -> it.averageSpeed ^compareBy
        }
    })

    leaderboardAdapter = LeaderboardAdapter(users)
    recyclerView.adapter = leaderboardAdapter
}
```

Obrázek 41: LeaderboardActivity – sortLeaderboard
vlastní zdroj

Metoda *sortLeaderboard* slouží v předchozí metodě na seřazení tabulky podle vybrané možnosti spinneru. Metoda nejdříve rozlišuje momentálně vybranou obtížnost podle velikosti matice a zobrazuje pouze ty výsledky. Bez vybrané varianty zobrazuje všechny. V druhé části provádí řazení dle vybrané možnosti spinneru podle nastavených kritérií; *USER_NAME* jsou řazeny abecedně, hodnoty *TOTAL_ATTEMPTS* i *SUCCESSFUL_ATTEMPTS* mají obě řazení záporné a řadí se tedy od největší

a hodnoty *AVERAGE_SPEED* mají řazení kladné, tedy budou postupně řazeny od nejmenší.

Nakonec metody je vytvořen nový adapter s aktuálně seřazenými uživateli a je nastaven pro *recyclerView*, čímž se aktualizuje zobrazení žebříčku.

6 Instalace a vyhodnocení

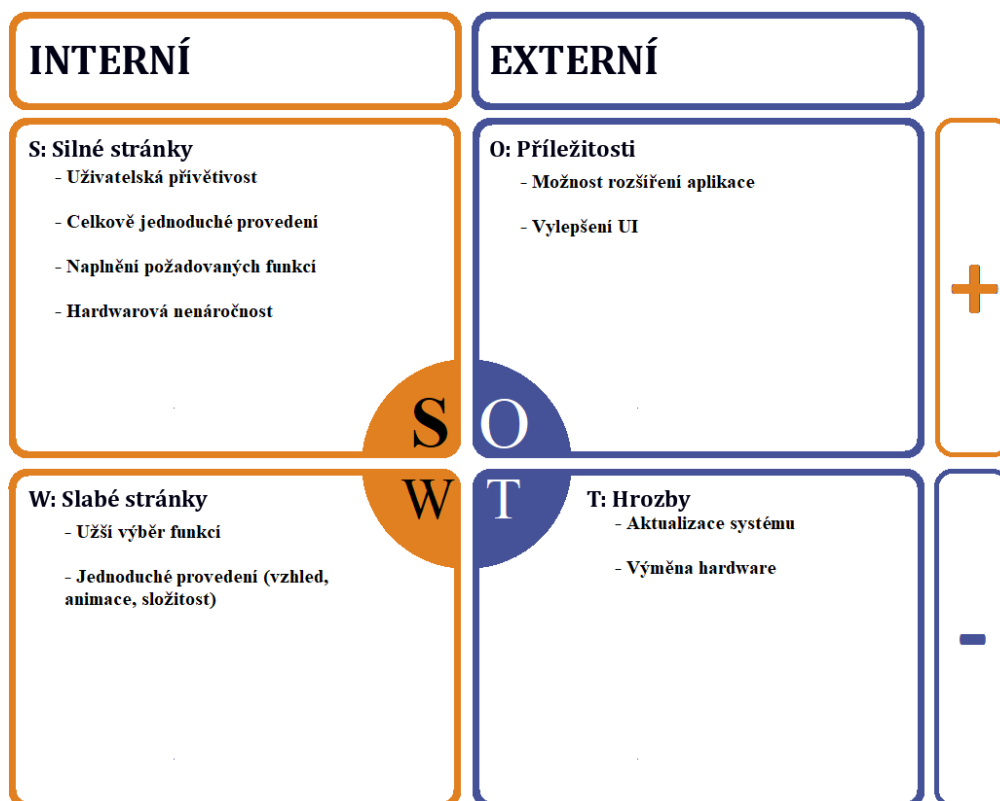
Poslední fází při vývoji této aplikace je instalace do tabletu gyroskopického simulátoru, otestování funkčnosti a vyhodnocení získaných dat. Aplikaci lze snadno otestovat i bez zasazení do simulátoru, jelikož tablet je odnímatelný a byl poskytnut k zapůjčení pro účely vývoje a testování.

Instalace na tablet byla provedena a tablet byl poskytnut Centru tělesné výchovy a sportu, kde bude využíván pro testování schopností studentů pod vlivem gyroskopického simulátoru.



Obrázek 42: Gyroskopický trenážér [12]

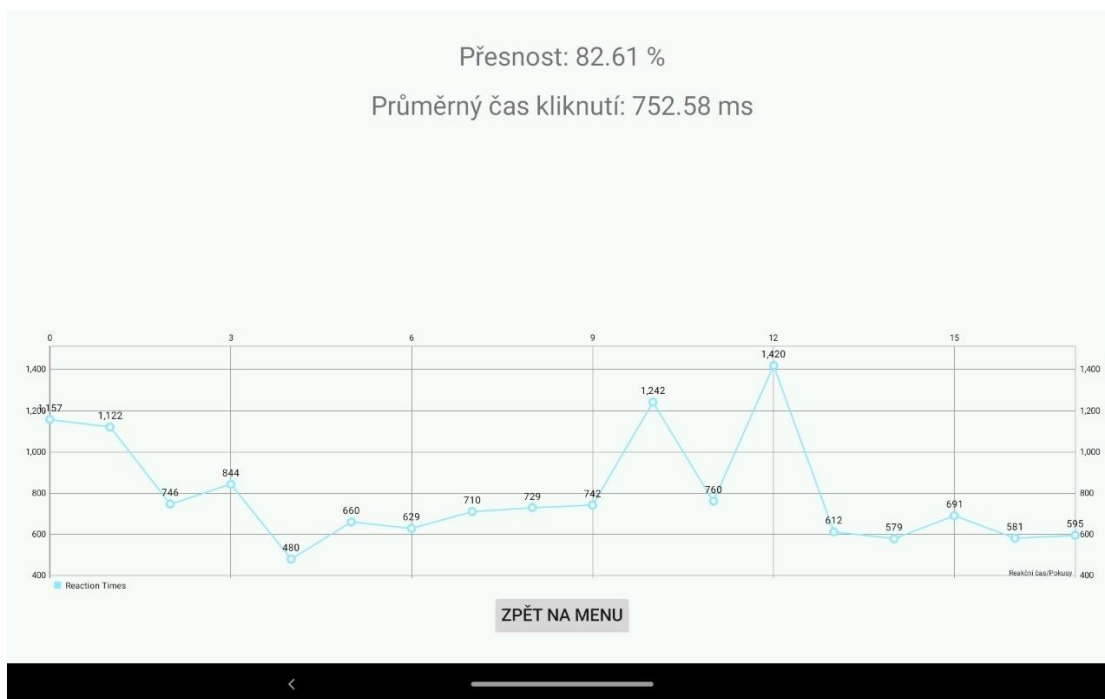
Z provedených měření v simulátoru i mimo něj lze sledovat značné zhoršení přesnosti, jak při vyšších obtížnostech, tak i při více pokusech. Díky implementaci žebříčku výsledků a grafu, který zřetelně zobrazuje rozdíly v reakčních časech a přesnosti uživatele, lze z dat provést vyhodnocení podle kritérií učitelů. V delším časovém období a po větším počtu testování by poté bylo snadné pozorovat postupné zlepšování studentů v jejich přesnosti a rychlosti reakcí při používání aplikace, k čemuž byla samozřejmě v první řadě vytvořena.



Obrázek 43: SWOT Analýza aplikace
vlastní zdroj

Aplikace je vytvořena jako doplnění nedostatků již předinstalované aplikace dodávané se simulátorem. Je navržena tak, aby byla co nejvíce uživatelsky přívětivá a jednoduchá na používání. Toho bylo podle obdržených názorů dosaženo a snadný přístup k datům umožňuje jednoduché a rychlé vyhodnocení výsledků. Podle dalších požadavků Centra tělesné výchovy a sportu lze aplikaci dále rozšířit a vylepšovat.

Celková jednoduchost je zároveň jistou nevýhodou. Omezení malého tabletu nedovoluje vytváření složitých herních ploch, které by obsahovaly například věrnou napodobeninu skutečného letadla. Současná podoba aplikace také nemá příliš mnoho možností a budoucí rozšíření by bylo rozhodně přínosné při jejím používání. Ačkoliv by se tento problém neměl vyskytnout, tak je jistá hrozba v případě aktualizací systémů, které by mohly neočekávaně znemožnit používání aplikace. V případě výměny tabletu by bylo potřeba aplikaci opět manuálně nainstalovat z instalačního balíčku přiloženého k práci.



Obrázek 44: Graf výsledků
vlastní zdroj

LEHKÁ	STŘEDNÍ	TĚŽKÁ	Jméno
Test1 E	Počet pokusů: 10	Úspěšné pokusy: 10	Průměrný čas: 447 ms
Test2 E	Počet pokusů: 20	Úspěšné pokusy: 19	Průměrný čas: 450 ms
Test3 M	Počet pokusů: 11	Úspěšné pokusy: 6	Průměrný čas: 674 ms
Test4 M	Počet pokusů: 21	Úspěšné pokusy: 15	Průměrný čas: 576 ms
Test5 H	Počet pokusů: 12	Úspěšné pokusy: 10	Průměrný čas: 712 ms
Test6 H	Počet pokusů: 23	Úspěšné pokusy: 19	Průměrný čas: 752 ms

VYMAZAT VÝSLEDKY

Obrázek 45: Žebříček výsledků
vlastní zdroj

ZÁVĚR

Cíle práce byly splněny. Aplikace, kterou jsem vyvinul, je plně funkční a splňuje požadavky určené Centrem tělesné výchovy a sportu. Téma práce bylo zvoleno s úmyslem zaměřit se především na praktickou část práce a zajistit, aby výsledek byl použitelný v praxi.

Značná část práce je věnována podrobnému rozebrání všech částí aplikace, přičemž funguje zároveň jako její dokumentace pro účely případného dalšího vývoje a jejího rozšiřování.

Analýza možností implementace aplikace mi umožnila vybrat nejvhodnější technologie a knihovny pro dosažení optimálního výsledku. Na základě průzkumu jsem navrhl efektivní řešení, které bylo následně implementováno do tabletu gyroskopického trenažéru.

Provedené vyhodnocení aplikace potvrdilo správnost zvoleného přístupu a kvalitu výsledné aplikace. V závěrečné fázi práce jsem se zaměřil na popis možného využití navrženého řešení a zmínil některé výhody i nevýhody navrženého řešení.

SEZNAM POUŽITÉ LITERATURY

- [1] VÁVRA, Jiří. & UJBÁNYAI, Miroslav. *Programujeme pro Android*. 1. vyd. Praha: Grada Publishing a.s., 2013. s. 15 – 58. ISBN 978-80-247-4863-4
- [2] PHILLIPS, Bill. & HARDY, Brian. *Android Programming: The Big Nerd Ranch Guide*. 1. vyd. USA: Big Nerd Ranch, Inc., 2013. s. 1 – 210. ISBN 978-0321804334
- [3] BELINSKI, E. (nedatováno) *Android API Levels* [online]. [cit. 2024-10-2]. Dostupné z: <https://apilevels.com>
- [4] GeeksforGeeks. (2024) *Android Architecture* [online]. [cit. 2024-20-2]. Dostupné z: <https://geeksforgeeks.org/android-architecture/>
- [5] Tutorials Point. (nedatováno) *Java Tutorial* [online]. [cit. 2024-22-2]. Dostupné z: <https://www.tutorialspoint.com/java/index.htm>
- [6] GeeksforGeeks. (2024, leden 31). *C++ Programming Language* [online]. [cit. 2024-25-2]. Dostupné z: <https://www.geeksforgeeks.org/c-plus-plus/>
- [7] W3Schools. (nedatováno) *XML Introduction* [online]. [cit. 2024-28-2]. Dostupné z: https://www.w3schools.com/xml/xml_what.asp
- [8] LUTKEVICH, B. (2022, prosinec 28). *Kotlin* [online]. [cit. 2024-5-3] Dostupné z: <https://www.techtarget.com/whatis/definition/Kotlin>
- [9] Android Developers. *Developer guides: Android Fundamentals* [online]. [cit. 2024-15-3]. Dostupné z: <https://developer.android.com/guide/components/fundamentals>
- [10] Android Developers. (nedatováno). *Android API reference* [online]. [cit. 2024-25-3] Dostupné z: <https://developer.android.com/reference>
- [11] FORGIE, A. (2019, listopad 7). *F-35 Lightning II cockpit simulator* [online]. [cit. 2024-30-3] Dostupné z: <https://kmyu.tv/news/local/cockpit-simulator-shows-off-the-partially-made-in-utah-f-35-lightning-ii-fighter-jet>
- [12] GÖGH, L. (2023). *Gyroskopický trenažér*. [cit. 2024.-20-5]. CTVS

SEZNAM PŘÍLOH

Příloha A – Zdrojový kód aplikace

Příloha A Zdrojový kód aplikace

activity_main.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context="com.example.diplomka_test.MainActivity">

    <TextView
        android:id="@+id/nadpisTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="35sp"
        android:text="Trenažerová aplikace pro měření přesnosti
doteku"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"/>

    <TextView
        android:id="@+id/obtiznostTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:text="Vyberte obtížnost:"
        android:layout_below="@+id/pokusyEditText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="64dp"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="32dp"
        android:layout_below="@id/obtiznostTextView">

        <Button
            android:id="@+id/startButtonEasy"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Jednoduchá"
            android:textSize="20sp"
            android:padding="10dp"
            android:layout_marginEnd="8dp"/>

        <Button
            android:id="@+id/startButtonMedium"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Střední"
            android:textSize="20sp"
            android:padding="10dp"
            android:layout_marginEnd="8dp"/>

        <Button
            android:id="@+id/startButtonHard"
            android:layout_width="wrap_content"
```



```

        android:layout_height="wrap_content"
        android:text="Těžká"
        android:textSize="20sp"
        android:padding="10dp"/>

</LinearLayout>

<EditText
    android:id="@+id/userNameEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Zadejte své jméno"
    android:inputType="text"
    android:textSize="24sp"
    android:layout_below="@id/nadpisTextView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="32dp"
    android:gravity="center_horizontal"/>

<EditText
    android:id="@+id/pokusyEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:hint="Zde zadejte počet pokusů (výchozí je 10)"
    android:textSize="24sp"
    android:layout_below="@+id/userNameEditText"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="32dp"
    android:gravity="center_horizontal"/>

<Button
    android:id="@+id/showLeaderboardButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/upozorneniTextView"
    android:text="Zobrazit žebříček"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="128dp"
    android:gravity="center_horizontal"/>

<TextView
    android:id="@+id/upozorneniTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ujistěte se, že je vypnuté automatické otáčení
displeje!"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="32dp"
    android:textSize="20sp"/>

</RelativeLayout>

```

activity_game.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp"
tools:context="com.example.diplomka_test.GameActivity">

    <TextView
        android:id="@+id/accuracyTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text=""
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"/>

    <GridLayout
        android:id="@+id/gridLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerInParent="true" />

    <Button
        android:id="@+id/endButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="20sp"
        android:padding="10dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp"
        android:visibility="gone"/>
    <!--Nastavíme tlačítko na "gone", aby nám nezavazelo v matici
    (po ukončení hry se zobrazí)-->

    <TextView
        android:id="@+id/averageTimeTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text=""
        android:layout_below="@id/accuracyTextView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"/>

    <TextView
        android:id="@+id/clickTimesTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text=""
        android:layout_below="@id/averageTimeTextView"
        android:layout_centerHorizontal="true"
```

```

        android:layout_marginTop="16dp"/>

        <com.github.mikephil.charting.charts.LineChart
            android:id="@+id/reactionTimeChart"
            android:layout_width="match_parent"
            android:layout_height="300dp"
            android:layout_above="@id/endButton"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="16dp"
            android:visibility="gone"/>
        <!--Nastaven na gone, jinak v průběhu hlásí absenci dat-->

    </RelativeLayout>

```

activity_leaderboard.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp"
tools:context="com.example.diplomka_test.LeaderboardActivity">

    <!-- Drop-down pro výběr řazení -->
    <Spinner
        android:id="@+id/sortBySpinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentTop="true"
        android:layout_marginEnd="16dp"
        android:layout_marginTop="16dp" />

    <!-- Tlačítka pro filtry -->
    <LinearLayout
        android:id="@+id/buttonContainer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp">

        <Button
            android:id="@+id/easyButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Lehká" />

        <Button
            android:id="@+id/mediumButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Střední"
            android:layout_marginStart="8dp" />

        <Button
            android:id="@+id/hardButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Těžká"
            android:layout_marginStart="8dp" />

    </LinearLayout>

    <!-- Tlačítko pro vymazání obsahu souboru -->
    <Button
        android:id="@+id/clearButton"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="Vymazat výsledky"
        android:layout_below="@+id/recyclerView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"/>

<!-- RecyclerView pro zobrazení žebříčku -->
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/buttonContainer"
    android:paddingTop="8dp"
    android:paddingBottom="8dp"
    android:clipToPadding="false"
    android:scrollbars="vertical" />

</RelativeLayout>

```

item_leaderboard.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/userNameTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="18sp"
        android:text="Jméno" />

    <TextView
        android:id="@+id/attemptsTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Počet pokusů: 0" />

    <TextView
        android:id="@+id/successfulAttemptsTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Úspěšné pokusy: 0" />

    <TextView
        android:id="@+id/averageSpeedTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Průměrný čas: 0 ms" />

</LinearLayout>
```

item_leaderboard_header.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/userNameTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="18sp"
        android:text="Jméno"
        android:gravity="center"/>

    <TextView
        android:id="@+id/attemptsTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Počet Pokusů"
        android:gravity="center"/>

    <TextView
        android:id="@+id/successfulAttemptsTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Úspěšné pokusy"
        android:gravity="center"/>

    <TextView
        android:id="@+id/averageSpeedTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="16sp"
        android:text="Průměrný čas"
        android:gravity="center"/>

</LinearLayout>
```

MainActivity.kt

```
package com.example.diplomka_test

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.decorView.systemUiVisibility =
View.SYSTEM_UI_FLAG_FULLSCREEN
        supportActionBar?.hide()
        setContentView(R.layout.activity_main)

        // Tlačítka pro start obtížnosti
        val startButtonEasy: Button =
findViewById(R.id.startButtonEasy)
        startButtonEasy.setOnClickListener {
            startGame(3)
        }

        val startButtonMedium: Button =
findViewById(R.id.startButtonMedium)
        startButtonMedium.setOnClickListener {
            startGame(10)
        }

        val startButtonHard: Button =
findViewById(R.id.startButtonHard)
        startButtonHard.setOnClickListener {
            startGame(20)
        }

        val showLeaderboardButton: Button =
findViewById(R.id.showLeaderboardButton)
        showLeaderboardButton.setOnClickListener {
            openLeaderboard()
        }
    }

    // Fce pro start hry
    private fun startGame(matrixSize: Int) {
        val intent = Intent(this, GameActivity::class.java)
        intent.putExtra("MATRIX_SIZE", matrixSize)

        val nameEditText: EditText =
findViewById(R.id.userNameEditText)
        val userName = nameEditText.text.toString()
        // Předej jméno uživatele do GameActivity
        intent.putExtra("USER_NAME", userName)

        // Získání počtu pokusů z EditText
        val pokusyEditText: EditText =
```



```

findViewById(R.id.pokusyEditText)
    val pocetPokusu =
pokusyEditText.text.toString().toIntOrNull() ?:
GameActivity.DEFAULT_MAX_ATTEMPTS
    intent.putExtra("MAX_ATTEMPTS", pocetPokusu)

    startActivity(intent)
}

private fun openLeaderboard() {
    val intent = Intent(this, LeaderboardActivity::class.java)
    startActivity(intent)
}
}

```

GameActivity.kt

```
package com.example.diplomka_test

import android.content.Context
import android.content.Intent
import android.graphics.Color
import android.os.Bundle
import android.os.SystemClock
import android.view.View
import android.widget.Button
import android.widget.GridLayout
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import kotlin.random.Random
import java.io.*
import com.github.mikephil.charting.charts.LineChart
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.data.LineData
import com.github.mikephil.charting.data.LineDataSet

class GameActivity : AppCompatActivity() {

    companion object {
        const val MATRIX_SIZE = 5    // Už posílaná z MainActivity
        const val DEFAULT_MAX_ATTEMPTS = 10
    }

    private lateinit var buttons: Array<Array<MyButton>>
    private var totalAttempts = 0
    private var successfulAttempts = 0
    private var litButton: MyButton? = null
    private var maxAttempts = 0
    private lateinit var userName: String

    private val clickTimes = mutableListOf<Long>()

    // Časové proměnné pro měření času mezi kliknutími
    private var startTime: Long = 0
    private var totalTime: Long = 0
    private var clickCount: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Skrytí "status baru"
        window.decorView.systemUiVisibility =
View.SYSTEM_UI_FLAG_FULLSCREEN
        // Skrytí "action baru"
        supportActionBar?.hide()
        setContentView(R.layout.activity_game)

        userName = intent.getStringExtra("USER_NAME") ?: "Unknown"

        val matrixSize = intent.getIntExtra("MATRIX_SIZE", 3)
```

```

        maxAttempts = intent.getIntExtra("MAX_ATTEMPTS",
DEFAULT_MAX_ATTEMPTS)

        createMatrix(matrixSize)

        val endButton: Button = findViewById(R.id.endButton)
        endButton.setOnClickListener {
            endGame()
        }

        // Inicializace startTime při spuštění aktivity
        startTime = SystemClock.elapsedRealtime()

        lightRandomButton()
    }

    // Vytvořen matice
    private fun createMatrix(matrixSize: Int) {
        val gridLayout: GridLayout = findViewById(R.id.gridLayout)
        gridLayout.removeAllViews()
        gridLayout.columnCount = matrixSize
        gridLayout.rowCount = matrixSize

        // Získáme velikost displeje v pixelech
        val screenWidth = resources.displayMetrics.widthPixels
        val screenHeight = resources.displayMetrics.heightPixels

        // Použijeme velikost displeje na velikost tlačítek (málo =
        velký, hodně = malý)
        // - 40 jako rezerva
        val buttonWidth = ((screenWidth - 40) / matrixSize )
        val buttonHeight = ((screenHeight - 40) / matrixSize )

        // Vytvoření tlačítek
        buttons = Array(matrixSize) { row ->
            Array(matrixSize) { col ->
                MyButton(this, row, col).apply {
                    layoutParams = GridLayout.LayoutParams().apply {
                        width = buttonWidth
                        height = buttonHeight
                        rowSpec = GridLayout.spec(row)
                        columnSpec = GridLayout.spec(col)
                    }
                    setOnClickListener {
                        onButtonClick(this)
                    }
                    gridLayout.addView(this)
                }
            }
        }

    }

    private fun lightRandomButton() {
        val random = Random.Default
        val row = random.nextInt(buttons.size)
        val col = random.nextInt(buttons.size)

        litButton = buttons[row][col]
    }

```

```

        litButton?.lightUp()
    }

    private fun onClick(button: MyButton) {
        totalAttempts++

        if (button != litButton) {
            if (totalAttempts >= maxAttempts) {
                endGame()
            }
            return
        }

        successfulAttempts++
        button.turnOff()

        // Získání času mezi kliknutími a aktualizace celkového času
        a počtu kliknutí
        val currentTime = SystemClock.elapsedRealtime()
        if (clickCount > 0) {
            val reactionTime = currentTime - startTime
            totalTime += reactionTime
            clickTimes.add(reactionTime)
        }
        clickCount++
        startTime = currentTime

        if (totalAttempts < maxAttempts) {
            lightRandomButton()
        } else {
            endGame()
        }
    }

    private fun removeMatrix() {
        val gridLayout: GridLayout = findViewById(R.id.gridLayout)
        gridLayout.removeAllViews()
    }

    private fun showReactionTimeChart() {
        val chart: LineChart = findViewById(R.id.reactionTimeChart)
        val entries = clickTimes.mapIndexed {index, time ->
Entry(index.toFloat(), time.toFloat())}

        val dataSet = LineDataSet(entries, "Reaction Times")
        val lineData = LineData(dataSet)

        chart.data = lineData
        chart.invalidate() // Refresh grafu
    }

    private fun endGame() {
        removeMatrix()
        val accuracy = calculateAccuracy()
        val accuracyTextView: TextView =
findViewById(R.id.accuracyTextView)
        accuracyTextView.text = String.format("Přesnost: %.2f %%",
accuracy)
    }

```

```

        // Získání hodnoty matrixSize
        val matrixSize = intent.getIntExtra("MATRIX_SIZE", 3)

        // Výpočet průměrného času kliknutí a zobrazení na konci hry
        val averageTimePerClick = if (clickCount > 0)
totalTime.toDouble() / clickCount else 0.0
        val averageTimeTextView: TextView =
findViewById(R.id.averageTimeTextView)
        averageTimeTextView.text = String.format("Průměrný čas
kliknutí: %.2f ms", averageTimePerClick)

        // Uložení výsledku do souboru
        val userResult = DataManager.UserResult(userName, matrixSize,
totalAttempts, successfulAttempts, averageTimePerClick.toLong())
        DataManager.saveResultToFile(userResult, this) // Předáváme
aktuální kontext

        // Zobrazí graf reakčních časů
        val reactionChart =
findViewById<LineChart>(R.id.reactionTimeChart)
        reactionChart.visibility = View.VISIBLE
        showReactionTimeChart()
        // Nastavení popisku grafu
        reactionChart.getDescription().setText("Reakční čas/Pokusy");

        val endButton: Button = findViewById(R.id.endButton)
        endButton.text = "Zpět na menu"
        endButton.visibility = View.VISIBLE
        endButton.setOnClickListener {
            returnToMenu()
        }
    }

    private fun returnToMenu() {
        val intent = Intent(this, MainActivity::class.java)
        startActivity(intent)
    }

    private fun calculateAccuracy(): Double {
        return if (totalAttempts == 0) {
            0.0
        } else {
            (successfulAttempts.toDouble() / totalAttempts) * 100
        }
    }
}

class MyButton(context: Context, val row: Int, val col: Int) :
androidx.appcompat.widget.AppCompatButton(context) {

    init {
        setBackgroundColor(Color.WHITE)
    }

    fun lightUp() {

```

```

        setBackgroundColor(Color.RED)
    }

    fun turnOff() {
        setBackgroundColor(Color.WHITE)
    }
}

object DataManager {
    private const val FILENAME = "user_results.txt"

    // Třída UserResult pro ukládání výsledků
    data class UserResult(
        val userName: String,
        val matrixSize: Int,
        val totalAttempts: Int,
        val successfulAttempts: Int,
        val averageSpeed: Long
    )

    fun saveResultToFile(result: UserResult, context: Context) {
        val line =
            "${result.userName},${result.matrixSize},${result.totalAttempts},${result.successfulAttempts},${result.averageSpeed}"
        try {
            context.openFileOutput(FILENAME, Context.MODE_APPEND).use {
                it.write((line + "\n").toByteArray())
            }
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }
}

```

LeaderboardActivity.kt

```
package com.example.diplomka_test

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.LinearLayoutManager
import java.io.FileNotFoundException
import android.widget.Toast
import android.widget.AdapterView
import android.widget.AdapterView

class LeaderboardActivity : AppCompatActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var layoutManager: LinearLayoutManager
    private lateinit var leaderboardAdapter: RecyclerView.Adapter<*>
    private lateinit var sortBySpinner: Spinner
    private var currentMatrixSize: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_leaderboard)

        recyclerView = findViewById(R.id.recyclerView)
        layoutManager = LinearLayoutManager(this)
        recyclerView.layoutManager = layoutManager

        // Inicializace spinneru pro filtraci podle výsledku
        sortBySpinner = findViewById(R.id.sortBySpinner)

        // Nastavení posluchače pro spinner
        setupSpinner()

        val easyButton: Button = findViewById(R.id.easyButton)
        val mediumButton: Button = findViewById(R.id.mediumButton)
        val hardButton: Button = findViewById(R.id.hardButton)
        val clearButton: Button = findViewById(R.id.clearButton)

        // Tlačítka pro filtraci leaderboardu podle obtížnosti
        easyButton.setOnClickListener {
            currentMatrixSize = 3
            loadUsersByMatrixSize(3)
        }

        mediumButton.setOnClickListener {
            currentMatrixSize = 10
            loadUsersByMatrixSize(10)
        }
    }
}
```

```

        hardButton.setOnClickListener {
            currentMatrixSize = 20
            loadUsersByMatrixSize(20)
        }

        // Tlačítko pro vymazání výsledků
        clearButton.setOnClickListener {
            clearResults()
        }
        // Bez vybrání obtížnosti zobrazí všechny
        loadAllUsers()
    }

    // Fce pro zobrazení všech výsledků
    private fun loadAllUsers() {
        val users = loadUsersFromStorage()
        leaderboardAdapter = LeaderboardAdapter(users)
        recyclerView.adapter = leaderboardAdapter
    }

    // Fce pro filtraci výsledků podle velikosti matice (obtížnosti)
    private fun loadUsersByMatrixSize(matrixSize: Int) {
        val users = loadUsersFromStorage().filter { it.matrixSize ==
matrixSize }
        leaderboardAdapter = LeaderboardAdapter(users)
        recyclerView.adapter = leaderboardAdapter
    }

    // Fce pro vymazání obsahu .txt s výsledky
    private fun clearResults() {
        val fileName = "user_results.txt"
        try {
            applicationContext.deleteFile(fileName)
            Toast.makeText(this, "Výsledky byly úspěšně smazány.",
Toast.LENGTH_SHORT).show()
            loadAllUsers()
        } catch (e: Exception) {
            Toast.makeText(this, "Nastala chyba při mazání výsledků.",
Toast.LENGTH_SHORT).show()
        }
    }

    // Fce pro načtení výsledků z .txt
    private fun loadUsersFromStorage(): List<User> {
        val users = mutableListOf<User>()
        try {
            val fileName = "user_results.txt"
            applicationContext.openFileInput(fileName).bufferedReader().useLines {
lines ->
                lines.forEach { line ->
                    val parts = line.split(",")
                    if (parts.size == 5) {
                        val userName = parts[0]
                        val matrixSize = parts[1].toInt()

```



```

        val totalAttempts = parts[2].toInt()
        val successfulAttempts = parts[3].toInt()
        val averageSpeed = parts[4].toLong()
        users.add(User(userName, matrixSize,
totalAttempts, successfulAttempts, averageSpeed))
    }
}
}
} catch (e: FileNotFoundException) {
    // Soubor neexistuje nebo je prázdný
}
return users
}

class LeaderboardAdapter(private val users: List<User>) :
RecyclerView.Adapter<LeaderboardAdapter.LeaderboardViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): LeaderboardViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.item_leaderboard,
parent, false)
        return LeaderboardViewHolder(view)
    }

    override fun onBindViewHolder(holder: LeaderboardViewHolder,
position: Int) {
        val user = users[position]
        holder.bind(user)
    }

    override fun getItemCount(): Int {
        return users.size
    }

    inner class LeaderboardViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        fun bind(user: User) {

itemView.findViewById<TextView>(R.id.userNameTextView).text =
user.name

itemView.findViewById<TextView>(R.id.attemptsTextView).text = "Počet
pokusů: ${user.totalAttempts}"

itemView.findViewById<TextView>(R.id.successfulAttemptsTextView).text
= "Úspěšné pokusy: ${user.successfulAttempts}"

itemView.findViewById<TextView>(R.id.averageSpeedTextView).text =
"Průměrný čas: ${user.averageSpeed} ms"
        }
    }
}

// Enum třída pro možné hodnoty řazení
enum class SortByOption {

```

```

        USER_NAME,
        TOTAL_ATTEMPTS,
        SUCCESSFUL_ATTEMPTS,
        AVERAGE_SPEED
    }

    // Globální proměnná pro uchování aktuálně vybrané možnosti řazení
    private var currentSortOption = SortByOption.USER_NAME

    // V metodě setupSpinner() nastavení posluchače pro změnu výběru
    ve spinneru
    private fun setupSpinner() {
        val sortByOptions = arrayOf("Jméno", "Počet Pokusů", "Úspěšné
pokusy", "Průměrný čas")
        val adapter = ArrayAdapter(this,
        android.R.layout.simple_spinner_item, sortByOptions)

        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdo
wn_item)
        sortBySpinner.adapter = adapter

        // Nastavení posluchače pro změnu výběru ve spinneru
        sortBySpinner.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>?, view:
            View?, position: Int, id: Long) {
                val sortByOption = when (position) {
                    0 -> SortByOption.USER_NAME
                    1 -> SortByOption.TOTAL_ATTEMPTS
                    2 -> SortByOption.SUCCESSFUL_ATTEMPTS
                    3 -> SortByOption.AVERAGE_SPEED
                    else -> SortByOption.USER_NAME
                }
                currentSortOption = sortByOption
                // Řazení žebříčku podle vybrané možnosti
                sortLeaderboard()
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {
                // Nepotřebujeme implementovat
            }
        }
    }

    // Metoda pro řazení seznamu uživatelů podle aktuálního výběru ve
    spinneru
    private fun sortLeaderboard() {
        val users = if (currentMatrixSize != 0) {
            loadUsersFromStorage().filter { it.matrixSize ==
            currentMatrixSize }.toMutableList()
        } else {
            loadUsersFromStorage().toMutableList()
        }

        users.sortWith(compareBy<User> {
            when (currentSortOption) {
                SortByOption.USER_NAME -> it.name
                SortByOption.TOTAL_ATTEMPTS -> -it.totalAttempts
            }
        })
    }

```

```
                SortByOption.SUCCESSFUL_ATTEMPTS -> -
it.successfulAttempts
                SortByOption.AVERAGE_SPEED -> it.averageSpeed
            }
        })

        leaderboardAdapter = LeaderboardAdapter(users)
        recyclerView.adapter = leaderboardAdapter
    }
}
```

User.kt

```
package com.example.diplomka_test

data class User(
    val name: String,
    val matrixSize: Int,
    var totalAttempts: Int,
    var successfulAttempts: Int,
    val averageSpeed: Long
)
```