VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VÝUKOVÝ SOFTWARE PRO VIZUÁLNÍ A TEXTOVÉ PROGRAMOVÁNÍ V LUA/LÖVE

LEARNING SOFTWARE FOR VISUAL AND TEXT-BASED PROGRAMMING IN LUA/LÖVE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PETR MEDEK**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání diplomové práce

Student:	**Medek Petr, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Výukový software pro vizuální a textové programování v Lua/LÖVE** **Learning Software for Visual and Text-Based Programming in Lua/LÖVE**
Kategorie:	Uživatelská rozhraní

Zadání:

1. Seznamte se s programovacím jazykem Lua a s rámcem LÖVE. Prozkoumejte možnosti vizuálního ("drag and drop") programování a existující nástroje vč. podpory pro generování kódu programovacího jazyka. Nastudujte přístupy k výuce programování a existující tutoriály (nejen) pro Lua/LÖVE.
2. Navrhněte webovou aplikaci pro výuku programování v Lua/LÖVE s podporou vizuální ("drag and drop") i textové tvorby programu. Aplikace musí podporovat jednoduché spuštění výsledných programů na platformě Android (s automatickým nahráním na zařízení).
3. Po konzultaci s vedoucím aplikaci implementujte. Vytvořte také výukové tutoriály s různou rychlostí postupu a obtížností (např. strukturované a objektové programování) vhodné pro výuku na základních (oba stupně) a středních školách.
4. Řešení otestujte, vyhodnoťte a diskutujte výsledky. Výsledný software publikujte jako open-source.

Literatura:

- S. Kenlon. Modular Programming with LÖVE. In: Developing Games on the Raspberry Pi. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4170-7. Dostupné z: [https://doi.org/10.1007/978-1-4842-4170-7_3]
- E. Pasternak, R. Fenichel a A. N. Marshall. Tips for creating a block language with blockly. In: 2017 IEEE Blocks and Beyond Workshop. IEEE, USA, 2017. Dostupné z: [https://doi.org/10.1109/BLOCKS.2017.8120404]
- Alternatives to Scratch. *Scratch Wiki* [online]. 2020 [cit. 2020-10-26]. Dostupné z: [https://en.scratch-wiki.info/wiki/Alternatives_to_Scratch]
- Programming in DRAKON. Drakon.Tech [online]. 2021 [cit. 2021-08-18]. Dostupné z: [https://drakon.tech/read/programming_in_drakon]

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, rozpracovaný bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2021
Datum odevzdání:	18. května 2022
Datum schválení:	11. října 2021

Abstrakt

Tato práce se zabývá vizuálním programováním v jazyce Lua s frameworkem LÖVE. V rámci práce vznikla webová aplikace pro výuku programování v Lua/LÖVE. Aplikace podporuje jak blokové, tak i textové programování. Vytvořené hry lze jednoduše spustit na platformě Android pomocí vytvořené Android aplikace. Webová aplikace obsahuje také výukové tutoriály vhodné pro výuku na základních a středních školách.

Abstract

This thesis pertains visual programming in Lua language with LÖVE framework. Within this thesis was created web application for teaching programming in Lua/LÖVE. Application supports both block and text based programming. Created games can be easily launched on Android platfrom using the Android app that was created in this thesis. The web app also includes tutorials that are well suited for education in schools.

Klíčová slova

vizuální programování, Blockly, blokové programování, Lua, LÖVE, Scratch, AppInventor, výuka programování, výukový software, 2D hry, hry, Android, Android hry, webová aplikace, tutoriály

Keywords

visual programming, Blockly, block based programming, Lua, LÖVE, Scratch, AppInventor, programming education, educational software, 2D games, games, Android, Android games, web application, tutorials

Citace

MEDEK, Petr. *Výukový software pro vizuální a textové programování v Lua/LÖVE*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. MAREK RYCHLÝ, Ph.D.

Výukový software pro vizuální a textové programování v Lua/LÖVE

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Marka Rychlého Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Medek
9. května 2022

Poděkování

Tímto bych chtěl poděkovat RNDr. Marku Rychlému Ph.D. za vedení této práce, trpělivost a dobré rady.

Obsah

Kapitola 1

Úvod

Hry pro mobilní platformy jsou v posledních letech velmi oblíbené a pro jejich vytváření existuje řada programovacích jazyků a frameworků. Jedním z těchto jazyků je `Lua`, skriptovací procedurální jazyk, který je dostatečně malý, aby se vešel na různé platformy. V této práci je používán společně s Lua framework `LÖVE` pro tvorbu 2D her, který funguje na různých platformách, v této práci je ale hlavně používán pro platformu Android. Pro tvorbu her existuje řada tutoriálů, které mají různé přístupy k výuce programování. Jedním z těchto přístupů je vizuální programování, kdy příkazy mají podobu tzv. bloků, které dohromady tvoří program.

Cílem této práce je seznámení se s programovacím jazykem Lua, s frameworkem `LÖVE` a s již existujícími přístupy k výuce programování. V rámci práce vznikla webová aplikace pro výuku pomocí vizuálního programování, která funguje jako nadstavba nad `Lua/LÖVE`. Webová aplikace využívá bloků a principu "drag and drop", která zjednodušuje výuku programování. Aplikace podporuje spuštění vytvořených her na platformě Android. Software aplikace je publikován jako open-source. V aplikaci jsou zahrnuty i výukové materiály, které lze využít pro výuku na základních a středních školách. Vznikla také Android aplikace, která umožňuje jednoduché nahrávání projektů z webové aplikace na zařízení Android.

Programovací jazyk Lua a framework `LÖVE` jsou představeny v kapitole 2. Je zde popsána architektura frameworku `LÖVE`, konfigurační soubory a jeho moduly. V kapitole 3 jsou rozebrány stávající přístupy k výuce programování a jsou představeny existující tutoriály pro `Lua/LÖVE`, které jsou mezi sebou porovnány. Kapitola 4 se zabývá typy vizuálního programování (blokové programování, programování toku dat), dále jsou srovnány vybrané nástroje pro vizuální programování, jako jsou například Scratch nebo Blockly. V kapitole 5 jsou nejprve specifikovány požadavky na webovou aplikaci a dále jsou popsány existující aplikace. V další části je navržena Android aplikace pro nahrávání a spouštění her, dále se zaměřuje na komunikaci mezi webovou aplikací a Android aplikací. Následující část kapitoly se věnuje návrhu webové aplikace, například editorům kódu a spouštění her v prohlížeči. Je navrženo uživatelské rozhraní a struktura tutoriálů pro tvorbu her. Webová aplikace umožňuje uživateli pomocí bloků programovat 2D hry v jazyce `Lua/LÖVE` a vytvořené hry spouštět na platformě Android. Kapitola 6 vysvětluje samotnou implementaci Android aplikace, představuje uživatelské rozhraní, jak probíhá komunikace s webovou aplikací a popisuje princip Firebase Cloud Messaging. Nakonec se kapitola věnuje spouštěním her na Android zařízení. Sedmá kapitola 7 se zabývá implementací webové aplikace, je zde popsáno fungování Lua Language Serveru a podrobně rozepsáno fungování části backend (generování her, komunikace s Androidem, ukládání dat). Frontend popisuje práci s editory kódu a spouštění hry v prohlížeči. Nakonec je popsáno vytváření bloků a tutoriály,

které vznikly v rámci práce. Osmá kapitola 8 popisuje testování a publikování aplikace jako open-source a jsou diskutovány možnosti dalšího vývoje webové i Android aplikace. Závěr 9 shrnuje celou diplomovou práci a způsob, jakým bylo zadání splněno.

Kapitola 2

Lua/LÖVE

V této kapitole je představen programovací jazyk `Lua`, viz 2.1. Dále je popsán framework `LÖVE` v sekci 2.2, kde je vysvětleno jeho fungování a také jsou prezentovány klíčové funkce a moduly tohoto frameworku.

2.1 Jazyk Lua

`Lua` je open-source skriptovací jazyk, který podporuje jak procedurální, tak objektově orientované programování. `Lua` je dynamicky interpretovaný programovací jazyk s automatickou správou paměti – obsahuje tzv. garbage collector. Je jednoduché se jazyk `Lua` naučit a díky malé velikosti zhruba 1.3 MB je jeden z nejpoužívanějších skriptovacích jazyků pro vestavěné systémy a programování her. Další výhodou je, že `Lua` je velmi rychlý jazyk [13].

`Lua` může sloužit jako vestavěný jazyk, to znamená, že `Lua` není samostatná aplikace ale knihovna, kterou lze propojit s jinými aplikacemi. Jazyk `Lua` byl navrhnut, aby spolupracoval s programovacím jazykem C, a to díky C API, které umožňuje vytvoření vazby mezi jazykem `Lua` a knihovnami jazyka C [7]. `Lua` funguje na všech platformách, které mají standardní kompilátor jazyka C.

Syntax jazyka `Lua` je jednoduchá na čtení a je dobře pochopitelná i pro začátečníky, proto je její učení relativně snadné. Na druhou stranu lze funkčnost jazyka `Lua` rozšířit řadou knihoven. Těchto knihoven existuje díky aktivní komunitě mnoho [13].

2.2 Framework LÖVE

`LÖVE` je populární framework pro tvorbu 2D her a je dostupný jako open-source. `LÖVE` je nadstavba nad skriptovací jazyk `Lua`, který je podrobně popsaný v sekci 2.1, a využívá OpenGL[1]. `LÖVE` je jednoduchý na používání a díky velké komunitě aktivních uživatelů existuje mnoho dostupných materiálů a knihoven. Aplikace vytvořené pomocí `LÖVE` fungují na platformách Windows, Mac OS X, Linux Android a iOS [4].

Conf.lua

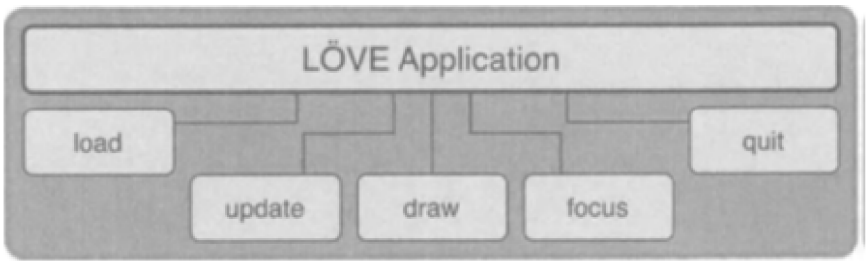
Pro nastavení aplikace slouží soubor `conf.lua`, který obsahuje funkci `love.conf()`. Soubor `conf.lua` je spuštěn před souborem `main.lua`. Tento soubor definuje například šířku

[1] https://www.opengl.org/

a výšku hry, verzi, název a autora. V souboru je možné vypínat a zapínat moduly, které bude aplikace používat. Podobu konfiguračního souboru lze najít na LÖVE wiki[2] [1].

Architektura

LÖVE na rozdíl od ostatních frameworků funguje na základě tzv. callback funkcí. Tyto funkce musí být deklarovány pro správné fungování programu. LÖVE volá tyto funkce, pokud jsou deklarovány. Příklad architektury je blíže popsán v obrázku níže 2.1 [21].



Obrázek 2.1: Architektura LÖVE a její callback funkce [21]

Pro fungování vytvářeného programu jsou klíčové funkce: `love.load()`, `love.update()` a `love.draw()`. Funkce určují základní fungování programu a musí být definované v každé aplikaci. Mezi nejdůležitější funkce patří:

- `love.load()` – Tato funkce je volána vždy při spuštění hry, je podobná funkci `main()` v jazyku C. Funkce je volána pouze jednou, využívá se pro načítání zdrojů, pro nastavení systému a globálních proměnných [21].
- `love.update(dt)` – Tato funkce je volána opakovaně. Parametrem funkce je `dt` – delta time, což je čas uběhlý od posledního volání funkce v sekundách. Tato hodnota je velmi malá, proto se funkce volá několikrát za sekundu [21]. Zjednodušeně řečeno je to čas, za který je vykreslen jeden snímek. Funkce `love.update()` se stará o události a matematiku, je volána předtím, než je snímek vykreslen [4].
- `love.draw()` – Tato funkce slouží pro vykreslování objektů na obrazovku. Příkazy na vykreslení volané mimo tuto funkci nebudou mít žádný účinek, nebudou fungovat [21].
- `love.mousepressed/mousereleased(x,y, button)` – Tyto funkce se starají o stisknutí/uvolnění tlačítka myši. Funkce při zavolání získá pozici x, y a informaci, jaké tlačítko myši bylo použito.
- `love.keypressed/released(key, unicode)` – Tyto funkce zajišťují stisknutí/uvolnění tlačítka klávesnice. Parametr `key` obsahuje název stisknuté klávesy a parametr `unicode` obsahuje ASCII kód klávesy[3].
- `love.focus()` – Funkce je používána pro zjištění, zda je uživatel v aktuálním okně. Tato funkce zajišťuje pozastavení hry, pokud uživatel klikne na jiné okno [21].
- `love.quit()` – Funkce je volána po uzavření okna uživatelem, stará se o uložení herních dat a uvolnění paměti počítače [21].

[2] https://love2d.org/wiki/Config_Files
[3] https://love2d.org/wiki/KeyConstant

Moduly

`Lua/LÖVE` nabízí řadu modulů, kde každý modul se specializuje na určitou oblast, například audio. V modulu se nachází všechny funkce spojené s danou oblastí. Níže jsou popsány některé z hlavních modulů:

- `love.audio()` – Modul přináší rozhraní pro přehrávání zvuku, lze zde najít funkce jako hlasitost, pozastavení a spuštění zvuku.
- `love.event()` – Modul se stará o odbavení událostí, jako je stisknutí tlačítka myši nebo klávesnice. Fronta událostí je používaná pro komunikaci uvnitř aplikace.
- `love.filesystem()` – Tento modul umožňuje přístup k lokálnímu systému souborů. Aplikace má přístup pouze ke dvěma adresářům: kořenové složce, kde je uložena hra, a ke složce, kde uživatel ukládá hry. V každém operačním systému jsou složky definovány jinak.
- `love.physics()` – Tento modul přidává fyziku do her, využívá k tomu knihovny Box2D[4].
- `love.graphics()` – `LÖVE` graphics patří k nejdůležitějším modulům, protože pomocí něj lze vykreslovat jednotlivé objekty, jako jsou například tvary, linie, obrázky nebo také speciální objekty jako částice nebo plátno (canvas). V `LÖVE` existuje několik typů grafických objektů:
  - `Canvas` – `Canvas` je používán pro vykreslování mimo obrazovku. Změny nejsou viditelné, dokud není celý `Canvas` vykreslen na obrazovku. Vykreslení probíhá ve funkci `love.draw()` popsané výše. Vykreslením celého `Canvas` místo jednotlivých objektů se snižuje počet operací, které by byly nutné pro každý snímek.
  - `Drawable` – Všechny objekty, které mohou být vykresleny na obrazovku.
  - `Font` – Definuje tvar písma.
  - `Image` – Používán pro vykreslování obrázků.
  - `ParticleSystem` – Využívá se pro vytváření částicových efektů, jako je kouř nebo oheň.
  - `Quad` – Čtyřúhelník s texturou a souřadnicemi. Používá se při vytváření animace, která je vytvořena ze série objektů uložených v jenom obrázku. `Quad` se využívá pro zobrazení jen určité části tohoto obrázku.
  - `SpriteBatch` – Využívá jednoho obrázku pro vykreslení několika identických kopií. To se využívá například pro opakující se pozadí.

[4] https://box2d.org/

Kapitola 3

Výuka programování a tutoriály

V této kapitole jsou charakterizovány jednotlivé přístupy k výuce programování, jedná se o analýzu kódu, stavební bloky, jednoduché jednotky a celé systémy, viz sekce 3.1. Jsou zde shrnuty výhody a nevýhody jednotlivých přístupů a je navrhnuta kombinace, která je pro tuto práci nejvhodnější. Sekce 3.2 pak analyzuje existující tutoriály pro `Lua/LÖVE`, kde je porovnáno, jak vypadají a jak přistupují k výuce. Na závěr je shrnuto, které typy tutoriálů jsou pro tuto práci přínosné.

3.1 Přístupy k výuce programování

Na výuku programování se lze dívat z více úhlů, nejde pouze o pasivní přijímání informací, ale také o získání zkušeností při samotné tvorbě programu. Cynthia C. Selby [19] ve svém článku popisuje čtyři možné přístupy k výuce programování a hodnotí jejich přínos pro studenty. Tyto přístupy autorka pojmenovala: analýza kódu, stavební bloky, jednoduché jednotky a celé systémy.

Analýza kódu (code analysis)

Tento přístup klade důraz na čtení a pochopení logiky programování předtím, než studenti sami začnou psát programy. Je to založeno na použití pseudokódu[1]. U tohoto přístupu tedy není potřeba používat žádný konkrétní programovací jazyk či vývojové prostředí. Výhodou je, že se studenti nemusí učit, jak funguje nové vývojové prostředí, a nejsou tak příliš zahlceni novými informacemi. Studenti se mohou rovnou soustředit na logiku a pochopení základních algoritmů, jako je například lineární vyhledávání. Studenti se také učí hledat chyby v kódu a kontrolovat správné fungování programu. Nevýhodu tohoto přístupu je, že se studenti nedostanou k samotnému programování, což může některé studenty odradit. Tento přístup také nepodporuje samostatné učení, protože studenti nedostávají zpětnou vazbu. Dalším negativem je, že pro některé studenty je obtížné napsat vlastní program, i když chápou již vytvořené řešení. Navíc používání pseudokódu může vést k tomu, že studenti v budoucnu nepochopí syntax programovacího jazyka [19].

Stavební bloky (building blocks)

V tomto přístupu se studenti nejprve učí samostatně jednotlivé části, než je začnou kombinovat dohromady. Tento přístup se zaměřuje na vysvětlení syntaxe jazyka, například pro-

[1] https://it-slovnik.cz/pojem/pseudokod

měnné, přiřazování, podmínky a cykly. Všechny tyto části mohou být učeny samostatně. Tyto znalosti je pro tvorbu programů možné kombinovat dohromady. Na rozdíl od přístupu `Analýza kódu`, `Stavební bloky` využívají specifického programovacího jazyka a vývojového prostředí. Vývojové prostředí dává studentovi okamžitou zpětnou vazbu, zda je kód syntakticky správně a bez chyb. Nevýhodou může ale být složitost samotného vývojového prostředí pro začátečníky. Navíc tento přístup nezaručuje, že kód, který je syntakticky správně, je i správně logicky [19].

Jednoduché jednotky (simple units)

Studenti v tomto přístupu využívají již vytvořené části kódu pro vytvoření komplexnějšího programu. Studenti si vytváří sadu užitečných fragmentů kódu (`jednoduché jednotky` – simple units) tím, že řeší jednoduché a dobře definované problémy a tyto části kódu poté studenti využijí při řešení komplexnějších problémů. Příkladem takových jednotek může být: čtení vstupu, vypsání výstupu a cykly. Tyto jednotky je možné spojovat pro vytvoření složitějšího programu. Výhodou této metody je, že studenti mají přístup k fragmentům kódu a tedy lepší startovní pozici pro vývoj programu. Studenti se také učí v kódu hledat chyby a je pro ně jednodušší identifikovat problémy v logice programu. Čím se zvyšuje složitost problému, tím je třeba mít větší znalosti. Tento přístup učí studenty složité problémy rozdělit na menší, lépe řešitelné části. Je zde ale potřeba vývojové prostředí, které může být pro studenty složité a způsobovat jim potíže, které jsou blíže vysvětleny v přístupu `Stavební bloky` [19].

Celé systémy (full systems)

Přístup celých systémů je založen na tom, že jednotlivé koncepty jsou představeny, až když jsou potřeba pro řešení problému. Studenti používají od začátku celý programovací jazyk a vývojové nástroje s ním související. Znamená to, že student se učí vše naráz: fungování vývojového prostředí, chování částí kódu, hledání chyb atd. Jednotlivé koncepty jazyka a jeho syntax se ale učí postupně, když jsou potřeba pro vyřešení problému. Tento přístup se může zdát složitý, ale jeho výhodou je, že se zde mohou řešit reálné problémy, které studenti znají ze života. Některé studenty to motivuje k další práci, ale pro začátečníky to může být náročné. Přístup vede k učení řešení problémů a k rozdělení problémů na menší řešitelné části [19].

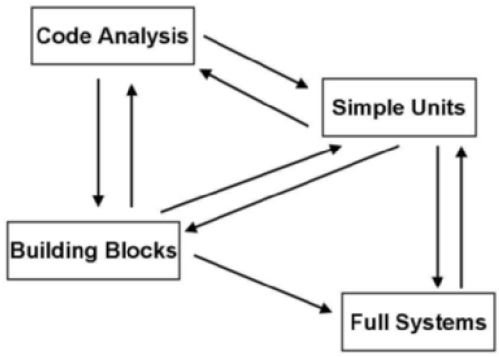
3.1.1 Shrnutí

Programátor by měl být schopen napsat, vysvětlit a hledat chyby v kódu. Některé přístupy umožňují tyto znalosti rozvíjet ve větší míře:

- `Analýza kódu (code analysis)` – hledání chyb, vysvětlení funkčnosti
- `Stavební bloky (building blocks)` – hledání chyb, vysvětlení funkčnosti
- `Jednoduché jednotky (simple units)` – psaní kódu
- `Celé systémy (full systems)` – psaní kódu

Z toho vyplývá, že kombinací těchto přístupů student získá nebo prohloubí své znalosti a naučí se lépe programovat. Autorka článku `Four approaches to teaching programming` [19] navrhuje několik možných kombinací těchto přístupů. Prvním je lineární kombinace,

která kombinuje přístupy od `stavebních bloků` přes `jednoduché jednotky` po přístup `celé systémy`. Jako druhou možnou kombinaci navrhuje autorka použít přístup `Analýza kódu` přes `stavební bloky`, `jednoduché jednotky` až po `celé systémy`. Student se nejprve naučí funkčnost algoritmů pomocí pseudo kódu, poté se naučí syntax a jednoduché části vybraného jazyka. Tyto znalosti dále využije pro tvorbu znovupoužitelných částí kódu, které zkombinuje pro tvorbu reálných programů. Všechny možné kombinace přístupů, které autorka navrhuje, lze vidět v následujícím obrázku 3.1.



Obrázek 3.1: Možné kombinace přístupů [19]

Příkladem může být nástroj Scratch, na který se zaměřuje sekce 4.2.2, Scratch kombinuje přístupy v pořadí `stavební bloky`, `jednoduché jednotky` a `celé systémy`. Student v tomto nástroji využívá 2D bloky, které reprezentují část kódu. Propojením více bloků se vytváří program. Student se učí, jak jednotlivé bloky fungují a jak mohou být logicky spojeny dohromady. Když se student naučí jednotlivé části, je schopen vytvářet větší a složitější programy. Tato práce vychází z nástroje Scratch, který kombinuje tyto přístupy k učení. Jeví se to jako nejvhodnější způsob pro blokové programování, studenti jsou takto schopni se naučit programovat, jelikož začínají od nejjednodušších konceptů a dostávají se ke složitějším.

3.2 Tutoriály pro Lua/LÖVE

Na internetu existuje řada tutoriálů pro `Lua/LÖVE`. V této práci jsou popsány a porovnány čtyři existující tutoriály. Níže popsané tutoriály jsou vybrány z toho důvodu, že se od sebe liší a každý z nich má své silné a slabé stránky.

3.2.1 Oficiální tutoriály pro Lua/LÖVE

Na oficiálních wiki stránkách `Lua/LÖVE`[2] existuje několik tutoriálů pro práci s frameworkem `LÖVE`. Většina tutoriálů je napsaná v angličtině a také v několika dalších jazycích, například v němčině, čínštině či španělštině, nicméně pouze jedno téma je také v češtině. Tutoriály jsou rozděleny dle témat: například Fyzika, Audio nebo Animace. Jednotlivé tutoriály ale na sebe nijak nenavazují. Náročnost jednotlivých témat je rozdílná. Lze zde třeba najít jednoduché téma "Getting Started", ale také pokročilé téma "Physics". Navíc není náročnost

[2] https://www.love2d.org/wiki/Category:Tutorials

témat nijak pro uživatele označena. Všechny tutoriály obsahují část kódu a slovní popis, co kód dělá a jak funguje, v některých případech jsou zde i názorné obrázky. Na stránkách je i dokumentace k Lua/LÖVE a k jejím modulům, takže je jednoduché si najít informace o neznámých funkcích. Tyto oficiální tutoriály jsou spíše vhodné pro vyřešení konkrétního problému než pro naučení se se samotným frameworkem LÖVE.

3.2.2 Tutoriál – learn2love

Dalším tutoriálem pro Lua/LÖVE je kniha[3], jejímž autorem je Jay Thomas a která se zaměřuje na fungování jazyka Lua a frameworku LÖVE. V první kapitole autor vysvětluje základy jazyka Lua – například cykly, funkce a tabulky. V druhé kapitole autor představuje framework LÖVE a vysvětluje jeho funkčnost. Jednotlivé podkapitoly na sebe navazují. Nejprve jsou vysvětleny jednoduché principy LÖVE, jako je struktura programu nebo callback funkce, další podkapitoly se zabývají programováním hry Brakeout. Třetí kapitola se zabývá pokročilým programováním v Lua, kapitola ale není bohužel dokončena. Samotná struktura tutoriálu je obdobná jako v oficiálních tutoriálech pro LÖVE, jedná se o část kódu a slovní vysvětlení jak kód funguje. Na konci každé podkapitoly je zadán úkol na procvičení probíraného tématu, uživatel si může úkol vyzkoušet a ověřit si, zda dané téma pochopil. Tutoriál je zpracován srozumitelně, takže i začátečník je schopen pochopit, jak Lua/LÖVE funguje. Začíná základními tématy a poté následují těžší témata. Tutoriál vždy staví na tom, co se již uživatel v předchozích kapitolách naučil.

3.2.3 Kniha – Developing Games on the Raspberry Pi

Kniha, jejímž autorem je Seth Kenlon, je zaměřena především na vývoj her pro Raspberry Pi, nicméně kapitoly dvě a tři se zabývají frameworkem LÖVE. Druhá kapitola začíná instalací LÖVE a pokračuje vysvětlením základních konceptů. V této kapitole je na hře s kostkami ilustrováno, jak vytvořit hru pomocí LÖVE. Tutoriál postupuje krok po kroku a autor podrobně vysvětluje, co dělá a proč. Vše je doplněno kódem, který si čtenář může sám vyzkoušet. Na konci kapitoly se nachází několik úkolů, kde autor navrhuje, jak hru vylepšit. Třetí kapitola se zabývá modulárním programováním v LÖVE, autor čtenářům na tvorbě hry Blackjack vysvětluje, jak fungují objekty a třídy. Podobně jako v předchozí kapitole autor vytváří hru Blackjack krok po kroku a popisuje její funkčnost, vše je doplněno kódem a na konci kapitoly se opět nachází úkoly pro vylepšení hry. Výhody tohoto tutoriálu spočívají v tom, že pomocí programování hry se čtenář naučí principy LÖVE. Autor klade důraz na logiku hry a vysvětluje, jak přijít k řešení daného problému pomocí LÖVE.

3.2.4 Tutoriál – Lua and LÖVE 11

Na webových stránkách[4] se nachází tutoriály pro tvorbu jednoduchých her v LÖVE, každý z jedenácti tutoriálů obsahuje pravidla dané hry, logiku hry a odkaz na již vytvořenou hru. Tutoriál je vytvořen z pohledu logiky hry, jednotlivé části jsou názorně ukázány pomocí obrázků samotné hry, kódu a jeho popisu. Tutoriál jde krok po kroku a zvládne ho následovat jakýkoliv začátečník. Jednotlivé části tutoriálu jsou graficky rozlišeny, takže je orientace v textu jednoduchá. V každé podkapitole je možnost nahlédnout do dosud vytvořeného kódu hry. Tutoriály jsou zaměřeny na úplné začátečníky, proto ale neobsahují některé pokročilejší funkce, které jiné tutoriály mohou nabízet. Styl tohoto tutoriálu je přehledný, jednoduchý

[3] https://rvagamejams.com/learn2love/
[4] https://simplegametutorials.github.io/love/

a hlavně pro uživatele zábavný. Uživatel se učí principy LÖVE během programování her, navíc tutoriál nahlíží na LÖVE očima uživatele, který chce vytvořit hru, což může být pro uživatele zábavnější.

3.2.5 Shrnutí

Výše popsané tutoriály se zabývají programováním v Lua/LÖVE. Oficiální tutoriály jsou vhodné především pro řešení konkrétního problému nebo naučení určité problematiky. Tutoriál learn2love podrobně popisuje fungování jazyka Lua a frameworku LÖVE, témata na sebe navazují a jsou řazena dle obtížnosti od nejlehčího po nejtěžší. Je spíše vhodný pro začátečníky, kteří se chtějí naučit celý Lua/LÖVE. Na rozdíl od předchozích dvou tutoriálů kniha Developing Games on the Raspberry Pi vysvětluje programování v LÖVE z pohledu samotného programování hry. Čtenář se pomocí programování hry naučí, jak LÖVE funguje. Kniha obsahuje tutoriály na dvě hry, z toho jeden tutoriál se zabývá modulárním programováním. Poslední popsaný tutoriál Lua and LÖVE 11 funguje obdobně jako předchozí tutoriál. Učí studenty programovat konkrétní hry, je zaměřen na začátečníky a nabízí tutoriál celkem na jedenáct her. Tato práce je inspirována především tutoriálem Lua and LÖVE 11, protože tento tutoriál má dobrou strukturu, je přehledný a srozumitelný, tvorba her navíc motivuje studenty k učení. Pro potřeby této práce je práce také inspirována modulárním programováním, které je popsané v knize Developing Games on the Raspberry Pi.

Kapitola 4

Vizuální programování a jeho nástroje

Kapitola popisuje vizuální programování, jaké existují možnosti a na jakém principu fungují, viz sekce 4.1. Nejprve se kapitola zabývá programováním toku dat, především je ale zaměřena na blokové programování, protože je to relevantní pro tuto práci. V sekci 4.2 jsou popsány nástroje pro vizuální programování, nejedná se o výčet všech existujících nástrojů, ale pro účely této práce jsou vybrány nástroje, které mohou být inspirací pro vytvářenou webovou aplikaci.

4.1 Vizuální programování

Vizuální programovací jazyky umožňují uživateli programovat prostřednictvím manipulace grafických prvků, například přetahováním bloků na obrazovce, používáním vývojových diagramů nebo stavových diagramů, používáním ikon a netextových částí. Hodně vizuálních jazyků využívá i textové programování nebo kombinuje textové a vizuální programování [14]. Využití grafických prvků jednoduše umožňuje uživateli navrhovat a vytvářet aplikace. Využívá se toho často při výuce programování, protože aplikace zabraňuje tvorbě syntaktických chyb, a díky srozumitelnému uživatelskému rozhraní nejsou uživatelé zahlceni novými informacemi. Některé nástroje umí převádět vizuální reprezentaci do textové podoby kódu, tím se uživatel učí, jak má výsledek vypadat v textovém programovacím jazyce. Kuhail, M. A. et al. rozdělují vizuální programování do čtyř skupin: blokové programovací jazyky, diagramové jazyky, ikonové jazyky a formulářové jazyky. Tato práce je zaměřena především na blokové a diagramové vizuální jazyky [9].

Jelikož vizuální programování je zaměřeno hlavně na začátečníky, kteří nemají velké zkušenosti s programováním, je nutné, aby program sám popisoval, co dělá a jak funguje. Jednou z možností je používání přirozeného jazyka v programovacích jazycích. Tohoto využívá například blokové programování, které je vysvětleno v sekci 4.1.2. Programy zaměřené na úplné začátečníky by měly podle autora Alexandera Repenninga zahrnovat všechny tři následující oblasti, aby program byl srozumitelný [17]:

- **Syntaktická oblast** – Syntax je zachycena vizuálně, a to pomocí tvaru a barevného rozlišení. Bloky nejdou spojit, pokud to pravidla syntaxe nedovolují. [17]. Důvodem je, že začátečníci často dělají chyby v syntaxi, například chybějí závorky nebo středníky [10].

- **Sémantická oblast** – Sémantika programu je popsána v `help` funkcích, které popisují funkčnost jednotlivých bloků a jak mohou být použity. Jsou zde často také popsány další metody využití [17]. Reaguje se tím na situaci, kdy začátečníci nerozumí tomu, jak fungují funkce a operace jazyka [10].
- **Pragmatická oblast** – Pragmatická oblast popisuje konkrétní případ použití bloků. Vysvětluje konkrétně, co takto sestavené bloky znamenají. Pragmatické vysvětlení nepopisuje, jak blok obecně funguje, ale pro co byl uživatelem v dané situaci použit [17]. Reaguje na situaci, když začátečník neví, kdy a jak použít danou funkci [10].

4.1.1 Programování toku dat

Při programování toku dat je program reprezentován orientovaným grafem. Uzly grafu jsou základní instrukce jako aritmetické nebo porovnávací operátory. Orientované hrany zde reprezentují vazby mezi uzly grafu. Data putují grafem po hranách a tvoří frontu u uzlů, které je zpracovávají podle jejich pořadí příchodu. Hrany mířící k uzlu se nazývají vstupní hrany, hrany mířící od uzlu se nazývají výstupní. Při spuštění programu jsou data vložena na vstupní hrany, tímto je spuštěn zbytek programu. Když uzel obdrží data z předem daného počtu vstupních hran, stane se uzel spustitelný (fireable) a uzel se poté sám spustí (není ale specifikováno kdy). Uzel vezme data ze svých vstupů, provede operaci, dá nová data na výstup, poté přestane pracovat a čeká, až bude zase spustitelný. Díky tomuto principu můžou být instrukce vykonány, jakmile jsou dostupná data a uzel je spustitelný. To umožňuje paralelní vykonávání instrukcí [8]. Programování toku dat patří do kategorie diagramových vizuálních jazyků zmíněných výše 4.1.

4.1.2 Blokové programování

Blokové programování je součástí vizuálního programování, které v posledních letech nabývá na popularitě. Prostředí blokového programování funguje obdobně jako puzzle – jednotlivé dílky do sebe zapadají a tvoří výsledný program. Díky svému vzhledu a tvaru bloky uživateli napovídají, jak mají být použity a spojeny s ostatními. Samotné programování probíhá tak, že uživatel přesouvá bloky do programovací části prostředí a spojuje je dohromady, tím vytváří výsledný program [23]. Bloky reprezentují jak složitější struktury jako například metody nebo cykly, tak jednoduché operátory jako sčítání, odčítání nebo také proměnné [16]. Pokud by dva bloky porušily syntax jazyka, prostředí je neumožní spojit dohromady. Díky tomu se můžeme vyhnout syntaktickým chybám ve vytvořených programech. Uživateli napovídá, jak správně používat bloky, nejenom samotný tvar bloku, ale také barevné rozlišení a tvarově naznačená možnost vnoření dalších bloků (tvoření cyklů, podmínek) [23]. Na obrázku 4.1 můžeme vidět, jak tvar a barva bloku pomáhají při blokovém programování. Levý blok je ve tvaru písmene `C`, to uživateli napovídá, že může přidávat další bloky dovnitř tohoto bloku. Pravý blok je vytvořen pomocí dvou bloků: bloku `print` a bloku s textem `Hello World`. Oba dva mají stejnou barvu, protože spadají do stejné kategorie, v tomto případě práce s textem.

Nástroje pro blokové programování jsou součástí větší skupiny strukturovaných editorů, kde uzel je základní jednotka v abstraktním syntaktickém stromu (AST) reprezentujícím výsledný program. Uzly AST jsou v tomto případě bloky. V prostředí existují zábrany, aby bloky mohly být přidány do AST pouze správným způsobem, takto je předcházeno syntaktickým chybám [23].

Obrázek 4.1: Rozdíly barev a tvarů bloků pomáhají uživateli při programování.

Blokové programování se často využívá při výuce, studenti se tímto seznamují s principy programování. Tento typ programování je přístupnější a někdy i zajímavější pro začátečníky, kteří neznají standardní programovací jazyky. Většina nástrojů je založena na následujících společných principech:

- Nástroje jsou zaměřeny především na děti [16], například Scratch používají nejčastěji děti mezi 8 – 16 lety [18].
- Nástroje dále berou ohled na strukturu a syntax již existujících programovacích jazyků.
- Umožňují uživateli propojit různé typy projektů (hry, animace, hudbu) [16].

Existuje mnoho nástrojů, které využívají blokové programování, mezi nejznámější patří Scratch[1], Snap! [2] nebo Blockly [3]. Ty budou podrobněji popsány v následující části.

4.2 Nástroje

Pro vizuální programování existuje mnoho dostupných nástrojů. Tato práce se zaměřuje pouze na několik nástrojů, které jsou velmi využívané, a to konkrétně na DRAKON, Scratch, Snap!, AppInventor a Blockly. Jsou popsány a jsou zhodnoceny jejich výhody a nevýhody.

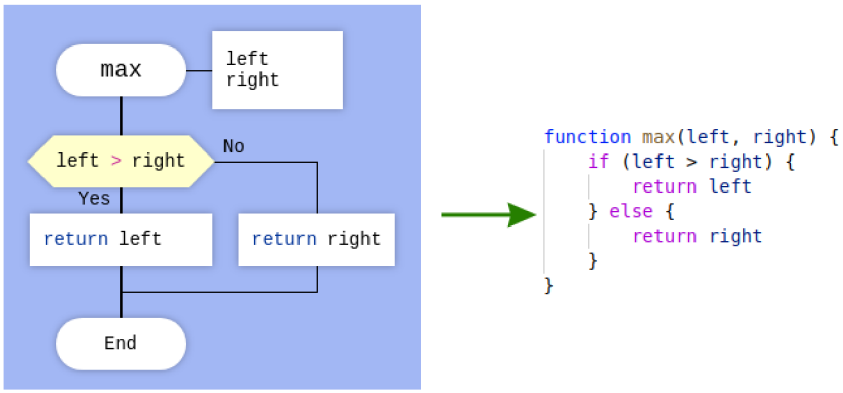
4.2.1 DRAKON

DRAKON je vizuální programovací jazyk, vyvinutý pro lepší znázornění a pochopení algoritmů a patří do sekce jazyků toku dat. Jeho syntax je založena na vývojových diagramech (flowcharts). DRAKON reprezentuje algoritmy pomocí vývojových diagramů, ty jsou přehledné a konzistentní díky striktním vizuálním pravidlům, které DRAKON dodržuje. Oproti textové reprezentaci kódu je ve vývojovém grafu na první podhled vidět další krok, algoritmu stačí pouze následovat hrany v grafu. Takto popsané algoritmy jsou pro uživatele velice intuitivní a dobře pochopitelné. DRAKON nevyužívá principu "drag and drop", jak je tomu u blokového programování, ale uživatel kliká na zvýrazněné body v grafu pro jeho úpravu nebo rozšíření. Vytvářený graf kreslí editor, ne uživatel. DRAKON obsahuje generátor kódu, který převede vytvořený vývojový diagram na kód jednoho z podporovaných jazyků (Java, D, C#, C/C++, Python, Lua a další) [2].

Na obrázku 4.2 je možné vlevo vidět kód psaný v jazyku DRAKON. Uživatel nejprve definuje strukturu pomocí vývojového diagramu, poté doplní pouze části kódu v JavaScript do diagramu. Nejsou potřeba klíčová slova jako if, else nebo cykly, protože toto je definováno

[1] https://scratch.mit.edu/
[2] https://snap.berkeley.edu/
[3] https://developers.google.com/blockly/

vývojovým diagramem. Výsledkem je pak vpravo vygenerovaný kód JavaScript, který je možné vidět na pravé straně obrázku 4.2.



Obrázek 4.2: Generování kódu v jazyce DRAKON [2]

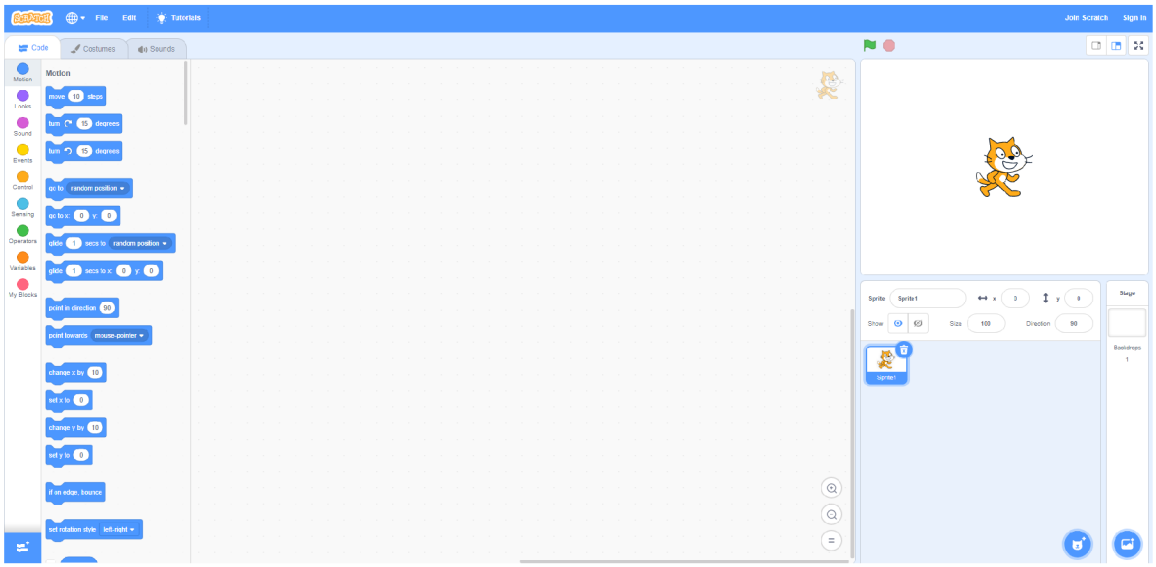
4.2.2 Scratch

Scratch je prostředí pro vizuální programování, ve kterém lze vytvářet různorodé projekty jako například: animované příběhy, hry, videa, tutoriály, simulace nebo hudební videa. Aplikace Scratch kombinuje dohromady programování společně s audiem a obrázky, proto je možné vytvářet různorodé projekty. Aplikace je založena na blokovém programování, které je vysvětleno v sekci výše 4.1.2. Základním cílem aplikace Scratch bylo představit programování lidem bez předchozích zkušeností. Aplikace je založena na samoučení, kdy se uživatel učí během tvorby samotného projektu nebo se také inspiruje z projektů jiných uživatelů. Aplikace obzvláště podporuje výpočetní myšlení a řešení problémů, tyto schopnosti lze přenést do normálního života. Scratch je zdarma a je dostupný v téměř 50 jazycích, používá ho zhruba 120 000 uživatelů, nejčastěji děti mezi 8 – 16 lety. Původně byl používán pro mimoškolní výuku v komunitních centrech a knihovnách, ale později se začal používat i při výuce ve školách. Kolem Scratch je aktivní široká komunita uživatelů, která se navzájem od sebe učí a podporuje při tvorbě projektů [12]. Scratch byl vytvořen na základě těchto principů:

- **More tinkerable** – Uživatelé mohou volně používat bloky pro tvorbu originálních projektů. Aplikace je interaktivní a vybízí ke kombinování různých prvků, jako je grafika, animace, hudba atd. Jakmile jsou bloky propojeny, aplikace okamžitě funguje a uživatel hned vidí výsledek. Samotné prostředí pro bloky má metaforicky fungovat jako pomyslný pracovní stůl, kde lze například odložit nepoužité bloky pro pozdější využití.
- **More meaningful** – Aplikace se snaží, aby projekty mohly být různorodé a osobní podle toho, o co mají uživatelé zájem a co je baví. Uživatelé mohou nahrávat vlastní hudbu, obrázky a vytvářet vlastní grafiku.
- **More social** – Aplikace podporuje využívání projektů ostatních členů, čímž se navzájem od sebe učí a podporují se při další tvorbě. Uživatelé mohou použít cizí projekt pro základ své aplikace [18].

Uživatelské rozhraní

Scratch má přehledné uživatelské rozhraní, které se dobře ovládá. Všechno je přehledně v jediném okně, klíčové části jsou vždy viditelné. Jak lze vidět na obrázku 4.3, uživatelské rozhraní je rozděleno do čtyř panelů. První panel obsahuje osm kategorií bloků, po rozkliknutí kategorie se objeví všechny dostupné. Kategorie jsou rozděleny podle funkčnosti a barevně rozlišeny. V druhém panelu uživatel spojuje jednotlivé bloky a vytváří program. Lze zde přepínat mezi záložkami s úpravou obrázků a hudby. Třetí okno ukazuje a spouští výsledný program. Čtvrté okno ukazuje jednotlivé objekty, které se zobrazují v třetím okně. Po kliknutí na objekt se na pracovní ploše zobrazí bloky popisující chování daného objektu.



Obrázek 4.3: Uživatelské rozhraní nástroje Scratch.

4.2.3 AppInventor

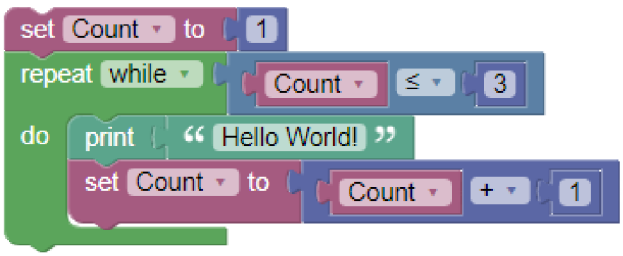
AppInventor je aplikace pro „drag and drop“ vizuální programování za účelem vytváření mobilních aplikací pro platformu Android. Aplikace funguje na principu blokového programování popsaného v sekci 4.1.2. Mobilní aplikace se vyvíjí ve dvou fázích: nejprve definuje jednotlivé komponenty uživatelského rozhraní v designeru komponent, například tlačítka, popisky atd. V druhé fázi se definuje chování aplikace pomocí blokového programování [22]. Při vývoji v AppInventor uživatelé mohou okamžitě vidět, jak použitý blok funguje v kontextu jejich aplikace. Toto umožňuje uživatelům inkrementální vývoj jejich aplikací a podporuje průběžné testování vytvořeného kódu. Bloky AppInventor podporují velkou řadu funkcí, například posílání SMS zpráv, přístup k GPS, Bluetooth nebo NFC. Tyto funkce by v textovém programování byly složitější na vytvoření, ale zde se jedná pouze o několik bloků, které obsahují celou funkcionalitu. Vytvořené aplikace mohou rovnou být staženy na připojený Android, exportovány ve formátu `.APK` nebo nahrány na Google Play. AppInventor je open-source a vývojové prostředí podporuje systémy Windows, Linux i Mac OS [15].

4.2.4 Snap!

Snap! je open-source nástroj pro blokové programování, které je popsané v sekci 4.1.2. Slouží pro vytváření animací, her a příběhů, funguje ve webovém prohlížeči. Snap! původně vycházel z nástroje Scratch (více o Scratch je vysvětleno v sekci 4.2.2), verze 4.0 je již ale napsaná v JavaScript. Například uživatelské rozhraní je podobné jako v nástroji Scratch, samotný Snap! obsahuje ale několik funkcí navíc. Snap! se nezaměřuje pouze na začátečníky, ale také na pokročilejší uživatele. Mezi pokročilé funkce patří například lambda kalkul nebo vnořené seznamy. Vytvořené programy lze převést na programovací jazyky Python, JavaScript nebo C. Vytvořené projekty mohou být uloženy lokálně, do Snap! cloudového úložiště nebo exportovány ve formátu XML (soubor obsahuje popis všech použitých bloků, použité obrázky a zvuky) [22].

4.2.5 Blockly

Blockly je open-source knihovna, pomocí které lze vytvářet aplikace pro blokové programování. Blockly sám o sobě není jazykem a není určen pro koncové uživatele. Developeři však mohou využít Blockly pro vytvoření svého vlastního jazyka. Blockly obsahuje uživatelské rozhraní s blokovým editorem a framework pro generování kódu v klasických programovacích jazycích. Blockly podporuje generování kódu pro jazyky: JavaScript, Lua, PHP, Dart, Python. Podporuje také přidání generátorů pro další jazyky. Je nutné, aby developeři definovali, jak a kam se má kód generovat a jak bude výsledný kód spuštěn. Knihovna je psaná v JavaScript, takže je jednoduché ji přidat jako část webové stránky. Jsou také dostupné verze pro Android a iOS [14]. Je dobré mít na paměti, že výsledné aplikace vytvořené v Blockly se mohou od sebe velice lišit, záleží na přístupu developerů k vývoji aplikace. Blok vytvořený v jedné aplikaci tím pádem pravděpodobně nebude fungovat v jiné. Na obrázku 4.4 je příklad jednoduchého programu vytvořeného pomocí Blockly. Bloky jsou rozlišeny barvou a tvarem podle jejich funkčnosti, jak bylo již popsáno v sekci blokové programování 4.1.2. Obrázek ukazuje, že k Blockly je možné přistupovat tak, že navrhované bloky kopírují strukturu existujícího programovacího jazyka, což ulehčuje uživatelům přechod k budoucímu programování v textové formě [14].



Obrázek 4.4: Ukázka blokového programování.

Doporučení pro práci s Blockly

Existuje několik způsobů, jak je možné popisovat bloky. Jedním z nich je použití ikon, které ale může být nevýhodné u abstraktních pojmů. Příkladem je použití podmínek bez textu, které může být těžko pochopitelné. I když jsou ikony jednoduché, příliš velké množství ikon může být pro uživatele těžko zapamatovatelné a matoucí. Obvykle se proto doporučuje používat malé množství jednoduchých bloků [14].

Další možností pro popisování bloků je použití přirozeného jazyka, jak lze vidět v obrázku 4.4, kde bloky obsahují text psaný v přirozeném jazyku, například v angličtině. Často se používají celé věty pro popis komplexních problémů, bloky s textem jsou pro uživatele lépe pochopitelné než samotné ikony. Tento přístup je srozumitelnější a pro mnoho uživatelů intuitivnější než samotné používání kódu. Je potřeba si uvědomit, že použitá slova a gramatika výrazně ovlivňují správné pochopení bloků uživatelem, doporučuje se proto nepoužívat nespisovná slova a slang. Na druhou stranu pokud je ale účelem seznámit uživatele s programovacím jazykem, je naopak vhodné použít technické termíny daného programovacího jazyka [14].

Dále se doporučuje ke každému bloku vytvořit jednoduchou dokumentaci s jeho popisem, fungováním a případy užití. Je důležité zahrnout i obrázek samotného bloku. To je jednoduché, jelikož Blockly podporuje režim pouze pro čtení, který zobrazí blok ve vybraném jazyce. Tímto způsobem je jednoduché překládat aplikaci do dalších jazyků [6].

Blokové programování často využívá výchozí hodnoty, aby se předešlo chybám, kdyby hodnota nebyla zadána. Bloky s výchozí hodnotou napovídají uživateli, jaká hodnota se očekává. Uživatel může výchozí hodnoty měnit tak, že je nahradí jiným blokem a pokud tento blok smaže, objeví se znovu výchozí hodnota [14].

Autoři Blockly pro srozumitelnost doporučují, aby vytvořená aplikace nepodporovala přístup ke všem funkcím programovacího jazyka. Aplikace by měla používat jednoduché bloky v menším množství, protože by jinak mohla být pro uživatele příliš komplikovaná. Příkladem jsou části jazyka, které jsou komplikované nebo problémové i pro zkušené programátory. Pokud by chtěl developer využívat větší množství bloků, dobrým příkladem je aplikace AppInventor, viz sekce 4.2.3. AppInventor dává uživateli možnost vyhledat bloky v rámci aplikace [14].

Díky funkčnosti blokového programování, které je vysvětleno zde 4.1.2, se uživatel vyvaruje syntaktických chyb. Mohou ale nastávat situace, kdy bloky vypadají spojeny, i když tomu tak není, a program potom nefunguje. Druhým problémem je, že Blockly umožňuje pro lepší ovládání pohybovat se skupinami již propojených bloků a propojit je s dalším blokem. V praxi to ale může vést k problémům. Pokud je uživatel nepozorný, může omylem spojit bloky, které nechtěl [6].

Kapitola 5

Návrh

Kapitola se zabývá specifikací cílů práce, viz 5.1, jsou zde popsány funkční požadavky navrhované aplikace. Jsou shrnuty a porovnány existující řešení a návrh se zaměřuje na prvky, které mohou být při návrhu užitečné. Sekce 5.2 se zabývá návrhem Android aplikace a komunikací s webovou aplikací pomocí REST API a Firebase Cloud Messaging. V další části kapitoly 5.3 je navrhnuta webová aplikace, především se zabývá architekturou aplikace, editory kódu, překladem vytvořeného kódu, návrhem databáze, případy užití aplikace a jednoduchým REST API. Následně je uvedena podoba uživatelského rozhraní 5.5. Na závěr je popsána struktura a témata tutoriálů, které bude aplikace obsahovat 5.6.

5.1 Specifikace požadavků

Cílem práce je vytvořit webovou aplikaci pro vizuální programování v `Lua/LÖVE`. Aplikace bude podporovat vizuální i textové programování. Uživatel bude mít možnost přepínat mezi blokovou a textovou reprezentací kódu. Aplikace by měla sloužit pro výuku programování jak pro začátečníky (studenti základních škol), tak i pro pokročilejší (studenti středních škol). Především se bude zaměřovat na výuku pomocí blokového programování, bude fungovat ve webovém prohlížeči a bude v českém i anglickém jazyce. Vytvořené hry budou spustitelné na zařízeních Android, přímo z webové aplikace. Toho je docíleno tím, že je vytvořena klientská aplikace na Android, která bude zajišťovat stažení a instalaci vytvořených her. Webová aplikace bude také obsahovat několik tutoriálů, jak si vytvořit vlastní hru v `Lua/LÖVE`. Tutoriály budou v různých stupních obtížnosti.

5.1.1 Funkční požadavky

Ze specifikace vychází několik funkčních požadavků, které by měla navrhovaná aplikace splňovat. Požadavky byly rozděleny do tří skupin: webová aplikace, blokové programování a výukové prvky. Toto rozdělení reflektuje cíle této práce.

Webová aplikace

Bude vytvořena webová aplikace pro blokové programování v `Lua/LÖVE`. Aplikace bude obsahovat menu kategorií, z nichž každá bude obsahovat bloky, které uživatel může využít ve svém projektu. Bloky se přetáhnou na pracovní plochu, kde se spojují dohromady. Uživatel bude moci v průběhu práce na projektu spustit vytvořené hry v prohlížeči. Aplikace může fungovat v režimu hosta nebo přihlášeného uživatele. V režimu hosta nebudou

projekty ukládány, aplikace bude v tomto případě sloužit pouze k vyzkoušení blokového programování. Přihlášený uživatel bude mít navíc možnost vytvářet projekty, které se budou ukládat do databáze. Uživatel může sdílet projekty s jinými uživateli a nahrát vlastní obrázky a zvuky, které poté může použít ve vytvářených hrách. Aplikace bude umožňovat exportovat kód nebo nahrát hru přímo na zařízení Android. Toho bude docíleno za pomoci klientské Android aplikace.

Blokové programování

Bloky budou rozděleny do kategorií a barevně rozlišeny, jejich tvar bude naznačovat jejich funkcionalitu (například cykly ve tvaru písmene C – mohou se přidávat bloky dovnitř). Aplikace zabrání tomu, aby se spojily bloky, které nejsou syntakticky správně, což ulehčí práci uživateli. Všechny bloky budou mít dokumentaci, jak fungují a jak mají být použity. Pokud by uživatel chtěl přejít z blokové na textovou verzi programování, dokumentace bude obsahovat odkazy na dokumentaci originálních funkcí `Lua/LÖVE`. Pro pokročilé uživatele zde bude možnost vytváření vlastních bloků. Spojením několika již existujících bloků lze vytvořit jeden nový blok, který bude mít stejnou funkčnost jako původní složené bloky.

Výukové prvky

Součástí webové aplikace budou tutoriály pro tvorbu her v `Lua/LÖVE`. Každý tutoriál bude obsahovat popis vytvářené hry a její pravidla. U každého tutoriálu bude možné se podívat na již vytvořenou hru a její kód, jak v blokové, tak v textové podobě. Tutoriály budou strukturovány do logických sekcí, každá bude obsahovat ukázky použitých bloků a textový popis, co daný blok dělá a proč byl použit.

5.1.2 Existující aplikace

V předchozí kapitole bylo uvedeno několik vybraných nástrojů pro blokové programování 4.2. Pro návrh aplikace byly použity některé prvky popsaných nástrojů, které jsou vhodné pro tuto práci. Inspirací pro uživatelské rozhraní byl nástroj Scratch, protože je zde uživatelské rozhraní přehledné a jednoduché na pochopení. Scratch využívá mnoho uživatelů, proto je jejich uživatelské rozhraní dobře rozvrženo a nový uživatel se v něm jednoduše zorientuje. V horní části obrazovky se nachází lišta se základním ovládáním aplikace (přihlášení, změna jazyka, tutoriály). Dále je praktická část, kde si uživatel může spustit kód a vidí tak hned výsledek své práce. Scratch má také přehledné rozdělení bloků do kategorií dle funkcionality. Struktura menu je inspirována nástrojem AppInventor, který pracuje s velkým množstvím bloků. Navrhovaná aplikace bude mít také větší počet bloků, které bude potřeba kategorizovat. AppInventor umožňuje vyhledávání bloků a tahle funkcionalita by měla být také zakomponována. V aplikaci budou některé bloky netriviální, například modul fyzika v `Lua/LÖVE` má některé složité funkce. Zde se aplikace inspiruje nástrojem Snap!, jehož bloky řeší pokročilé funkce, příkladem je lambda kalkul. K práci s bloky bude využita knihovna Blockly, která mimo jiné umožňuje převod bloků na kód jazyka `Lua`. Je také potřeba, aby obdobně jako v nástroji Scratch existovala možnost sdílet projekt s jiným uživatelem.

5.2 Návrh Android aplikace

Android aplikace slouží pro jednoduché nahrávaní a spouštění her vytvořených ve webové aplikaci na platformu Android. Uživatel si bude muset stáhnout aplikaci a přihlásit se pomocí účtu vytvořeného ve webové aplikaci. Aplikace bude pak sloužit pro tyto účely:

- zobrazení všech projektů uživatele
- stažení projektu a instalaci na zařízení
- smazání starých projektů ze zařízení Android

Cílem aplikace je zjednodušení procesu nahrávání a spouštění vytvořených her. Při prvním použití bude muset uživatel povolit některá oprávnění. Při dalším používání by již měla být aplikace nastavena a uživatel nebude muset nic potvrzovat. Oprávnění:

- `ACCESS_NETWORK_STATE` – Povoluje aplikaci získat informace o síti.
- `INTERNET` – Povoluje aplikaci otevírat sokety.
- `WRITE_EXTERNAL_STORAGE` – Povoluje aplikaci zapisovat do externího úložiště.
- `REQUEST_INSTALL_PACKAGES` – Povoluje aplikaci instalovat balíčky.

5.2.1 Kotlin

Pro vývoj aplikace bude použit programovací jazyk Kotlin, který je alternativou k jazyku Java. Android Studio umožňuje uživatelům automatický převod kódu z jazyka Java do jazyka Kotlin. Hlavní výhody oproti jazyku Java jsou [11]:

- **Více expresivní** – Je potřeba méně kódu pro stejnou funkcionalitu.
- **Bezpečnost** – Kotlin řeší problémy s potenciálními `null` hodnotami už při kompilaci, tím se předchází chybám za běhu kódu. Je nutné specifikovat, zda objekt může mít hodnotu `null` a poté tuto hodnotu kontrolovat než bude použita.
- **Funkcionální programování** – Využívá koncepty funkcionálního programování, například lambda výrazy.
- **Rozšiřitelnost** – Umožňuje rozšířit třídy o nové funkce. Je možné rozšířit i třídy, u kterých není přístup ke zdrojovému kódu.
- **Spolupráce s Javou** – Umožňuje využívat knihovny napsané v jazyce Java.

Kotlin bude použit, jelikož je doporučován společností Google jako primární jazyk pro vývoj Android aplikací a výhodou oproti jazyku Java je, že pro stejnou funkcionalitu je potřeba méně řádků kódu.

5.2.2 Komunikace s webovou aplikací

Pro autentizaci bude v rámci aplikace využit Firebase Auth. Tato knihovna od Firebase umožňuje autentizaci přes Firebase Tokeny, které lze generovat ve webové aplikaci a tím autentizovat uživatele mobilní aplikace. Pro uživatelské rozhraní je v plánu využít knihovnu Firebase UI, která slouží k automatickému vytvoření uživatelského rozhraní, například pro login, registraci nebo reset hesla.

Po přihlášení uživatel může stáhnout projekty přímo z Android aplikace. Stahování projektu probíhá tak, že v aplikaci vybere projekt, který bude po zvolení automaticky stažen, nainstalován a spuštěn. První možností je, že komunikace s webovou aplikací bude probíhat pomocí REST API, který je popsán zde 5.3.

Druhou možností komunikace mezi webovou a Android aplikací je tlačítko `Spustit hru na Android`, které se bude nacházet v menu webové aplikace. Po stisknutí tlačítka se odešle zpráva Android zařízení, který projekt má být stažen a nainstalován. Komunikace bude fungovat přes Firebase, který v bezplatném plánu obsahuje Firebase Cloud Messaging (FCM)[1]. FCM slouží k posílání zpráv mezi různými platformami, umí rozesílat zprávy jak na skupiny zařízení, tak na individuální zařízení identifikované unikátním tokenem. V tomto případě využiji tuto druhou možnost komunikace. FCM vyžaduje verzi Android 4.4 nebo vyšší, nainstalovanou aplikaci Google Play Store a také Google Play Services. Bez nainstalované aplikace Google Play Services nebude FCM fungovat, protože ji využívá pro zasílání zpráv. Dle informací od Firebase by 95 % zpráv mělo dorazit na cílové zařízení v průměru za 250ms [5], to je pro tento případ ideální, jelikož je potřeba, aby komunikace byla okamžitá a uživatel nemusel na nic čekat. V následujícím kódu je zobrazen příklad FCM zprávy, ta obsahuje pole `token` pro identifikaci zařízení a pole `data` s daty určenými pro aplikaci. V tomto případě se jedná o pole `project` a pole `url`. Po získání FCM zprávy se stáhne `.APK` soubor pomocí REST API, který je popsán zde 5.3.

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1",
    "data":{
      "project" : "example_project",
      "url" : "http://url_adresa_ke_stazeni_projektu/",
    }
  }
}
```

5.3 Návrh webové aplikace

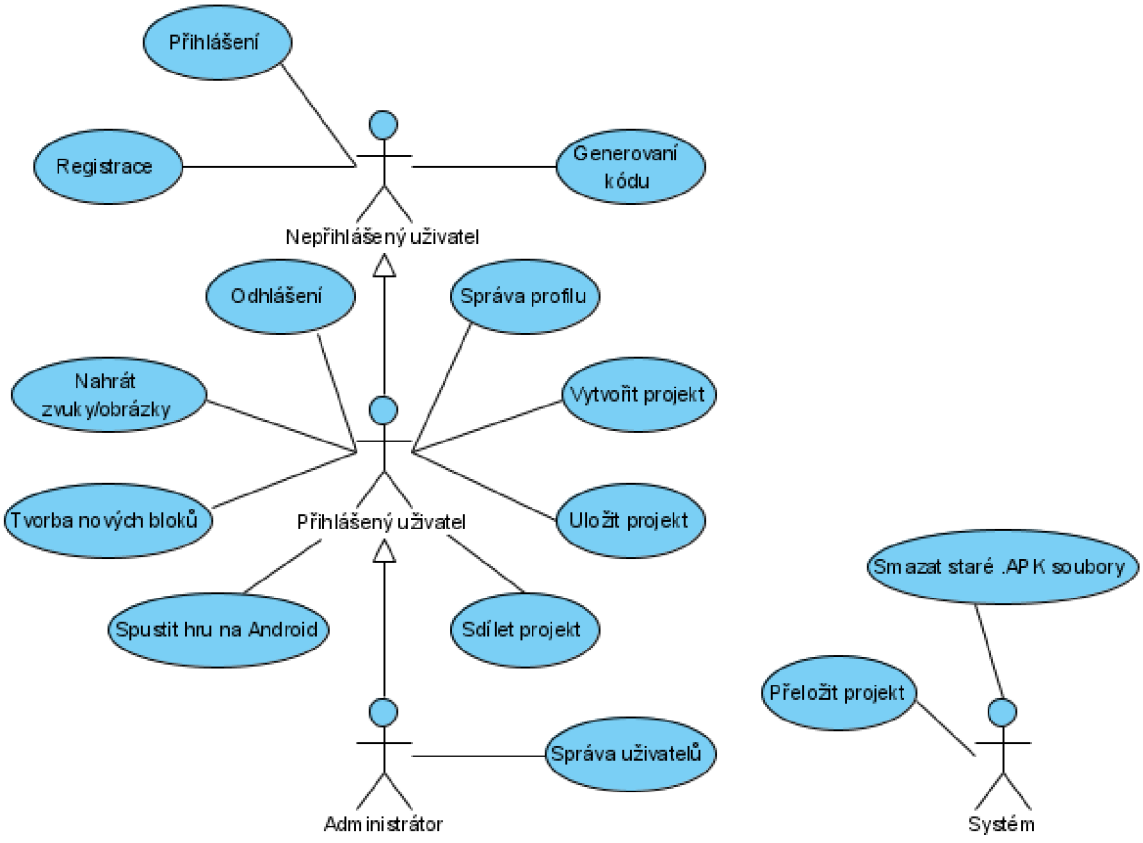
Webová aplikace slouží pro vytváření her v `Lua/LÖVE` pomocí blokového nebo textového programování. Obsahuje tutoriály, pomocí kterých se může uživatel naučit programovat konkrétní hry. Pro tvorbu webové aplikace bude použita architektura Model-View-Controller, protože ji využívá PHP framework Laravel, který je vysvětlen v sekci 7.1. Tato architektura se skládá ze tří vrstev: Model (datová vrstva), View (prezentační vrstva) a Controller (řídící vrstva). Architektura odděluje práci s daty, řízení aplikace a její vzhled. Toto rozdělení pomáhá k dobré struktuře webové aplikace a umožňuje efektivní vývoj webových aplikací.

[1] https://firebase.google.com/docs/cloud-messaging

- **Model** – Vrstva je zodpovědná za správu dat. Model je připojen k databázi a stará se o přidávání, odebírání a úpravu dat. Model vykonává požadavky vrstvy Controller a předává jí informace z databáze, jelikož vrstva Controller nemůže komunikovat přímo s databází. Model také nikdy přímo nekomunikuje s vrstvou View.
- **View** – Vrstva má na starosti zobrazení dat. Generuje uživatelské rozhraní, ve webových aplikacích k tomu nejčastěji využívá HTML a CSS. Data získává skrze vrstvu Controller, se kterou jako jedinou komunikuje.
- **Controller** – Tato vrstva spojuje vrstvy Model a View. Controller dává požadavky vrstvě Model, získaná data zpracuje a pošle vrstvě View pro zobrazení [20].

Diagram případů užití

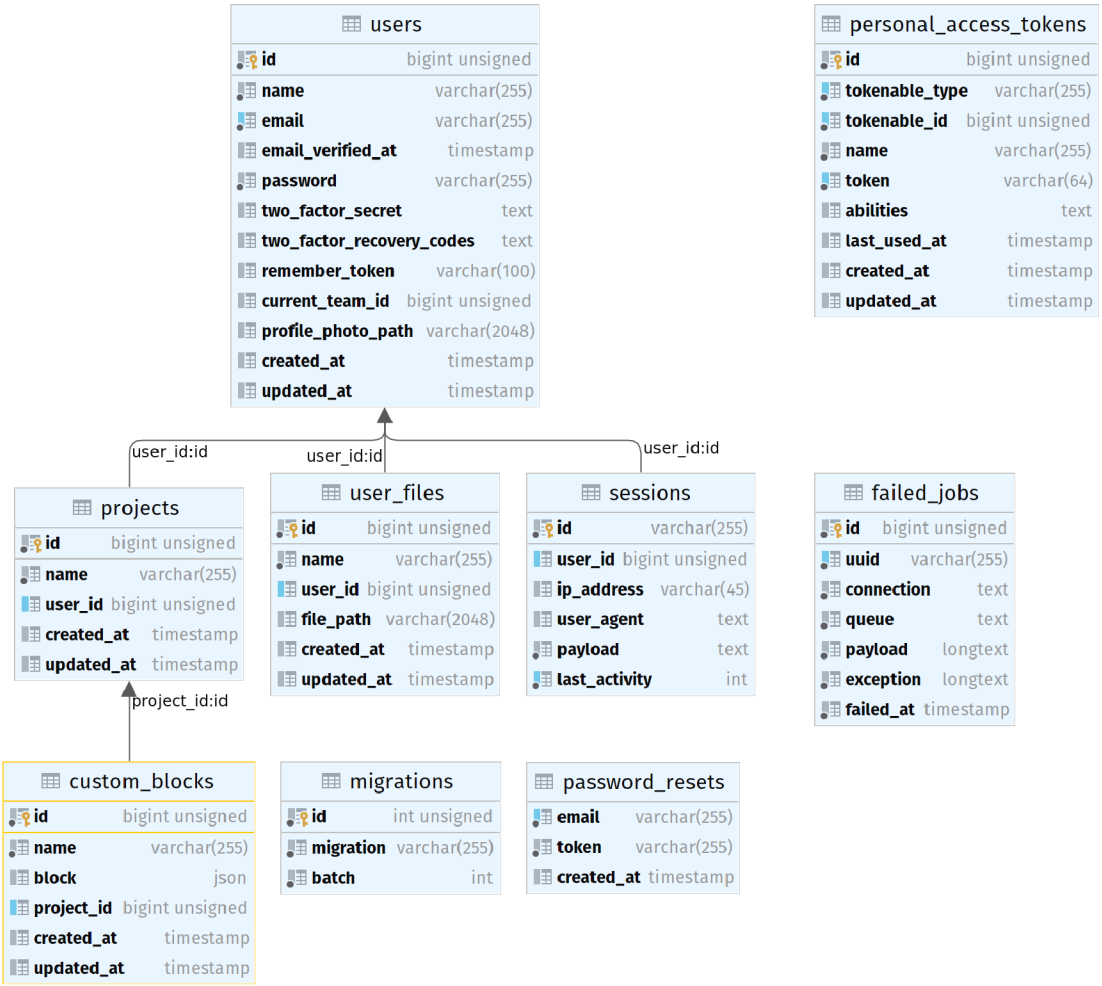
Aplikace bude obsahovat tři uživatelské role: administrátor, přihlášený uživatel a host. Host bude mít přístup k aplikaci, ale nebude schopen ukládat projekty a nahrávat vlastní soubory. Přihlášený uživatel bude mít přístup ke všem funkcím aplikace (ukládaní a sdílení projektů, nahrávání vlastních obrázků atd.). Administrátor může navíc spravovat jednotlivé uživatele. Posledním aktérem je systém, který se stará o mazání souborů a překlad projektů na `.APK` soubory. Následující obrázek popisuje, co mohou uživatelé v systému dělat 5.1.



Obrázek 5.1: Diagram případu užití pro webovou aplikaci

Návrh databáze

Databáze slouží jako perzistentní úložiště dat. Data jsou v aplikaci uložena podle následujícího schématu 5.2. Databáze má několik tabulek, například `users`, `projects`, `user_files`, `cusom_blocks`, které obsahují informace o uživatelích, dále uživatelem vytvořené projekty, nahrané obrázky a zvuky a jako poslední jsou vlastní bloky. Tabulka `Migrations` pochází z frameworku Laravel, jedná se o jednoduché verzování schématu databáze. Tabulka `Failed jobs` obsahuje `Laravel Jobs`, které systém nebyl schopen vykonat. `Laravel Job` bude v projektu využíván pro překlad vytvořené hry, viz 5.3. Tabulka `password_reset` má na starosti resetování hesel a tabulka `sessions` má ukládat informace o jednotlivých sezeních, ve kterých je uživatel aktivní.



Obrázek 5.2: Schéma databáze

Language Server Protocol (LSP)

Součástí webové aplikace je možnost textového programování. Aby byl editor vhodný pro programování, měl by obsahovat základní funkce jako zvýraznění syntaxe, automatické doplňování, dokumentaci a další. Aby tyto funkce byly součástí editoru, bude využit Language

Server Protocol (LSP), který zajišťuje komunikaci mezi editorem a jazykovým serverem (v tomto případě `Lua`). Jazykový server se stará o funkce popsané výše, pro jazyk `Lua` existuje několik takovýchto serverů, pro tento projekt bude využit Lua Language Server[2].

Editory kódu

Tato aplikace musí podporovat jak práci s kódem ve formátu bloku, tak ve formátu textu. Toho bude docíleno tak, že budou využity dva různé editory kódu. Pro práci s bloky je v plánu využít Blockly, který je vysvětlen zde 4.2.5. Blockly je jasná volba, protože je to kompletní knihovna pro práci s bloky a neexistuje žádná alternativa, která by nabízela stejnou funkčnost. Blockly využívá řada existujících aplikací pro práci s bloky, například Scratch, viz 4.2.2. Naopak pro textový editor existuje řada nástrojů, které je vhodné použít. Pro tuto práci bude využit Monaco editor[3], který je vyvíjen společností Microsoft a je to populární webový editor kódu. Monaco editor je dostupný jako balíček npm, což slibuje jednoduchou instalaci. Druhým důvodem pro výběr tohoto editoru je, že díky Monaco Language Client[4] je možné propojit Monaco editor pomocí LSP s Language serverem. Převod kódu bude možný ale pouze z blokového editoru do textového editoru, obráceně to není možné, jelikož z textového formátu kódu nelze jednoduše generovat bloky a vyžadovalo by to komplikované řešení.

Spuštění her v prohlížeči

Uživatelem vytvořené hry by měly být spustitelné přímo z prohlížeče. K tomu bude využit existující projekt `love.js`[5], který umožňuje spustit `Lua/LÖVE` ve webovém prohlížeči. Pro vytvoření spustitelné hry `love.js` potřebuje být projekt ve formátu `.love`, což je pouze přejmenovaný `.zip` soubor.

Překlad kódu

Pro spouštění her na platformě Android je nutné vytvořený kód přeložit, o to se bude starat webová aplikace. Když uživatel klikne na tlačítko `Spustit aplikaci na Android`, vytvoří se nový `Laravel Job`, který přeloží kód a vytvoří `.APK` soubor. Ten si pak může uživatel stáhnout a spustit pomocí klientské Android aplikace. Soubory `.APK` budou na serveru uloženy v souborovém systému, kde budou po jednom dni automaticky mazány.

REST API

Webová aplikace bude obsahovat jednoduché REST API pro komunikaci s Android aplikací. API bude přístupné pouze přihlášeným uživatelům (o tuto funkcionalitu se postará Laravel).

- `POST /login` – přihlásí uživatele
- `POST /register` – registruje uživatele
- `POST /token` – odešle FCM token serveru
- `POST /projects` – vrátí seznam všech projektů uživatele

[2] https://github.com/sumneko/lua-language-server
[3] https://microsoft.github.io/monaco-editor/
[4] https://github.com/TypeFox/monaco-languageclient
[5] https://github.com/Davidobot/love.js

- `POST /project/love` – přeloží projekt na `.love` soubor a vrátí url na stažení tohoto souboru

Vícejazyčnost

Navržená aplikace bude podporovat anglický a český jazyk. Framework Laravel, který bude pro implementaci webové aplikace použit, umožňuje vícejazyčnost. Překlady textu a uživatelského rozhraní jsou uloženy v `.json` souborech v adresáři `resources/lang`. Jazyk je vybrán podle preferencí uživatele v nastavení aplikace. Obdobně knihovna Blockly podporuje vícejazyčnost, jak je uvedeno v sekci 4.2.5. Jazyk aplikace lze vidět jednoduše v URL odkazu: `https://example.com/en/home`.

5.4 Návrh tvorby bloků

Blokové programování si lze představit jako puzzle – spojením jednotlivých bloků vzniká výsledný program, jak je vysvětleno v sekci 4.1.2. Bloky mohou reprezentovat složité i jednoduché struktury a jsou tvarově a barevně rozlišeny podle své funkčnosti. V tomto návrhu se vychází z architektury `Lua/LÖVE` popsané v sekci 2.2. Je brána v potaz aktuální verze `LÖVE` 11.4, na starší verze se nebere ohled. Vychází se především z callback funkcí. Pro každou callback funkci[6] je potřeba navrhnout vlastní blok. Je navrženo, aby tyto bloky měly tvar písmene `C`, tento tvar se obvykle využívá pro funkce a cykly a naznačuje uživateli možnost vnoření dalších bloků. Framework `LÖVE` má 37 těchto funkcí.

Framework `LÖVE` má řadu modulů uvedených zde 2.2. Každý modul bude reprezentován jako kategorie bloků, každá funkce modulu bude znázorněna jako blok. Jednotlivé kategorie budou barevně odlišeny. Moduly obsahují řadu funkcí, které mají na starosti podobné věci. Například v modulu `love.graphics()` existují funkce na elipsu, kruh a polygon. Pro tyto funkce není potřeba vytvářet tři různé bloky, ale bude vytvořen pouze jeden blok s možností výběru, jaký objekt má být vykreslen. Tímto bude snížen počet bloků a pro uživatele bude používání aplikace přehlednější.

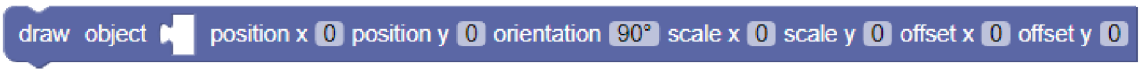
5.4.1 Tvorba bloků

Pro tvorbu bloků bude využita knihovna Blockly, blíže popsaná zde 4.2.5. Bloky lze vytvářet pomocí `.json` dokumentů, které definují tvar, barvu a funkci. Pro vytvoření bloků lze také využít grafického rozhraní ve vývojářských nástrojích Blockly[7]. Zde je možné sestavit blok pomocí blokového programování, vytvořený blok se exportuje do `.json` dokumentu a ten lze použít ve vytvářené aplikaci. Je zde také náhled, jak bude vypadat vytvořený blok a šablona v jazyce JavaScript pro generování kódu daného bloku. Tvorba bloku je ilustrována na příkladu funkce:

```
love.graphics.draw(drawable, x, y, r, sx, sy, ox, oy)
```

Ta vykreslí objekt na obrazovku, má parametry pozice x/y, rotace, škálovatelnost x/y a odsazení x/y. Pro tuto funkci byl vytvořen blok, který je zobrazen na obrázku 5.3. Blok má několik proměnných, většina je číselná, jediná rotace je ve stupních. Na pozici objekt lze vnořit další blok, který musí být typu `drawable`. Tvar bloku napovídá, že lze přidat bloky shora i zespodu.

[6] https://love2d.org/wiki/love
[7] https://blockly-demo.appspot.com/static/demos/blockfactory/index.html



Obrázek 5.3: Block Draw

V `.json` reprezentaci bloku je nutné zadat parametr `type`, který určuje typ bloku, a záleží na vývojáři, jak ho pojmenuje. Typ musí být unikátní mezi všemi bloky. Dále `.json` obsahuje pole argumentů, ve kterém se nachází výčet argumentů daného bloku. Argumenty obsahují pole `type` říkající, jaký typ vstupních dat přijímá. Pole `name` určuje unikátní jméno argumentu (to se pak využívá v generování kódu). Pole `check` se vyskytuje pouze tehdy, když lze vnořit další bloky, a kontroluje správný typ bloku, aby nedocházelo k syntaktickým chybám. V tomto příkladě se jedná o pozici `object`. Pole `value` obsahuje výchozí hodnotu. Na konci `.json` dokumentu se nachází parametr `inputsInline`, který definuje, že se argumenty zobrazí v řadě za sebou. Parametry `previousStatement` a `nextStatement` definují, jaký typ bloku lze připojit shora a zdola (v tomto případě je to bez omezení). Parametr `colour` určuje barvu bloku a parametr `tooltip` slouží jako nápověda, jak blok využít. Blok, který je možné vidět na obrázku 5.3, vypadá v `.json` následovně:

```
"type": "love_graphics_draw",
"args0": [
{
  "type": "input_value",
  "name": "object",
  "check": "drawable",
  "align": "CENTRE"
},
{
  "type": "field_number",
  "name": "position x",
  "value": 0
},
],
"inputsInline": true,
"previousStatement": null,
"nextStatement": null,
"colour": 230,
"tooltip": "Draws a Drawable object on the screen
            with optional rotation and scaling.",
```

5.4.2 Generování kódu

Pro generování kódu knihovna Blockly nabízí tři základní metody:

- `getFieldValue()` – vrátí hodnotu zadaného argumentu
- `valueToCode()` – vygeneruje kód vnořeného bloku (v tomto příkladu pole `object`)
- `statementToCode()` – vygeneruje kód segmentu vnořených bloků

Knihovna Blockly je napsaná v jazyce JavaScript, ale lze vytvořit generátory kódu pro jakýkoliv jazyk. V tomto případě se jedná o jazyk `Lua`. Ke každému bloku je nutné vytvořit generátor kódu pro cílový jazyk. Vývojářské nástroje Blockly zobrazují šablonu pro generování kódu daného bloku. Šablona pro generování využívá metody popsané výše, vývojář musí pouze doplnit pomocí cílového jazyka, co má blok dělat. Využívá k tomu proměnné, které definoval při tvorbě bloku. Následující kód zobrazuje blok z obrázku 5.3. Proměnná `code` obsahuje generovaný kód v jazyce `Lua`.

```
Blockly.Lua['love_graphics_draw'] = function(block) {
  var value_object = Blockly.Lua.valueToCode(block,
                     'object', Blockly.Lua.ORDER_ATOMIC);
  var number_position_x = block.getFieldValue('position x');
  var number_position_y = block.getFieldValue('position y');
  var angle_name = block.getFieldValue('NAME');
  var number_scale_x = block.getFieldValue('scale x');
  var number_scale_y = block.getFieldValue('scale y');
  var number_offset_x = block.getFieldValue('offset x');
  var number_offset_x = block.getFieldValue('offset y');
  var code = 'love.graphics.draw(' + value_object + ','
             + number_position_x + ',' + number_position_y + ','
             + angle_name + ',' + number_scale_x + ','
             + number_scale_y + ',' + number_offset_x + ','
             + number_offset_x + ',' + ')';
  return code;
};
```

5.5 Návrh uživatelského rozhraní

Dobrý návrh uživatelského rozhraní určuje, zda bude pro uživatele jednoduché aplikaci používat. Jednou z hlavních inspirací je nástroj Scratch. V sekci 5.1.2 bylo popsáno, které prvky jsou považovány za důležité a kterými se lze inspirovat při návrhu aplikace.

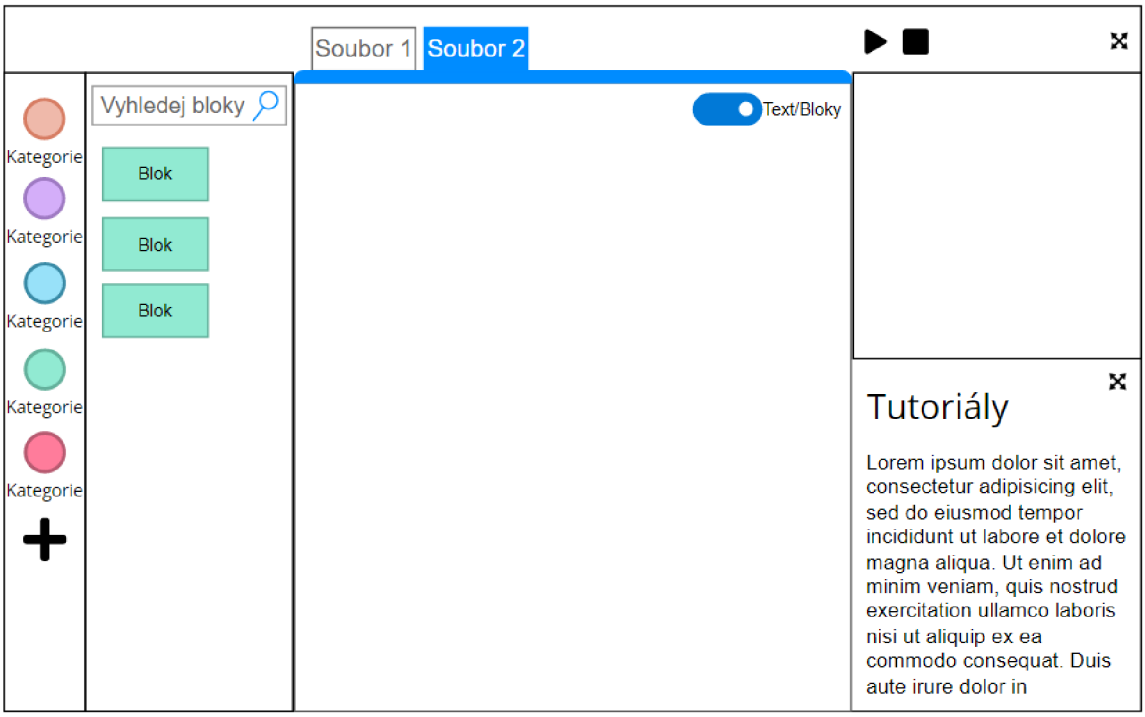
Uvítací stránka

Uvítací stránka bude velmi jednoduchá, budou zde základní informace o aplikaci. Dále zde bude možnost přihlášení nebo přístup bez přihlášení (používání aplikace jako host). Tlačítkem dokumentace se uživatel dostane ke stránce s popisem jednotlivých bloků. Tlačítkem tutoriály může uživatel přejít na tutoriály vytvořené speciálně pro tuto aplikaci. Dále bude uvítací stránka obsahovat odkazy na oficiální stránky `Lua/LÖVE` a další důležité zdroje.

Hlavní aplikace

Samotná webová aplikace bude rozdělena na horní lištu s ovládáním aplikace, levé boční menu s bloky, pracovní plochu uprostřed a pravé okno pro zobrazení vytvořené hry. Pracovní plocha bude mít dvě možnosti zobrazení: blokové nebo textové, uživatel tak může překliknout mezi oběma reprezentacemi kódu. Aplikace umožní rozdělit projekt do více souborů (vhodné pro velké projekty). Soubory se zobrazí v záložkách nad pracovní plochou. Obsah pracovní plochy se bude měnit podle toho, v jakém souboru uživatel pracuje. Velikost pravé části s oknem hry a s tutoriály bude nastavitelná, uživatel může určit, kolik místa pracovní

plochy budou okna zabírat. Pokud bude uživatel pracovat s tutoriálem, zobrazí se vpravo dole. Toto rozdělení lze vidět na následujícím obrázku 5.4.



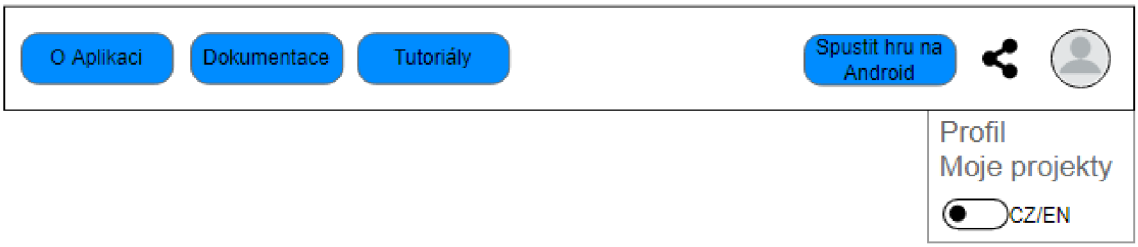
Obrázek 5.4: Mockup rozložení aplikace

Horní lišta

Lišta v horní části obrazovky bude obsahovat základní ovládání aplikace:

- **Přihlášení/odhlášení** – Zobrazí dialogové okno pro přihlášení uživatele.
- **Profil** – Zobrazí profil uživatele, uživatel může změnit e-mail, heslo a login.
- **Dokumentace** – Zobrazí stránku s dokumentací jednotlivých bloků.
- **Moje projekty** – Zobrazí seznam projektů daného uživatele. Výběrem projektu bude uživatel přesměrován na daný projekt. U projektu lze editovat název nebo projekt smazat.
- **Sdílet projekt** – Vyvolá dialogové okno pro sdílení projektu, uživatel zadá e-mailovou adresu uživatele, se kterým chce projekt sdílet.
- **Jazyk** – Přepíná jazyk mezi češtinou a angličtinou.
- **Spustit na Android** – Automaticky nahraje otevřený projekt na zařízení Android a spustí jej, viz sekce 5.2.2.
- **Tutoriály** – Přesměruje na stránku s tutoriály, kde budou tutoriály na blokové programování, které byly přímo vytvořeny pro tuto webovou aplikaci. Dále zde budou odkazy na externí zdroje pro programování v `Lua/LÖVE`.

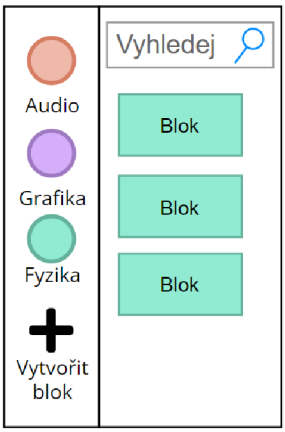
- **O aplikaci** – Přesměruje na stránku se základním představením aplikace. Bude zde popis fungování aplikace a jak ji správně používat. Popis bude doplněn ilustrativními obrázky.



Obrázek 5.5: Mockup horní lišty

Menu s bloky

Na obrázku 5.6 je vidět rozdělení bloků do kategorií podle funkcionality, každá kategorie je barevně rozlišena. Rozdělení kategorií se řídí základními funkcionalitami programovacího jazyka (matematika, práce s textem, logické funkce, cykly). Navíc zde budou kategorie, které kopírují rozdělení modulů ve framework `Lua/LÖVE`, jak je uvedeno v sekci 2.2. V levé části se pak nachází samotné bloky vybrané kategorie, které může uživatel použít přetáhnutím na pracovní plochu. Levý panel obsahuje vyhledávací pole, aby uživatel mohl blok vyhledat. Při pravém kliknutí na blok se zobrazí možnost přesměrování na dokumentaci daného bloku. Pod kategoriemi bude možnost vytvořit vlastní bloky.



Obrázek 5.6: Mockup menu s bloky

Okno pro zobrazení hry

V horní části se nachází okno pro zobrazení hry. Tlačítky start a stop může uživatel spustit hru a může tak hned vidět, co vytvořil. Aby uživatel viděl hru na celé obrazovce, bude se zde nacházet tlačítko pro maximalizaci tohoto okna.

Okno tutoriálů

Pokud chce uživatel využít tutoriál, okno tutoriálu se zobrazí v dolní části pravého panelu pod oknem hry. Je možné nastavit velikost tutoriálu tak, aby zabíral celý pravý panel a překrýval okno pro zobrazení hry.

5.6 Návrh tutoriálů

Při návrhu tutoriálů se vychází z již existujících tutoriálů blíže popsaných v sekci 3.2. Tutoriál má za cíl naučit uživatele programovat hru v `Lua/LÖVE`, principy programování se uživatel bude učit v průběhu programování hry. Tvorba her motivuje uživatele k učení. Tutoriály budou označeny podle obtížnosti a uživatel by měl ideálně začínat od nejjednoduššího po nejtěžší. Okno tutoriálu si uživatel bude moci ve webové aplikaci zobrazit přímo při programování hry (nemusí tedy přebíhat mezi dvěma okny), viz 5.5.

Struktura tutoriálu

Inspirací pro strukturu tutoriálů jsou tutoriály `Lua and LÖVE 11`, viz 3.2.4. Tyto tutoriály jsou velice dobře sestavené a pro uživatele srozumitelné. Tutoriály budou strukturované následujícím způsobem:

- **Název** – název tutoriálu.
- **Obtížnost** – obtížnost tutoriálu.
- **Odkaz na projekt vytvořené hry** – otevře projekt s kompletně vytvořenou hrou. Uživatel se může podívat, jak projekt funguje nebo může rovnou hru hrát.
- **Pravidla hry** – vysvětlení pravidel hry.
- **Přehled** – vysvětlení důležitých prvků hry z hlediska programování.
- **Obsah** – odkazy na jednotlivé sekce.
- **Jednotlivé sekce** – Každý tutoriál bude rozdělen do několika sekcí podle logiky vývoje hry. Každá sekce bude obsahovat kód a jeho slovní popis (co kód dělá a proč byl použit). Tutoriál bude umožňovat zobrazit kód v podobě bloků nebo textu. Kód mohou případně doplňovat obrázky, jak hra v tu chvíli vypadá. Na konci každé sekce bude možné se podívat na celý dosud vytvořený kód.

Témata tutoriálů

Pro vytvoření tutoriálů byly vybrány následující hry:

- **Snake** – hra Snake je ideální pro začátečníky, protože je to jednoduchá hra a pravidla zná téměř každý. Tutoriál nebude dlouhý a zvládne ho udělat kdokoliv, což může motivovat uživatele, kteří zkoušejí svoji první hru. Obtížnost je jednoduchá, uživatel se naučí principy `Lua/LÖVE`.
- **Asteroids** – je hra s jednoduchými pravidly a je ideální pro osvojení základních principů `Lua/LÖVE`. Tutoriál bude navíc přidávat složitější prvky, při tvorbě hry bude využit například obtížnější modul fyziky. Obtížnost je mírně pokročilá.

- **BlackJack** – tuto hru zná také většina uživatelů, ale je nutné připomenout pravidla hry. V tomto tutoriálu bude ilustrováno pokročilejší použití frameworku `Lua/LÖVE`, a to konkrétně modulární programování. Tutoriál je inspirován knihou `Developing Games on the Raspberry Pi`, viz 3.2.3.

Kapitola 6

Implementace Android aplikace

Tato kapitola popisuje implementaci Android aplikace, která má sloužit ke správě a spouštění projektů uživatele. V první části 6.1 jsou uvedeny využité technologie a také na jaké verze Androidu je aplikace určena. Poté je popsáno uživatelské rozhraní 6.2 a z čeho se skládá. Další část kapitoly je věnována komunikaci Android aplikace se serverem webové aplikace, viz 6.3. Pro autentizaci je využit Firebase Auth, který je uveden v sekci 6.4, pro posílání zpráv z webové aplikace do mobilního zařízení je využit Firebase Cloud Messaging, viz sekce 6.5. Následuje spouštění projektů uživatele 6.6. Posledním tématem jsou změny oproti návrhu aplikace 6.7.

6.1 Využité technologie

Android aplikace byla navržena tak, aby byla jednoduchá, přehledná a používání bylo pro uživatele intuitivní. Android aplikace slouží ke správě projektů uživatele a k jednoduchému spouštění projektů na zařízení Android. Je nutné ji používat v kombinaci s webovou aplikací, která byla pro tuto práci vytvořena. Pro fungování aplikace je nutné mít uživatelský účet. Po přihlášení uživatele ukáže aplikace seznam projektů uživatele, umožní stáhnout vytvořený projekt do zařízení Android a také projekt spustit nebo smazat. Díky propojení s webovou aplikací pomocí Firebase Cloud Messaging aplikace umožňuje uživateli automaticky nahrát vytvořený projekt z webu, viz sekce 6.5 a uživatel nemusí soubory do telefonu nahrávat ručně.

Pro implementaci bylo jako vývojářské prostředí využito Android Studio, které je volně dostupné. Je to nejlepší prostředí pro vývoj Android aplikací, obsahuje v sobě Android SDK (Software development kit), který má všechny potřebné funkce pro vývoj aplikací. Pro testování byly využity emulátory Android telefonů, které jsou také dostupné z Android Studia. Aplikace byla navržena tak, aby fungovala na zařízeních Android od API 24 (Android 7.0 Nougat) do API 32 (Android 11). To znamená, že dle informací od Google by aplikace měla fungovat na 89 %[1] všech Android zařízení.

6.2 Uživatelské rozhraní

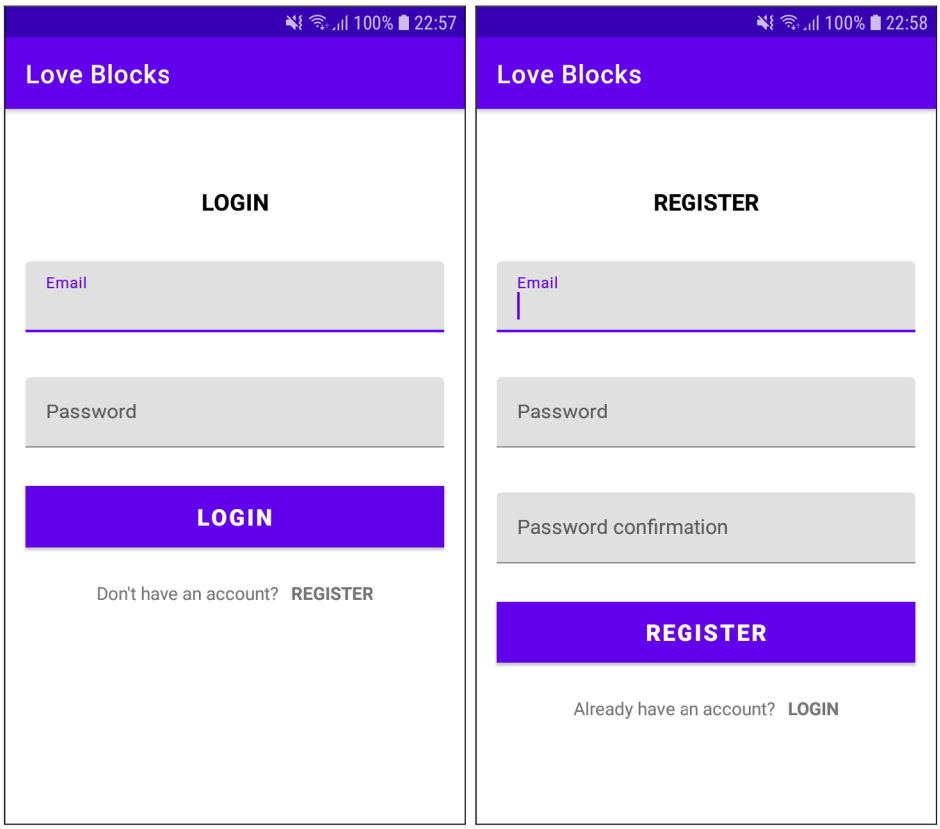
Grafické rozhraní je vytvořeno pomocí layouts (obrazovek), které jsou v kódu reprezentovány jako soubory `XML`, každá obrazovka koresponduje s jednou aktivitou. V případě této práce byly vytvořeny tři obrazovky: `activityLogin.xml`, `activityRegister.xml`,

[1] Dle informací dostupných z Android Studia

`activityMain.xml`, které korespondují s aktivitami: `MainActivity`, `LoginActivity` a dále `RegisterActivity`. Do `XML` souborů se přidávají jednotlivé grafické komponenty jako text, tlačítka nebo menu.

6.2.1 Login a register view

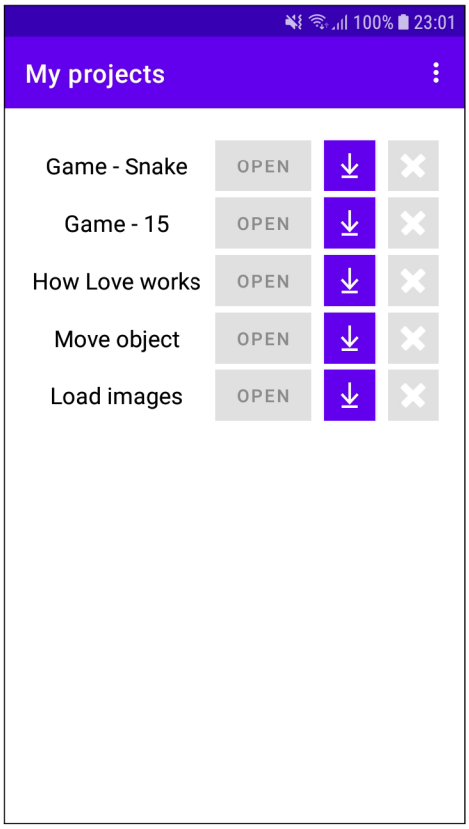
Obrazovky Login a Register by měly sloužit pro přihlášení a registraci uživatelů. Obrazovky jsou téměř shodné, Login je tvořen dvěma poli pro zadání e-mailu, hesla a také tlačítkem pro odeslání zadaných hodnot. Obrazovka Register obsahuje oproti Login obrazovce navíc pole pro potvrzení hesla. Na obou obrazovkách je vždy odkaz na druhou obrazovku. Pro vytvoření obrazovek byly využity komponenty z knihovny `Material Design`, která se běžně pro Android používá. Každá obrazovka má takzvaný rodičovský layout, který určuje, jak budou komponenty umístěny v rámci obrazovky. Pro tyto obrazovky byl využit `constraintlayout`, protože nabízí jednoduchý způsob, jak mají komponenty na sebe navazovat. Pro ilustraci lze uvést funkci `layout_constraintStart_toEndOf`, která připojí začátek komponenty ke konci definované komponenty. Tento layout se hodí pro responzivní design, jelikož umí jednoduše měnit velikosti komponent, aby se vešly na obrazovku. U zařízení s malou obrazovkou docházelo k problému, že se komponenty nevešly na jednu obrazovku, z toho důvodu byla do layout přidána komponenta `ScrollView`, která nabízí řešení posouváním obrazovky.



Obrázek 6.1: Obrazovky pro přihlášení a registraci

6.2.2 Main view

Do Main view se lze dostat po přihlášení do aplikace a funguje jako domovská obrazovka. Slouží hlavně pro zobrazení a správu projektů uživatele. Obdobně jako v předchozích obrazovkách byl využit `constraintlayout`. Obrazovka Main view obsahuje horní lištu, kde lze najít menu s několika funkcemi, jako jsou aktualizace projektů, odhlášení a informace o aplikaci. Menu je definované v souboru menu_main.xml. Obrazovka ukazuje seznam existujících projektů uživatele. Seznam projektů byl vytvořen pomocí `RecyclerView`, který slouží pro zobrazení seznamů. Nejprve bylo definováno, jak má vypadat jedna položka v seznamu a stejně budou vypadat ostatní položky v seznamu, pouze zde budou obměněna data. Položka seznamu je definovaná v souboru `recyclerview_adapter.xml` a skládá se z názvu projektu a tlačítek pro otevření, stažení a odstranění projektu.



Obrázek 6.2: Obrazovka projektů

6.3 Komunikace s webovou aplikací

Pro tento projekt je důležité, aby Android aplikace uměla komunikovat s webovou aplikací. Toho je docíleno pomocí REST API, které je blíže popsáno v sekci 5.3. Pro tvorbu HTTP requests byla využita knihovna `retrofit2`, což je HTTP klient pro Android. V souboru `RestAPI.kt` bylo pro každý request vytvořeno rozhraní (`Interface`) a bylo definováno o jaký typ request se jedná. Pro `GET` bylo využito `@GET('api/cesta')` a pro `POST` bylo využito `@POST('api/cesta')`. Komunikace probíhá pomocí dokumentů ve formátu `.json`, proto byly nastaveny hlavičky HTTP requests na `@Headers("Content-Type:`

application/json"). Dále je v rozhraní samotná funkce, která má jako argument typ dat, které přijímá, a typ dat, které vrací (všechny třídy s daty jsou definovány v souboru `RestAPI.kt`). Jako poslední se v rozhraní nachází funkce `create()`, kde je zadána URL adresa webového serveru a konvertor. Ten se používá pro serializaci do a z `.json` formátu. V případě této práce se jedná o `GsonConverter`, který pracuje s Gson[2]. Tato funkce vrací retrofit instanci s danými parametry. Následující kód ukazuje jedno z těchto rozhraní, konkrétně pro login uživatele.

```
interface ApiLogin {
    @Headers("Content-Type: application/json")
    @POST("login")
    fun loginUser(@Body loginData: LoginData): Call<LoginResponse>

    companion object {
        fun create(): ApiLogin {
            val retrofit = Retrofit.Builder()
                .addConverterFactory(GsonConverterFactory.create())
                .baseUrl(BASE_URL)
                .build()
            return retrofit.create(ApiLogin::class.java)
        }
    }
}
```

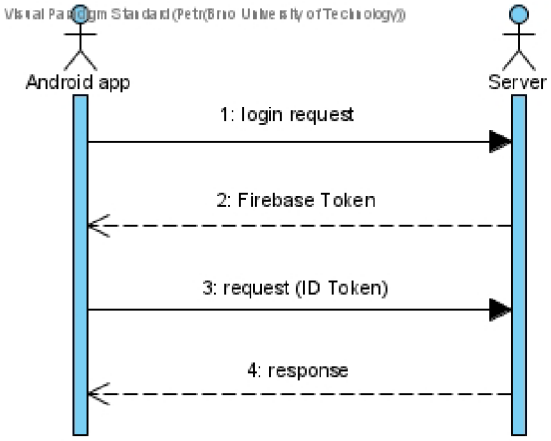
Pro komunikaci Android aplikace s webovou aplikací byla vytvořena třída `RestApiManager`, která obsahuje funkce pro komunikaci s REST API. Každá funkce přijímá jako vstup data, která se mají poslat na server a callback funkci, která se má zavolat až retrofit přijme odpověď ze serveru. Ve funkci se vždy vytvoří retrofit instance, která byla definována v daném rozhraní. Pomocí funkce `retrofit.enqueue()` se asynchronně pošle request webovému serveru a po přijetí odpovědi je pak zavolána callback funkce.

Pro komunikaci s webovou aplikací jsou používány dva druhy tokenů. Prvním je Firebase token, který se používá jednorázově při registraci uživatele, viz 6.4. Druhým tokenem je Firebase ID Token[3], který se používá při komunikaci se serverem využívajícím vlastní autentizační systém, což je případ této práce. Firebase ID Token lze získat z Firebase Tokenu po úspěšné registraci. ID Token se potom posílá s každým následujícím request na server a slouží pro ověření uživatele. Server si ověří uživatele tak, že si z ID Tokenu vygeneruje a ověří unikátní ID uživatele. Komunikace s webovou aplikací funguje podle diagramu 6.3.

6.4 Firebase Auth

V projektu je využito několik knihoven od Firebase, původně bylo v plánu využít knihovny Firebase UI pro vytvoření jednoduchého přihlašování a registrace uživatelů. Firebase UI implementuje funkcionalitu pro autentizaci uživatele, a to včetně vytvoření všech potřebných tzv. layouts a tříd. Tuto knihovnu nakonec nebylo možné využít, protože nepodporuje možnost autentizace přes vlastní server, což je v této práci potřeba, ale podporuje pouze využití Firebase autentizace. K autentizaci v aplikaci je využívána knihovna Firebase Auth, která

[2] https://github.com/google/gson
[3] https://firebase.google.com/docs/auth/admin/verify-id-tokens



Obrázek 6.3: Diagram komunikace mezi Android aplikací a serverem

umožňuje několik různých způsobů autentizace. Pro tuto aplikaci byla využita standardní autentizace pomocí e-mailu a hesla, protože se dobře implementuje i ve webové aplikaci. Jak už bylo řečeno, pro projekt není využita autentizace přes Firebase, ale je využit vlastní autentizační systém přes server s webovou aplikací. Je to z toho důvodu, aby všechna data byla uložena pohromadě v databázi na serveru.

Firebase Auth s vlastním autentizačním systémem funguje následovně: Při spuštění aplikace se otevře aktivita `MainActivity`, zde se pomocí `Firebase.auth.currentUser` v metodě `onStart` zkontroluje, zda je uživatel přihlášen. Pokud uživatel není přihlášen, vytvoří se intent na `LoginActivity` a uživatel je přesměrován na login obrazovku, kde je také možné nalézt odkaz na registraci. Uživatel zde vyplní příslušné údaje a potvrdí je tlačítkem, tím je zavolána metoda `login()` nebo `register()`, která odešle data na webový server. Aplikace odesílá tyto data:

- `email` – e-mail uživatele
- `password` – heslo
- `fcm_token` – Firebase Cloud Messaging Token, viz 6.5
- `name` – jméno (pouze při registraci)
- `password_confirmation` – potvrzení hesla (pouze při registraci)
- `terms` – souhlas s podmínkami použití (pouze při registraci, připraveno do budoucna, aplikace momentálně žádné nemá)

Webový server zpracuje data a vrátí buď chybové hlášení, které se zobrazí ve formě dialogového okna, nebo vygeneruje a pošle uživateli unikátní Firebase Token ve formátu secure `.json` Web token (JWTs). Pomocí metody `signInWithCustomToken()` a Firebase Tokenu dojde k autentizaci uživatele. Pokud je tato metoda úspěšná, spustí se `userIsLoggedIn()`, kde se vytvoří intent na `MainActivity` a uživatel je přesměrován na hlavní obrazovku. Vygenerovaný Firebase Token expiruje po jedné hodině, to ale neznamená, že by byl přihlášený uživatel po této době odhlášen. Pouze nelze tento token po uplynutí této doby znovu použít pro přihlášení a je potřeba vygenerovat nový.

6.5 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) slouží pro komunikaci z webové aplikace do aplikace Android, jak je uvedeno v sekci 5.2.2. FCM umí posílat dva druhy zpráv, datové zprávy a notifikace. V této aplikaci jsou využívány oba typy těchto zpráv. Pro využití FCM je nutné v souboru `AndroidManifest.xml` vytvořit službu (service), která má na starosti FCM. Tato služba musí rozšiřovat třídu `FirebaseMessagingService`. V manifestu je také možné definovat vzhled notifikace, například ikonu a její barvu. Služba je implementována v souboru `Messaging.kt`, důležitou funkcí je `onNewToken()`, která je volána vždy, když je vygenerován nový unikátní FCM Token. Po ní následuje metoda `sendRegistrationToServer()`, která odešle vygenerovaný token webovému serveru přes REST API pomocí knihovny retrofit, viz 6.3. FCM Token se také posílá serveru při přihlášení nebo registraci uživatele, viz 6.4. Pokud uživatel ve webové aplikaci stiskne tlačítko `Odeslat na Android`, odešle se zpráva na Android zařízení identifikované pomocí FCM Tokenu. Toto slouží k automatickému stažení a spuštění projektu na Android zařízení. Zpráva obsahuje informace o notifikaci a data ve formátu klíč/hodnota, má následující strukturu:

- `Předmět notifikace` – Open project (vždy stejné)
- `Text notifikace` – jméno projektu
- `url` – url ke stažení projektu
- `name` – jméno souboru

Jak je zpráva zpracována, záleží na aktuálním stavu Android zařízení. Pokud je aplikace v popředí, notifikace se neukáže a zpráva je zpracována metodou `onMessageReceived()`. V tomto případě funkce `onMessageReceived()` zkontroluje, že data poslaná se zprávou existují, a pomocí třídy `MyWorker.kt` spustí asynchronní Job. Ten stáhne data z url odkazu definovaného ve zprávě a uloží jej do zařízení, viz 6.6. Naopak, pokud je aplikace na pozadí nebo vypnutá, zobrazí se notifikace klasicky v horní liště mezi ostatními notifikacemi. Po kliknutí na notifikaci se otevře aplikace a data poslaná s notifikací jsou předána jako klíč/hodnota do Intent ve formátu extra hodnoty. Při spuštění aplikace funkce `checkIfAccessedFromFCMNotification()` kontroluje, jestli jsou tyto extra hodnoty zadány. Pokud ano, obdobně jako v prvním případě dojde ke stažení projektu. Poté jsou tyto extra hodnoty smazány, aby se stahování projektu neopakovalo znovu při dalším spuštění. Po stažení projektu dojde vždy k jeho spuštění, viz 6.6.

6.6 Spouštění her

Spouštění vytvořených her funguje pomocí oficiální aplikace `LÖVE for Android`[4], která je dostupná z Google Play Store. Android aplikace automaticky při spuštění kontroluje, zda je `LÖVE for Android` nainstalovaná a pokud není, vyzve uživatele k její instalaci. V Android verzi 11 je nutné přidat do AndroidManifest položku `queries` a zde definovat jméno aplikace, o které chceme zjišťovat informace. To vypadá následovně:

```
<queries>
    <package android:name="org.love2d.android" />
</queries>
```

[4] https://play.google.com/store/apps/details?id=org.love2d.android

Projekty uživatele jsou staženy do zařízení, jak bylo uvedeno v sekcích 6.3 a 6.5. Stažené projekty jsou ve formátu `.love`. Při spouštění her se ale objevilo několik problémů. Dle oficiální dokumentace love2d[5] by se hra měla stáhnout buď do složky `/sdcard/lovegame`, to ale funguje pouze do Android verze Pie, nebo do složky:

```
/sdcard/Android/data/org.love2d.android/files/games/lovegame
```

V tomto případě problém nastává u verze Androidu 11, konkrétně u API 29, kde Google nově představil Scoped Storage. To přináší lepší ochranu dat na externím úložišti tím, že aplikace jsou limitovány a mají pouze přístup do vlastní složky aplikace nebo do sdílených složek, zde je příkladem složka Stažené [3]. Pro tuto práci je potřeba spouštět projekty vytvořené uživatelem pomocí aplikace `LÖVE for Android`, to ale znamená, že se musí přenést data projektu do složky této aplikace, což je kvůli Scoped Storage u nejnovějších verzí Androidu zakázáno. Bohužel dokumentace `Lua/LÖVE` se tímto problémem vůbec nezabývá a doporučené řešení je připojení zařízení přes USB a přetažení souborů ručně. Což je pro uživatele zdlouhavé a pro zadání této práce velmi nevyhovující.

Pro vyřešení tohoto problému bylo vyzkoušeno hned několik postupů. Nejprve se nabízela možnost využít v AndroidManifest flag `requestLegacyExternalStorage`, kdy by se nevyužívalo Scoped Storage. Toto řešení ale funguje pouze do API verze 29 a v následujících verzích není funkční. Druhou možností bylo nevyužívat aplikace `LÖVE for Android` a rovnou generovat `.APK` soubory, od této myšlenky ale bylo upuštěno, protože generování `.APK` souboru je složité a uživatel by musel dlouho čekat. Také by byl problém, aby generování fungovalo automaticky bez zásahu uživatele. Nabízela se další možnost podporovat pouze nižší verze Androidu, což vzhledem k zadání této práce nedávalo smysl. Dalším řešením, které bylo vyzkoušeno, bylo využití oprávnění `MANAGE_EXTERNAL_STORAGE`. Pokud by aplikace měla být publikována na Google Store, bylo by nutné od Google Store získat souhlas k využití tohoto oprávnění. Tento souhlas většinou získávají aplikace, pro které je smysluplné, aby měly přístup k jiným složkám, což není tento případ. Nakonec bylo výsledné řešení nalezeno pomocí informací o aplikaci `LÖVE for Android` v Google Store, jde o možnost spouštění přes správce souborů, kdy po kliknutí na soubor `.love` by se měla zobrazit možnost "**spustit pomocí aplikace**". Nejprve tento způsob fungoval pouze na některých zařízeních, ale po specifikaci typu souboru už funguje na všech zařízeních. Implementace tohoto řešení spočívá v tom, že projekt je stažen do sdílené složky Stažené, kam mají přístup všechny aplikace bez omezení. Dále se vytvoří Intent, je mu předán stažený soubor a je nastaven typ souboru na `application/x-love-game`. Spuštěním tohoto Intent se spustí aplikace `LÖVE for Android` s projektem uživatele. V kódu je vidět vytvoření a spuštění `Intent`.

```
fun openLove2dApp(loveFilePath: String, context: Context){
    val uri = FileProvider.getUriForFile(context,
                "blocks.love.fileProvider", File(loveFilePath))
    val intent = Intent()
    intent.action = Intent.ACTION_VIEW
    intent.setDataAndType(uri, "application/x-love-game")
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
    context.startActivity(intent)
}
```

[5] https://love2d.org/wiki/Getting_Started

6.7 Změny oproti návrhu

Při implementaci aplikace byly oproti návrhu některé věci pozměněny, protože se ukázalo, že existuje lepší řešení nebo dané řešení není možné použít. Byla například opuštěna myšlenka generování `APK` souborů, protože by to bylo zdlouhavé a `APK` soubory by zabíraly hodně místa. Místo toho lze využít existující aplikace `LÖVE for Android` a generovat soubory typu `.love`. Není tedy nakonec potřeba oprávnění `REQUEST_INSTALL_PACKAGES`. V původním návrhu se nepočítalo s využitím aplikace `LÖVE for Android`, tu je nutné instalovat z Google Play Store. Android aplikace zkontroluje, zda je aplikace `LÖVE for Android` instalovaná, a pokud není, zobrazí uživateli dialogové okno, aby ji instaloval. Stejně tak aplikace kontroluje připojení k internetu, a že na zařízení jsou instalovány Google Play Services. Další změnou bylo nepoužití Firebase UI pro uživatelské rozhraní, protože nevyhovovalo požadavkům na autentizaci, jak bylo vysvětleno v sekci 6.4, místo toho bylo použito pouze Firebase Auth.

Kapitola 7

Implementace webové aplikace

Tato kapitola se zabývá implementací webové aplikace. Nejprve jsou představeny využité technologie 7.1 jako například framework `Laravel` pro backend a `VueJS` pro frontend. Je popsáno fungování Lua Language Server a jeho zakomponování do webové aplikace, viz 7.2. Dále je podrobně popsán celý backend webové aplikace, viz 7.3, a to konkrétně: uložení dat, komunikace s Android a práce s projekty. Sekce frontend se zabývá editory kódu a jejich komunikací, dále je podrobně vysvětleno spouštění her v prohlížeči. Následuje sekce, kde je popsáno vytváření bloků pro blokový editor, viz 7.5, a jsou představeny tutoriály, které byly vytvořeny v rámci této práce, viz 7.7. Poslední sekce se zaměřuje na změny oproti návrhu 7.8.

7.1 Využité technologie

Laravel Jetstream + InertiaJS

Pro tvorbu webové aplikace byl vybrán PHP framework Laravel. Laravel využívá architektury MVC, která je popsána v sekci 5.3. Pro práci s datovou a relační vrstvou využívá Eloquent ORM a Query Builder. Laravel nabízí mnoho funkcionalit, jako jsou například Queues nebo Jobs. Laravel je jednoduchý na používání a díky projektu Laravel Jetstream[1], který byl vytvořen jako začáteční bod pro vývoj webových aplikací, je ideální pro tento projekt. Jetstream nabízí implementaci základních funkcí každé aplikace, jako je login, registrace nebo verifikace e-mailu. Jetstream funguje společně s nástavbou `InertiaJS`[2], která přináší do Laravel jednoduchou manipulaci se šablonami napsanými v `VueJS`. Díky `InertiaJS` lze využít standardní Laravel směrování (routing) i v `VueJS` aplikacích a není potřeba instalovat další knihovny.

VueJS

Pro tvorbu části frontend byl použit JavaScript framework VueJS[3]. VueJS zjednodušuje tvorbu uživatelských rozhraní a umožňuje opětovné používání vytvořených komponent. Díky nástroji InertiaJS popsaném výše je spojení VueJS a Laravel velice jednoduché. Pro správu balíčků a závislostí je využit NPM[4].

[1] https://jetstream.laravel.com/2.x/introduction.html
[2] https://inertiajs.com/
[3] https://vuejs.org/
[4] https://www.npmjs.com/

Docker

Pro nasazení webové aplikace byl použit systém Docker, konkrétně `docker-compose`. Aplikace se skládá z několika kontejnerů: nginx, php, mysql a language-server. Tyto kontejnery tvoří hlavní část aplikace a jsou spuštěny automaticky. Dále aplikace obsahuje kontejnery `npm`, `artisan` a `composer`, ty jsou určeny pro spuštění příkazů typu:

```
docker-compose run --rm [kontejner] [příkaz]
```

Kontejner vykoná příkaz a poté se odstraní. To se hodí, pokud nejsou nainstalovány tyto technologie lokálně na serveru. Kontejnery jsou propojeny sítí. Tzv. environment variables pro `docker-compose` jsou definovány buď v samotném `docker-compose.yml` u jednotlivých kontejnerů, nebo v souboru `.env`.

V tomto projektu je použit webový server Nginx, konkrétně byl vybrán docker image `jonasal/nginx-certbot`[5]. Tento image byl vybrán z toho důvodu, že se automaticky stará o tvorbu a obnovu HTTPS certifikátů. Tato funkčnost je důležitá, jelikož je potřeba, aby byl projekt jednoduše nasaditelný v reálném prostředí. Jako databáze je použit docker image `mysql:8.0`, data jsou pak uložena ve volume, což znamená, že i při výpadku kontejneru nedojde ke ztrátě dat. Image PHP je vytvořen pomocí víceúrovňového `Dockerfile`. Jako základ je využit image `node:16`, ze kterého jsou kopírovány binární soubory do image `php:8.1-fpm`. To umožňuje spustit node ve výsledném kontejneru, který je potřeba pro spouštění her ve webovém prohlížeči, viz 7.3.3. Dalším řešením by bylo instalovat node přímo do výsledného PHP kontejneru, ale tento způsob není vhodný, protože výsledný image by měl několikanásobně větší velikost. PHP `Dockerfile` také obsahuje pouze potřebné balíčky pro fungování framework Laravel. Poslední image je `language-server`, který obdobně jako PHP využívá víceúrovňového `Dockerfile` se základem `node:16`, binární soubory jsou kopírovány do image `bitnami/minideb:latest`, což je minimalistický image založený na Debianu. `Dockerfile` pak obsahuje kopírování souborů Lua Language Server a jeho spuštění pomocí node.

7.2 Lua Language server

Language server přidává základní funkce do textového editoru, jako je zvýraznění syntaxe, automatické doplňování a dokumentace. Pro jazyk `Lua` byl použit Lua Language Server[6], protože lze využít mnoho funkcí, je obvykle používán a stále vyvíjen.

Server

Samotný server je jednoduchý, funguje přes WebSocket na portu `8888`, komunikace mezi klientem a serverem je směrovaná přes Nginx port `8080`, a to proto, že při připojení přímo na port `8888` webové prohlížeče jako Firefox hlásí, že se jedná o nezabezpečenou komunikaci a spojení nefunguje. Spojení přes port `8080` přidává ssl certifikáty a komunikace mezi klientem a Language serverem funguje správně.

[5] https://hub.docker.com/r/jonasal/nginx-certbot
[6] https://github.com/sumneko/lua-language-server

Klient

Pro připojení k Language serveru je potřeba vytvořit klienta. K tomu byl použit balíček Monaco Language Client[7], který se přes WebSocket připojí na port `8080`. Samotné připojení klienta a serveru bylo relativně jednoduché. Problém nastal, když bylo potřeba přidat dokumentaci pro `Lua/LÖVE` moduly a funkce. Ty už jsou předem připravené a dostupné přímo v Language serveru. Problém byl v tom, že Lua Language Server cílí primárně na klasické editory, jako je například VSCode. V takovém případě by bylo řešením přidat konfigurační soubor Language serveru do složky projektu, Language server si pak při připojení tento konfigurační soubor přečte a podle toho bude fungovat. To ale nelze v tomto případě udělat, protože jako editor je využit Monaco code editor, který funguje ve webovém prohlížeči, a není tedy žádná složka s konfiguračními soubory pro přečtení. Upravení konfigurace samotného Language serveru nefungovalo, protože jak bylo později zjištěno, tato funkčnost lze nastavit jen na straně klienta. Nakonec byl nalezen jiný způsob, kdy konfigurace se předává Language serveru jako součást klienta. Klient se vytváří pomocí funkce `MonacoLanguageClient`, která bere jako parametry jméno klienta a možnosti nastavení klienta. V možnostech nastavení klienta byla definována položka `middleware` a teprve zde bylo možné přidat konfiguraci Lua Language Serveru. Bez tohoto kroku Language server konfiguraci ignoroval.

7.3 Backend

Backend byl implementován ve framework Laravel s balíčkem Laravel Jetstream, viz 7.1. Laravel je založen na architektuře MVC, který je vysvětlen v sekci 5.3. V rámci backend byly vytvořeny tzv. controllers a modely, které korespondují s ER diagramem databáze, viz 5.2, ten definuje vztahy mezi modely. Například projekt může mít několik souborů, ale soubor patří pouze jednomu projektu. Těchto vztahů bylo docíleno využitím funkcí Laravel[8]. V modelu projekt byla použita funkce `hasMany(ProjectFile::class)`, tím bylo definováno, že projekt má mnoho souborů. Tento proces byl opakován pro všechny vztahy v ER diagramu. V tzv. controller se pak nachází business logika aplikace.

Díky `InertiaJS` je využito směrování na straně serveru (v klasické VueJS aplikaci by se využil například vue router). Cesty jsou definovány v souboru `web.php`, každá cesta má definováno, o jaký typ se jedná (`GET`, `POST`, `DELETE` atd.), její jméno a jakou funkci controlleru volá. Některé cesty jsou seskupeny pod `AUTH` middleware, který kontroluje, zda je uživatel přihlášen. Druhým middleware, který je využit, je kontrola oprávnění. To bylo implementováno pomocí balíčku Laravel permission[9]. Je využito pouze dvou základních rolí, role uživatele a administrátora. Pro budoucí rozvoj aplikace je zde možnost přidání dalších rolí a oprávnění. Základní role a oprávnění jsou přidána při prvotní inicializaci databáze a jsou definována v souboru `RolePermissionSeeder.php`. Obdobně jako pro uživatelské role byl pro tutoriály vytvořen soubor `TutorialSeeder.php`, který se stará o přidání tutoriálů do databáze při její inicializaci.

Pro autentizaci byl využit Laravel Fortify, který je součástí Laravel Jetstream. Fortify zajišťuje vše od autentizace přes e-mail a heslo až po dvoufázovou verifikaci. Tyto funkce jsou zajištěny přes tzv. Actions, což je PHP třída, která zajišťuje jeden konkrétní úkol jako například přihlášení uživatele. Rozdělení kódu do Actions zajišťuje a zvyšuje přehlednost

[7] https://github.com/TypeFox/monaco-languageclient
[8] https://laravel.com/docs/9.x/eloquent-relationships
[9] https://spatie.be/docs/laravel-permission/v5/introduction

kódu a jeho znovupoužitelnost. Tohoto principu je využito například při generování her pro prohlížeč, kdy tato funkcionalita má vlastní Action. V rámci backend bylo potřeba vyřešit, v jakém formátu jsou data předávána části frontend. Pro to byly použity API Resources[10], které se využívají pro definování formátu dat. Funguje to tak, že data z databáze se předají API Resource a ten vrátí data v definovaném formátu.

7.3.1 Práce s projekty

Pro práci s projektem bylo v souboru `ProjectControler.php` definováno několik základních funkcí pro vytvoření, zobrazení, smazání a upravení projektu. Při vzniku projektu se zároveň vytvoří adresář a soubory main.lua, conf.lua. Při mazání projektu je smazán adresář projektu včetně veřejných souborů projektu na disku `public` ke stažení. Disky adresáře a soubory jsou blíže vysvětleny v sekci 7.3.2. Pro zobrazení projektu je použita funkce `show`, která využívá knihovnu `InertiaJS` a její funkce `Inertia::render()`. Tato funkce zobrazí definovanou `VueJS` komponentu společně s daty, které jsou jí předány. V PHP jsou tato data definována jako pole, která `InertiaJS` do `VueJS` komponenty předává jako tzv. `props`. Jsou předávána vždy jen potřebná data, toho je docíleno pomocí tzv. lazy loading. To v praxi znamená, že některá data jsou načtena pouze v případě, že si o ně frontend zažádá. Konkrétně je toho využito, když chce uživatel hrát hru v prohlížeči, a to pomocí funkce `Inertia::lazy()`. Druhou možností je, že se data načtou pouze při prvním načtení stránky, což se používá při předání kódu projektu uživatele. Následující kód ukazuje všechny způsoby načítání dat, které jsou vysvětleny výše. Kód zobrazí `VueJS` komponentu `Project/Show`, která zobrazí projekt:

```php
<?php
return Inertia::render('Project/Show', [
    'project' => $project,
    'owner' => $project->isProjectOwner(Auth::user()),
    'config' => $conf,
    'main' => static fn() => $project->getMainLua(),
    'gamePackage' => Inertia::lazy(static fn() =>
        RefreshGameAction::execute($project)),
]);
?>
```

Pokud nastanou nějaké chyby, vrací se chybové hlášení pomocí vestavěné funkce Laravel `Redirect::back()->with(TEXT ZPRÁVY)`. Chybové hlášení se zobrazí v části frontend jako vyskakovací zpráva (flash message). Tuto funkci bylo možné využít díky tomu, že směrování je na straně serveru a ne na straně klienta. V `ProjectControler.php` byla dále vytvořena funkce pro sdílení projektu mezi uživateli, dále funkce kopírování cizího projektu pro vlastní užití a poslední funkcí je automatická aktualizace souboru `conf.lua`, pokud ho uživatel ve webové aplikaci upravil.

7.3.2 Uložení projektů uživatele

Pro ukládání projektů uživatele bylo rozhodnuto využít souborový systém místo databáze. Prvním důvodem je spustitelnost hry v prohlížeči, kde je využit nástroj `love.js`, viz 7.4.4.

[10] https://laravel.com/docs/9.x/eloquent-resources

Ten vyžaduje, aby projekt uživatele byl uložen jako soubor. Druhým důvodem je stahování projektů uživatelem. Projekty se stahují ve formátu `.love`, což je přejmenovaný `.zip`. Z těchto důvodů je lepší mít kód projektu uložen v souboru než v databázi. Laravel umožňuje jednoduchou práci se souborovým systémem díky velkému množství vestavěných funkcí. Laravel obsahuje takzvané disky, každý disk reprezentuje umístění v lokálním úložišti. V tomto případě jsou využity dva disky `loacal` a `public`. Do veřejného disku jsou ukládány pouze soubory na stažení, soubory projektu jsou ve formátu `.love`. Zbylé soubory jsou uloženy na lokálním disku, kde každý projekt má vlastí složku s unikátním jménem (jméno složky je uloženo v databázi). V této složce se nachází soubory `main.lua` obsahující kód, který uživatel napsal, a soubor `conf.lua`, který je automaticky generovaný, ale uživatel může upravit jeho podobu ve webové aplikaci. Dále se v adresáři projektu nachází složka `games`, která obsahuje hry vygenerované pomocí `love.js`. Protože na stejném projektu může pracovat více lidí, jsou tyto hry rozděleny do složek podle ID uživatele, aby nedošlo k přepisování her. Jako poslední se zde nachází adresář `resources`, kde jsou uloženy soubory k projektu nahrané uživatelem, například obrázky, zvuky atd.

V databázi jsou uloženy s projektem informace o tom, jaký editor uživatel aktuálně používá (zda se jedná o Monaco nebo Blockly). Je to z toho důvodu, aby se uživatel znovu po otevření stránky vrátil do stejného editoru. Také je uloženo, jak vypadá pracovní plocha aktuálně používaného editoru. Pro Blockly je to sloupec `blockly_workspace`, kde jsou informace o použitých blocích, jejich pozici a hodnotách (tato data jsou automaticky generována, viz 7.4.3). U Monaco to funguje obdobně, v sloupci `monaco_workspace` se ukládají pouze informace o pracovní ploše, ne kód.

7.3.3 Generování hry pro prohlížeč

Pro zobrazení hry v prohlížeči je využit balíček `love.js`, který je popsán blíže v sekci 7.4.4 z pohledu části frontend. V části backend je využit `love.js` pro generování hry ze zdrojových kódů. Pro vygenerování hry je důležité, aby hra byla ve formátu `.love`, který je uveden zde 7.3.2. Hra se generuje pomocí následujícího příkazu:

```
npx love.js -t title -c loveFilePath outputFilePath
```

V předchozím příkazu je proměnná `loveFilePath` cesta k `.love` souboru a `outputFilePath` je cesta, kam se mají uložit vygenerované soubory. Přepínač `-c` značí, že se má hra vygenerovat v módu kompatibility (jinak nebude fungovat v některých prohlížečích). Příkaz vygeneruje několik souborů: `love.wasm`, `index.html`, `game.js`, `game.data`. Hra je obsažena v souborech `love.wasm` a `game.data`. Pro její správné spuštění je potřeba získat ze souboru `game.js` pouze data o hře (například id_hry, velikost, seznam souborů). Zbytek souboru není potřebný, protože souvisí s `index.html`, který není využíván. Nevyužité soubory jsou smazány, ostatní jsou přesunuty na disk `public` do složky projektu a data o hře jsou předána části frontend.

7.3.4 Komunikace s Android aplikací pomocí REST API

Pro komunikaci s Android aplikací jsou využity REST API a FCM, viz 6.3. Pro PHP neexistuje oficiální Firebase SDK, bylo tedy využito balíčku `kreait/laravel-firebase`[11], který obsahuje všechny důležité funkce pro práci s Firebase. API cesty pro login a registraci jsou zpracovány v souboru `FirebaseLoginController`. Přihlášení má na starosti

[11] https://github.com/kreait/laravel-firebase

funkce `loginAndroidUser()`. Ta se nejprve pokusí najít uživatele s daným e-mailem a pokud ho nalezne, zkontroluje, že se rovná hash hesla a hash uložený v databázi. Při chybě vrátí odpověď ve formátu `.json` s polem "errors" a popisem chyby. Při úspěšné autentizaci se uloží do databáze k danému uživateli FCM Token, viz 6.4. Tento token se automaticky posílá při každém přihlášení. Dále se vygeneruje unikátní Firebase Token pomocí funkce `createCustomToken($user->id)`. Funkce bere jako argument ID uživatele, protože z Firebase Tokenu se v Android aplikaci následně generuje ID Token, který slouží pro autentizaci uživatele při následujících requests na server. Funkce pak vrací `.json` odpověď s Firebase Tokenem a jeho expirační dobou. Registraci zajišťuje funkce `registerAndroidUser()`, která nejprve validuje data poslaná s request, a pokud validace selže, vrátí odpověď ve formátu `.json` s polem "errors". Při úspěšné validaci vytvoří nového uživatele a uloží jej do databáze. Stejně jako při přihlášení se vygeneruje unikátní Firebase Token pomocí funkce `createCustomToken($user->id)`, který vrací s expirační dobou ve formátu `.json`. `FirebaseLoginController` obsahuje ještě funkci `refreshFCMToken()`, která je volána v případě zaslání dat na API `/token`. Volá se v případě, kdy Android aplikace vytvořila nový FCM Token. Funkce nejprve zkontroluje validitu ID Tokenu zaslaného ve zprávě pomocí `verifyIdToken($idTokenString)` a pokud je validní, vygeneruje z něj ID uživatele. K danému uživateli se pak uloží nový FCM Token.

Funkce pro práci s projekty se nacházejí v souboru `ProjectController`. Pro získání projektů uživatele slouží API `/projects`, které volá funkci `getFirebaseUserProjects()`. Ta funguje obdobně jako předchozí funkce: validuje ID Token a vrací buď chybu, nebo seznam projektů uživatele. Pro definování formátu dat, které se posílají zpět, bylo využito API Resource, konkrétně soubor `ProjectResource`. Důležitější je funkce `getProjectsLoveFile()`, která odpovídá API `/project/love`. Funkce vytvoří z projektu soubor typu `.love` (soubor `.love` je pouze přejmenovaný ZIP archiv) a ten pak lze stáhnout a spustit na Android zařízení. Funkce nejprve ověří uživatele pomocí ID Tokenu, poté nalezne projekt a související soubory, včetně cest k těmto souborům. Nejprve se vytvoří dočasný soubor funkcí `tmpfile()`, přidá se přípona `.zip` a vytvoří se nový zip archiv funkcí `ZipArchive()`. Do archivu se přidají zdrojové soubory main.lua a conf.lua plus všechny soubory nahrané uživatelem (může se jednat např o zvuky nebo obrázky). Archiv se uloží a pokud vše proběhlo bez chyb, je soubor .zip přejmenován na .love a je uložen do složky `public`, aby si ho uživatel mohl stáhnout. Funkce pak vrací `.json` odpověď s url ke stažení `.love` souboru a jméno projektu

7.3.5 Odesílání FCM zpráv na Android zařízení

Pro jednoduché stažení projektů na Android zařízení je využit Firebase Cloud Messaging, viz 6.5. V části backend se jedná o funkci `sendMessageToAndroid()`, která je volána po kliknutí na tlačítko "Download to Android". Funkce nejprve najde aktuálního uživatele pomocí vestavěné funkce Laravel `Auth::user()` a ověří si, že v databázi existuje záznam s FCM Tokenem. Pokud ano, inicializuje se FCM pomocí funkce z Firebase SDK `Firebase::messaging()`. Dále se ověří validita FCM Tokenu pomocí funkce `validateRegistrationTokens()` a vytvoří se soubor `.love`, jak je blíže popsáno v sekci 7.3.4. Pro zasílání FCM zpráv na Android zařízení je využita automatická konfigurace `$config = AndroidConfig::new()`. Nakonec se vytvoří FCM zpráva, jejímž cílem je zařízení definované pomocí FCM Tokenu, ke zprávě se připojí konfigurace `$config` a data. Data obsahují url ke stažení `.love` souboru a název projektu. Poté je vytvořena notifikace

s názvem "**Open project**" a text obsahující název projektu. V kódu vypadá vytvoření zprávy následovně:

```
<?php
$message = CloudMessage::withTarget('token', $user['FCM_token'])
    ->withAndroidConfig($config)
    ->withNotification(Notification::create('Open project',
                                            $project['name']))
    ->withData(['url' => Storage::disk('public')->url($path),
                'name' => $project['name'] . '.love']);
$messaging->send($message);
?>
```

7.4 Frontend

Pro implementaci části frontend existuje řada vhodných frameworků. Pro tuto práci byl použit framework `VueJS` v3, protože je známý a dobře se s ním pracuje. `VueJS` funguje na principu komponent: to znamená, že aplikace může být rozdělena do několika malých komponent, každá s vlastním životním cyklem. Tyto komponenty jsou pak znovupoužitelné v různých částech aplikace. Pomocí `VueJS` se vytváří Single Page Aplikace, což je potřeba u této webové aplikace.

V každé `VueJS` komponentě se definuje, jaký layout komponenta používá, tento layout definuje, jak bude stránka vypadat. Celkem byly vytvořeny tři takovéto layouts: `LoginLayout`, `ProjectLayout` a `AppLayout`. `LoginLayout` slouží pro stránky registrace, přihlášení a změny hesla. `ProjectLayout` je pro samotnou stránku projektu, kde se pak nachází blokový a textový editor kódu. Poslední `AppLayout` slouží pro zbytek aplikace jako například hlavní stránka. Pro základní uživatelské rozhraní byla využita knihovna komponent NativeUI[12], protože je kompatibilní s `VueJS` v3 a obsahuje velké množství komponent i s dokumentací, jak je využít. NativeUI umožňuje jednoduchou práci s dialogovými okny a vyskakovacími zprávami (flash message). Nejprve bylo ale nutné definovat `dialog-provider` a `message-provider`, aby s nimi bylo možné pracovat. To proběhlo v layout souborech a bylo docíleno toho, že funkcionalita je dostupná v celé aplikaci.

V klasické Single Page Aplikaci by komunikace mezi backend a frontend probíhala pomocí API, v tomto případě komunikuje `VueJS` s `Laravelem` pomocí knihovny `InertiaJS`, viz 7.1. Je k tomu využíváno například funkcí `Inertia.get()` pro získání dat z části backend a `Inertia.post()` pro odeslání dat na backend. `InertiaJS` předává data komponentám v tzv. `props`.

`VueJS` komponenty tvořící frontend webové aplikace jsou vytvářeny pomocí tzv. Composition API. To umožňuje definovat logiku komponenty na jednom místě pomocí funkce `setup()`. Důležitou funkcí v rámci Composition API je Reactivity API, to umožňuje vytvářet reaktivní proměnné pomocí funkcí `ref()` a `reactive()`. Dále je možnost využití tzv. Lifecycle Hooks, díky kterým se lze připojit na životní cyklus komponenty. Zde jsou klíčové funkce `onMounted()` a `onBeforeUnmount`. Funkce `onMounted()` je volána, když je komponenta připravena k použití (tzv. mounted). Funkce `onBeforeUnmount` se volá předtím, než bude komponenta odstraněna. Zde je zajištěno odstranění dat vytvořených komponentou. Na principu Composition API je založeno fungování celé části frontend a pro práci s `VueJS` je doporučováno.

[12] https://www.naiveui.com/en-US/os-theme

7.4.1 Monaco editor

Pro textové programování ve webové aplikaci byl využit Monaco editor, byla pro něj vytvořena samostatná VueJS komponenta. Pro přidání Monaco editoru je nutné provést konfiguraci Webpack[13], aby věděl, které soubory má načíst. Webpack je využit v této práci pro generování JavaScript balíčků. Při konfiguraci se ale objevil problém, protože docházelo ke konfliktu mezi některými knihovnami a kód nebylo možné spustit. Řešením bylo využít balíček `monaco-editor/loader`[14], díky kterému lze načíst Monaco editor přes CDN bez nutnosti jakékoliv konfigurace ve Webpack.

Ve VueJS komponentě pro Monaco editor byla tedy nejprve pomocí monaco loader načtena instance Monaco editoru a bylo přidáno nastavení, aby editor používal jazyk `Lua`. Dále byl nastaven automatický layout, aby bylo možné měnit velikost editoru. Posledním nastavením bylo, aby ve výchozím módu byl editor pouze pro čtení. To je z toho důvodu, že není možný převod z textového formátu do formátu bloků, o tom je více napsáno v sekci 7.4.3. Po inicializaci editoru se spustí klient, který se připojí přes websocket k Lua Language Server, viz 7.2. Data jsou komponentě předána pomocí tzv. `props`, zde jsou definovány obsahy souborů `main.lua` a `conf.lua` a také se zde nachází informace o pracovní ploše. Informace o pracovní ploše jsou při uložení projektu vloženy do databáze. Při odchodu ze stránky nebo přepnutí na blokový editor se instance editoru odstraní a uzavře se spojení s Lua Language Serverem.

V pravém horním rohu se nachází tlačítko `Edit`, po jehož kliknutí se vypne mód pouze pro čtení a uživatel může editovat kód. Editor obsahuje dvě záložky, a to pro soubory `main.lua` a `conf.lua`. Při přepnutí mezi záložkami se aktuální stav souboru vždy uloží do proměnné a z jiné proměnné se načte stav druhého souboru. Obrázek ukazující webovou aplikaci se zapnutým Monaco editorem je dostupný v příloze B

7.4.2 Blockly editor

Pro blokový editor byla využita knihovna Blockly, podrobnější informace jsou v sekci 4.2.5. Pro blokový editor byla vytvořena VueJS komponenta. Nejprve bylo definováno, jak má vypadat pracovní plocha Blockly editoru a přiblížení a oddálení pracovní plochy (zoom). Také byl přidán plugin pro posouvání pracovní plochy pomocí kolečka myši a koš pro odstranění nepotřebných bloků. Dále byl definován toolbox, kde se nachází všechny bloky, které Blockly využívá. Bloky jsou rozděleny do kategorií podle jejich typu. Toolbox je uložen v souboru `toolbox.js`. Protože byly definovány všechny potřebné části, bylo možné poté vytvořit pracovní plochu Blockly editoru pomocí funkce `Blockly.inject()`. Pokud uživatel vytváří nový projekt, vytvoří se prázdná pracovní plocha. Pokud projekt již existuje, načte se pracovní plocha projektu, na kterém uživatel pracuje. Data jsou načtena pomocí `props`, které byly předány VueJS komponentě. Poté se přidá plugin pro kopírování jak v rámci pracovní plochy, tak v rámci různých pracovních ploch. To je užitečné při používáni tutoriálů, ze kterých si může uživatel kopírovat bloky přímo do své pracovní plochy.

Aby byla pracovní plocha responzivní a bylo možné měnit její velikost, je nutné volat funkci `Blockly.svgResize()` s aktuální šířkou a výškou. Ty jsou získány pomocí funkce `useResizeObserver()` z knihovny VueUse[15], tato funkce je volána vždy, když se změní velikost pozorovaného HTML elementu. Obrázek ukazující webovou aplikaci se zapnutým Blockly editorem je dostupný v příloze C

[13] https://webpack.js.org/
[14] https://www.npmjs.com/package/@monaco-editor/loader
[15] https://vueuse.org/

7.4.3 Komunikace mezi editory

Komunikace mezi Monaco editorem a Blockly editorem je zařízena pomocí `VueJS` komponenty `ShowProject`. Každý editor funguje jako jedna `VueJS` komponenta a `ShowProject` zobrazí pouze ten editor, který uživatel aktuálně používá. Editory komunikují pomocí událostí (events). Události prvního editoru jsou zachyceny komponentou `ShowProject` a předány dále druhému editoru. Prakticky to probíhá tak, že když uživatel chce přejít například z editoru Blockly do textového editoru Monaco, Blockly vygeneruje z bloků kód v jazyce `Lua`, vytvoří event a do něj přidá vygenerovaný kód. Komponenta `ShowProject` zachytí tento event a předá data editoru Monaco za pomoci `props`. Následující kód popisuje předání dat z Blockly editoru do Monaco editoru, Blockly komponenta čeká na přepnutí editoru, konkrétně na změnu proměnné `saveCode`, která je dostupná jako `prop`. Sledování proměnné zajišťuje funkce `watch()`. Po změně proměnné dojde k vygenerování kódu z bloků pomocí `workspaceToCode()` a funkce `emit()` vytvoří event, ve kterém jsou předána data.

```
watch(() => props.saveCode, () => {
    code.value = luaGenerator.workspaceToCode(workspace)
    emit('passCodeToMonaco', code.value)
})
```

Předávání kódu opačným směrem (z textového editoru do blokového) nefunguje, protože je obtížnější z textového kódu vytvářet bloky; také to není součástí této práce. Zde se nabízí v budoucnu možnost aplikaci rozšířit a implementovat vytváření bloků z textového kódu.

Na podobném principu komunikace funguje ukládání pracovní plochy a kódu projektu, viz 7.3.2. Komunikace probíhá mezi aktuálním editorem a komponentou `ShowProject`. Editor vytvoří event funkcí `emit()` a přidá data, která chce uložit. Komponenta `ShowProject` zachytí tento event a pošle na backend data funkcí `Inertia.post()`. V rámci funkce `Inertia.post()` je nutné definovat cestu, na kterou se mají data posílat. V následujícím příkladě se posílají data na cestu s názvem `project.update`, projekt je definován pomocí proměnné `project`, v tomto případě je to projekt s `ID` 1. Data projektu uživatele jsou definována jako JavaScript objekt. V následujícím příkladě se předává pracovní plocha Monaco editoru a kód projektu uživatele.

```
Inertia.post(route('project.update', { project: 1 }), {
    monaco_workspace: workspace,
    main: code,
})
```

7.4.4 Spouštění her v prohlížeči

Samotné generování her je popsáno v sekci 7.3.3 a funguje pomocí balíčku `love.js` pro generování hry ze zdrojových kódů. Z pohledu části frontend funguje generování hry tak, že uživatel klikne na tlačítko **Spustit hru**. To zažádá backend o vygenerování hry a tato hra se potom spustí. V kódu je to řešeno funkcí `Inertia.reload()`, která znovu načte aktuální stránku. Této funkci je definován parametr `only`, který znamená, že se mají načíst pouze některá data, v tomto případě pouze data hry (v části backend je tato funkcionalita zajištěna pomocí tzv. lazy loading). Po úspěšném získání dat z části backend se zavolá funkce `runGame()`, která spustí hru.

```
Inertia.reload({
    only: ['gamePackage'],
```

```
    onFinish: () => {
        gameMode.value = false
        if (gamePackage.value !== undefined) runGame()
    },
})
```

Balíček `love.js` standardně funguje tak, že ze zdrojových souborů vygeneruje data hry v souboru `game.data`, potřebné scripty jsou v souborech `game.js`, `love.js` a `love.wasm`. Otevření hry se pak provádí pomocí souboru `index.html`. Tento postup není v tomto případě vhodný, protože je využit framework `VueJS`. Aby se hra mohla ve webové aplikaci spouštět, byly upraveny automaticky generované soubory, aby byly kompatibilní s `VueJS`. To bylo uděláno tak, že byla vytvořena nová komponenta, do které byla vložena potřebná data z `index.html`. Dále bylo nutné upravit soubor `game.js`, aby odpovídal formátu Composition API, viz 7.4. Jako poslední bylo nutné vyřešit konflikty mezi knihovnami v konfiguraci Webpack. Soubor `love.js` zůstává ve všech případech stejný.

Spouštění her

Problém nastal u souboru `love.wasm`, který je pro každou vygenerovanou hru různý. Soubor `love.js` využívá soubor `love.wasm`, předpokládá ale, že se oba nachází ve stejném adresáři. To je problematické, protože aplikaci využívá více lidí zároveň, a proto by mohl pouze jeden uživatel spustit vygenerovanou hru, která by byla přepsána, pokud by generoval hru jiný uživatel. Z tohoto důvodu bylo nutné upravit soubor `love.js`. Ten je vytvořený jako jedna funkce, která bere jako argument objekt s informacemi o hře, jenž má spustit. Řešením tohoto problému bylo přidání druhého argumentu s cestou k aktuálnímu `love.wasm` souboru. Tímto způsobem je zajištěno, že každý uživatel může spustit svoji vlastní hru a nedochází k přepisování souborů.

Event listener

Soubor `love.js` funguje tak, že při spuštěné hře jsou všechny stisknutí kláves klávesnice nebo myši předávány automaticky hře. To by nebyl problém, pokud by na stránce byla pouze hra a nic jiného, ale ve webové aplikaci jsou editory kódu (textový a blokový) a ty by nefungovaly. Byla snaha to vyřešit zakázáním propagace všech generovaných events dané hře. Toto řešení bylo účinné a bylo možné používat klávesnici a myš, ale nefungovaly klávesové zkratky, takže některé funkce Monaco editoru nebyly dostupné. Nakonec bylo potřeba zvolit náročné řešení a pozměnit kód `love.js`. V souboru byly nalezeny všechny funkce související s registrací nových tzv. `event listener`. Jelikož soubor `love.js` je dělaný jako jedna funkce, musela být registrace nových `event listener` přesunuta a definována globálně. Soubor `love.js` funguje stejně, akorát při registraci `event listener` se odkaz uloží do pole `allEventHandlers`. Toto pole je exportováno do `VueJS` komponenty, která se stará o spouštění hry, a uloženo do pole `events`.

Prakticky to funguje tak, že když uživatel spustí hru, webová aplikace se přepne do herního módu, kdy se všechny stisknutí kláves předávají do hry. Po stisknutí tlačítka pauza se zruší herní mód webové aplikace. Tlačítko pauza funguje tak, že projde pole `events` a odstraní každý registrovaný `event listener`. Pokud uživatel klikne na tlačítko play, dojde znovu k registraci každého `event listener` z pole `events`. Tlačítko stop zastaví hru a odstraní každý registrovaný `event listener`.

Obrázek 7.1: Tlačítka pro kontrolování hry

Únik paměti (memory leak)

Při spouštění her dochází k únikům paměti (memory leak), a to přesně k 16 Mb při každém spuštění hry. Bylo to zjištěno při spuštění nástrojů pro vývojáře v prohlížeči Firefox. Konkrétně přibývá vždy 16 MB do ArrayBuffer[16]. Je to tím, že balíček `love.js` byl původně koncipován jako jedna HTML stránka, na které šla spustit pouze jedna hra. Jelikož hra nelze opakovaně spouštět, nedochází k velkým únikům paměti, a proto to pravděpodobně nebylo řešeno. K `love.js` ale neexistuje žádná dokumentace, takže je obtížné určit, jak které části fungují. Tento problém lze řešit znovunačtením stránky, ale webová aplikace funguje jako Single Page Aplikace a znovunačtení stránky může být tedy pro uživatele rušivé. Bylo to vyřešeno tak, že do horního panelu byl přidán ukazatel aktuálně alokované paměti a tlačítko pro případné znovu načtení stránky. To lze vidět na obrázku 7.2.

Obrázek 7.2: Ukazatel aktuálně alokované paměti

V kódu je použita funkce `performance.memory.usedJSHeapSize()` pro získání aktuální hodnoty alokované paměti. Tato funkce ale není dostupná v prohlížeči Firefox, proto v tomto prohlížeči dochází k odhadu aktuální alokované paměti. Bylo zjištěno, kolik MB paměti stránka alokuje při normálním provozu a je přidáno 16 MB při každém spuštění hry.

7.5 Vytváření bloků

Vytváření bloků funguje podle návrhu v sekci 5.4.1. Bloky byly vytvářeny s pomocí Blockly Developer Tools a dokumentace `Lua/LÖVE`, dokumentace definovala podobu bloků a jaké mají vstupy a výstupy. V rámci této práce bylo vytvořeno 227 bloků, které jsou rozděleny do 12 kategorií podle `Lua/LÖVE` modulů, například moduly grafika, fyzika, callback funkce, práce s klávesnicí nebo systém. Každá kategorie je definována svou vlastní barvou, aby to bylo pro uživatele přehledné. Nebyly ale definovány bloky pro některé `Lua/LÖVE` moduly, například pro práci s vlákny nebo práci s fonty, protože nejsou tak často používané. Dále nebyly definovány jako bloky funkce pro práci s tzv. Texture, Shader, Canvas atd. Tyto části jsou v `Lua/LÖVE` definované jako objekty, ale Blockly nepodporuje principy objektově orientovaného programování. Jelikož se jedná o složitější funkcionalitu, která se lépe řeší v textové podobě kódu, tak tyto funkce nebyly dále řešeny. To je další možné rozšíření aplikace do budoucna.

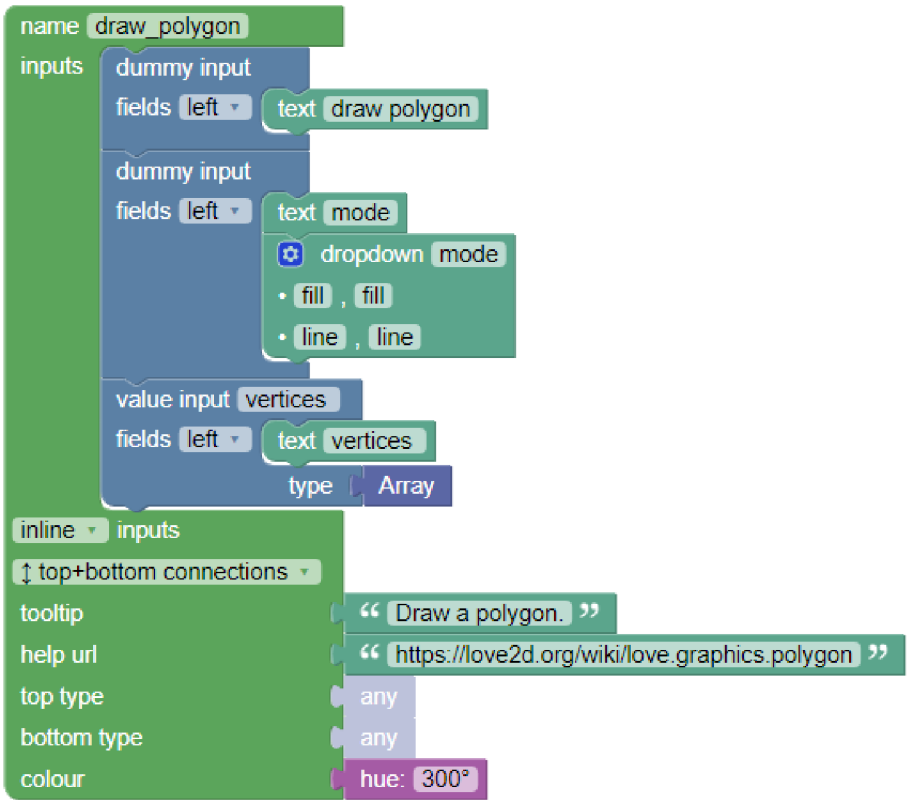
Nabízela se možnost vytvoření skriptu pro automatické generování bloků, ale tohle řešení by mělo několik problémů. Podle dokumentace `Lua/LÖVE` je podle názvu funkce těžké určit k čemu slouží. Většina funkcí může být definována různými způsoby a některé varianty jsou vhodnější pro tvorbu daného bloku – například funkce s nekonečným počtem argumentů pro tvorbu bloku vhodné nejsou. Pro bloky je nutné vytvořit popis jejich fungo-

[16] https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer

vání, aby uživatel mohl pochopit, co blok dělá. Z toho důvodu byly Bloky vytvářeny ručně tak, že v Blockly Developer Tools byly definovány vstupy a výstupy daného bloku. Existují tři různé možnosti vstupů:

- `Dummy input` – Používá se pro psaní textu, či popisu bloku. Také se zde využívá tzv. dropdown, který umožňuje výběr z více možností.
- `Value input` – Umožní vložení dalšího bloku s předem definovaným typem hodnoty, například pole nebo číslo.
- `Statement input` – Umožňuje vkládání dalších bloků uvnitř samotného bloku, což je užitečné například pro funkce nebo cykly.

U bloků lze definovat typ spojení s dalšími bloky pomocí tzv. connections, což určuje směr spojení (nahoru, dolů, vlevo, žádné). To je důležité například u bloků, které vrací hodnoty. U těch je nutné, aby měly spojení vlevo a tuto hodnotu mohly předat jinému bloku. Pokud blok nevrací žádnou hodnotu, bylo ponecháno spojení nahoru a dolů, aby se mohly bloky řadit za sebe. Každá callback funkce je vytvořená jako blok se vstupem typu `statement`, aby bylo možné vkládat bloky dovnitř této funkce. Pokud nějaká funkce využívá proměnné, tak dané proměnné jsou automaticky definovány při použití tohoto bloku. Na následujícím obrázku 7.3 je vidět vytváření bloku `draw polygon` pomocí Blockly Developer Tools. Obrázek 7.4 ukazuje již výsledný blok `draw polygon`, který uživatel používá ve webové aplikaci.



Obrázek 7.3: Definování bloků v Blockly Developer Tools

Obrázek 7.4: Výsledný blok Draw polygon vytvořený pomocí Blockly Developer Tools

7.6 Práce s bloky

Pomocí Blockly Developer Tools lze definovanou strukturu bloku stáhnout ve formátu `.json` a všechny takto definované bloky byly uloženy do souboru `blocks.json`. Dalším krokem bylo rozřazení bloků do kategorií, které odpovídají modulům `Lua/LÖVE`. Kategorie byly uloženy v toolbox, který se nachází v souboru `toolbox.js`, viz 7.4.2. V toolbox byly nastaveny u některých bloků výchozí vstupní hodnoty. To se dělá pomocí tzv. Shadow Block. To je typ bloku, který nese zadanou výchozí hodnotu a lze ho přepsat jiným blokem. Dále bylo nutné nastavit kód, který bude blokem generován. K tomu se využívají generátory kódu, které jsou v knihovně Blockly dostupné. Generování kódu je podrobně vysvětleno v sekci 5.4.2. Zde se ale objevil problém při generování bloků s proměnnými, kdy typ bloku má definovat interní Blockly konstanta `Blockly.Names.NameType.VARIABLE`. Ta ale v aktuální verzi není dostupná, stejně jako starší konstanta `Blockly.Variables.NAME_TYPE`. Řešením bylo definovat typ napevno pomocí řetězce `'VARIABLE'`. Tento problém by měl být vyřešen v následující verzi Blockly[17] a mělo by být možné použít klasické konstanty.

Při generování bloků se objevilo hned několik problémů. Prvním z nich bylo, že `Lua/LÖVE` má velké množství funkcí, které vrací více hodnot, což je ale problém, protože Blockly tuto funkcionalitu nepodporuje. Řešením bylo vrátit hodnoty v poli, kdy první hodnota pole odpovídá první vrácené hodnotě atd. To je vidět na následujícím kódu, kdy v proměnné `code` je `Lua/LÖVE` funkce `love.filesystem.load()` vložena do složených závorek, pomocí kterých se v jazyce `Lua` definuje pole.

```
Blockly.Lua.filesystem_load = function (block) {
    const textName = block.getFieldValue('name')
    const code = '{ love.filesystem.load( ' + textName + ' ) }'
    return [code, Blockly.Lua.ORDER_NONE]
}
```

7.7 Tutoriály

V rámci práce vzniklo několik tutoriálů, aby se uživatel mohl naučit programovat v jazyce `Lua/LÖVE` za pomoci blokového nebo textového programování. Struktura tutoriálů je stejná, jak je vysvětleno v sekci 5.6. Je zde vyznačena obtížnost tutoriálu a u tutoriálů her jsou vysvětlena pravidla hry, aby se uživatel orientoval v samotné hře. V návrhu se předpokládalo, že vzniknou tři tutoriály na hry Snake, Asteroids a BlackJack. Nakonec byly implementovány následující tutoriály, protože bylo potřeba, aby se uživatel prvně naučil základy v `Lua/LÖVE` a teprve potom mohl zkoušet tvorbu složitějších her.

- `How Lua/LÖVE works` – Vysvětluje základní princip frameworku `Lua/LÖVE`, jakým způsobem fungují callback funkce a jak se kreslí objekty na obrazovku.

[17] https://github.com/google/blockly/issues/6008

- `Check if key is pressed` – Vysvětluje, jak kontrolovat, zda je zmáčknutá daná klávesa.
- `Move object` – Vysvětlení principu, jak pohybovat s objekty na obrazovce pomocí stisku kláves.
- `Load image to the game` – Ukazuje načtení obrázku do hry.
- `How to shoot` – Ukazuje princip, jak vytvořit hráče, který dokáže střílet.
- `Fifteen - game` – Tutoriál ukazuje, jak vytvořit hru Fifteen, aby fungovala na dotykovém zařízení Android. Jedná se o jednoduchou hru, uživatel ji dobře zná a na programování není složitá.
- `Snake - game` – Tutoriál ukazuje, jak vytvořit hru Snake, aby fungovala na dotykovém zařízení Android. Zde se částečně přechází z blokového do textového programování. Struktura hry je definována pomocí bloků a složitější funkce jsou naprogramovány v textové podobě.

Tutoriály jsou v pravém dolním rohu webové aplikace a ukazuje se zde kompletní nabídka tutoriálů. Zde je možné otevřít tutoriál, který je vždy tvořen textovým popisem a částí kódu. Kód lze zobrazit buď v textové podobě, nebo v blocích. V blokovém formátu je okno tvořeno jako pracovní plocha Blockly s pluginem pro kopírování bloků, stejně jak bylo popsáno v sekci 7.4.2. Je tedy možné kopírovat bloky stisknutím pravého tlačítka myši a vybráním možnosti `COPY`, kopírovat lze z okna tutoriálu na hlavní pracovní plochu. Do budoucna se nabízí další rozvoj a rozšíření tutoriálů o další hry, které by obsahovaly složitější funkce `Lua/LÖVE`.

7.8 Změny oproti návrhu

Při implementaci aplikace došlo k několika změnám oproti návrhu. Jednou z větších změn byla změna schématu databáze. Nové schéma databáze se nachází v příloze A. Konkrétně byly přidány tabulky související s rolemi a oprávněními uživatelů, viz 7.3. Byla přidána také možnost, aby k jednomu projektu mělo přístup více uživatelů. K projektům byla přidána možnost ukládání pracovních ploch Blockly a Monaco editoru. Dále byla přidána informace o tom, který editor uživatel aktuálně využívá. Nebyla implementována možnost přidání vlastních bloků. O tuto možnost by bylo dobré rozšíření aplikace do budoucna. Dále nebyla implementována vícejazyčnost webové aplikace, aplikace je pouze anglickém jazyce. Implementace češtiny by nebyla složitá, ale bylo by nutné přeložit samotnou aplikaci a také všechny bloky, což je časově náročné. Oproti návrhu je také změna v generování hry pro Android zařízení, kdy soubory nejsou generovány ve formátu `.APK`, ale ve formátu `.love`, což je lepší řešení. Toto řešení je blíže vysvětleno v sekci 6.6. Nakonec došlo ke změnám v tutoriálech, které vznikly v rámci této práce. Změny reagují na potřebu uživatele se nejprve naučit základy `Lua/LÖVE`.

Kapitola 8

Nasazení systému, testování, publikování a možnosti rozšíření

V této kapitole je nejprve popsáno, jak byly aplikace testovány 8.1. Jedná se především o integrační testování, které má za cíl ověřit, že aplikace funguje jako celek. Samotný systém webové aplikace byl nasazen do Oracle Cloud, viz 8.2, obě aplikace byly publikovány jako open-source 8.3. Nakonec byly navrhnuty možnosti dalšího rozvoje aplikací 8.4.

8.1 Testování

Android aplikace podporuje verze API od 24 po API 32, viz 6.1. Bylo provedeno integrační testování, které probíhalo na několika fyzických zařízeních Android, konkrétně s verzí API 26 a verzí API 31. Cílem bylo otestovat, že aplikace funguje správně na různých API Android. Původně bylo v plánu testovat aplikaci pomocí emulátoru dostupného v programu Android Studio. Zde se ale objevil problém, že na emulátoru nebylo možné nainstalovat z Google Play Store aplikaci `LÖVE for Android`, která je potřeba pro spouštění `Lua/LÖVE` her na zařízení Android. Na emulátorech bylo tedy možné testovat pouze omezenou funkcionalitu vytvořené Android aplikace. Ukázalo se, že aplikace funguje a případné nalezené chyby byly opraveny. Jediným problémem, který přetrval, bylo zobrazení her aplikací `LÖVE for Android`, kdy aplikace `LÖVE for Android` pracuje v režimu celé obrazovky a u některých Android zařízení je část hry pod horní lištou. Toto zobrazení hry je způsobeno aplikací `LÖVE for Android`.

Testování REST API u webové aplikace bylo provedeno ručně pomocí nástroje `Postman`, který slouží pro zasílání HTTP request na server. Byl vytvořen určitý HTTP request a bylo kontrolováno, že odpověď serveru odpovídá očekávání. Testování REST API probíhalo v průběhu implementace. Tímto bylo ověřeno, že komunikace mezi Android aplikací a webovou aplikací je plně funkční. Také byly využity vývojářské nástroje Firefox pro testování funkčnosti webové aplikace. Testování se soustředilo na paměť, kterou aplikace využívá. Zde byl nalezen únik paměti (memory leak), který se nachází v balíčku `love.js`. Problém se nepodařilo odstranit, ale byl vyřešen přidáním tlačítka pro znovunačtení stránky. Celé řešení problému je vysvětleno v sekci 7.4.4.

8.2 Nasazení systému

K nasazení webové aplikace bylo využito Oracle Cloud Free Tier[1]. Oracle nabízí čtyři procesory s architekturou ARM a až 18 GB operační paměti. Protože funguje na ARM procesorech, bylo nutné upravit `docker-compose.yml`, aby všechny využité image fungovaly na ARM architektuře. Výsledek je v souboru `docker-compose-arm.yml`. Webová aplikace je dostupná na adrese `loveblocks.tk`[2]. Postup pro lokální nasazení webové aplikace je popsaný v souboru `README.md`, je nutné mít nainstalovaný `docker` a `docker-compose`.

8.3 Publikování

Android aplikace byla publikována v Google Play Store pod názvem `Love Blocks`[3], bylo nutné vytvořit Goodle developer účet. Potom v Google Play Console bylo nutné definovat typ a obsah aplikace, uvést, jak jsou údaje zabezpečeny a přidat podmínky použití. Bylo také potřeba přidat ikonu aplikace a obrázky, jak aplikace vypadá. Schvalovací proces trval několik dní, poté byla aplikace zveřejněna a je dostupná ke stažení. Zdrojový kód Android aplikace je volně dostupný na Github[4] pod `MIT` licencí. Původně bylo v plánu aplikaci publikovat v repozitáři `F-Droid`, který slouží pro publikování open-source Android aplikací, ale nebylo to možné kvůli podmínkám `F-Droid`, kdy není možné publikovat aplikaci využívající služeb Firebase[5]. V tomto případě je využíván Firebase Cloud Messaging a Firebase Auth.

Zdrojový kód webové aplikace byl publikován na Github[6] jako open-source pod `MIT` licencí. Webová aplikace byla také zveřejněna na několika fórech: Hacker News[7], love2d Reddit[8], love2d fórum[9]. Zde si mohou uživatelé aplikaci vyzkoušet, navrhnout možné zlepšení a napsat zpětnou vazbu, jak se s ní pracuje. Vybraná fóra se zabývají frameworkem `LÖVE`, proto jsou vhodná pro získání zpětné vazby.

8.4 Možnosti rozšíření

Webová aplikace je plně funkční a uživatel může programovat své hry pomocí bloků i textového programování. Přesto se ale nabízí několik možností vylepšení:

- `Odstranění úniku paměti` – V balíčku `love.js` dochází k únikům paměti, proto by bylo vhodné do budoucna balíček `love.js` upravit.
- `Lepší debugování` – Aplikace v současném stavu nemá dobrý způsob pro debugování her. To by bylo možné pomocí `love.js`, ale k balíčku neexistuje žádná dokumentace.
- `Přidání vlastních bloků` – Pro uživatele by mohla být přínosná možnost přidání vlastních bloků.

[1] https://www.oracle.com/cloud/free/
[2] https://loveblocks.tk/
[3] https://play.google.com/store/apps/details?id=blocks.love
[4] https://github.com/meda10/Love-blocks-android/tree/master
[5] https://f-droid.org/docs/Inclusion_Policy/
[6] https://github.com/meda10/Love-blocks-web
[7] https://news.ycombinator.com/item?id=31304492
[8] https://www.reddit.com/r/love2d/comments/ul35wx/love_blocks_code_editor_for_l%C3%B6ve/
[9] https://love2d.org/forums/viewtopic.php?f=5&t=92894

- `Doplnění bloků` – K některým modulům `Lua/LÖVE` nebyly vytvořeny bloky, což by bylo možné doplnit. Dále je možné vytvoření bloků pro objektově orientované programování. To v současné době není v knihovně Blockly možné.
- `Přeložení do češtiny` – Webová aplikace je pouze v anglickém jazyce, nabízí se překlad aplikace, tutoriálů a všech bloků do češtiny.
- `Údržba bloků` – Pro údržbu bloků vznikl skript `blockMaintenance.sh`. Tento skript kontroluje aktuálnost bloků pomocí `LÖVE API` a šel by rozšířit o automatické generování některých bloků. Některé bloky by bylo nutné udělat ručně.

Kapitola 9

Závěr

Cílem diplomové práce bylo vytvořit webovou aplikaci pro výuku programování v `Lua/LÖVE`, která má podporovat jak vizuální, tak textovou formu programování. V rámci práce vznikla také Android aplikace, která slouží pro nahrávání a spouštění vzniklých uživatelských projektů na Android zařízení.

V rámci práce jsem se nejprve seznámil s jazykem `Lua` a frameworkem `LÖVE`, který slouží pro tvorbu 2D her pro různá zařízení. Práce se zabývala přístupy k výuce programování, autorka článku Cynthia C. Selby [19] pojmenovává existující přístupy k výuce programování: analýza kódu, stavební bloky, jednoduché jednotky a celé systémy. Autorka dále doporučuje tyto přístupy kombinovat. V následující části jsem se zaměřil na existující tutoriály pro `Lua/LÖVE`, ty nevíce relevantní byly podrobně popsány a mezi sebou porovnány. Jako nejvíce přínosný hodnotím tutoriál `Lua and LÖVE 11`, který učí programovat přímo na tvorbě konkrétní hry. Tím jsem se částečně inspiroval při tvorbě tutoriálů v rámci této práce.

Zaměřil jsem se dále na téma možností vizuálního programování. Nejprve jsem se seznámil s principy vizuálního programování, a to konkrétně s programováním toku dat a blokovým programováním. V práci jsem se pak zabýval především blokovým programováním, protože bylo pro zamýšlenou aplikaci vhodnější. Pro blokové programování existuje řada nástrojů, na které jsem se zaměřil a porovnal je mezi sebou. Asi nejrozšířenějším nástrojem je Scratch, který byl také inspirací pro webovou aplikaci. Zabýval jsem se i knihovnou Blockly, kterou používám pro tvorbu bloků v rámci webové aplikace. Užitečné bylo doporučení autorů Blockly, jak správně s Blockly pracovat. Důležitou funkcionalitou je generování textové formy kódu z bloků.

V rámci práce jsem navrhl webovou aplikaci, kde uživatel může programovat hry ve frameworku `Lua/LÖVE` jak pomocí blokového, tak textového programování. Jedním z požadavků byla možnost spouštění her v prohlížeči. Pro jednoduché spuštění uživatelských projektů na platformě Android byla navržena Android aplikace. Tyto dvě aplikace mezi sebou komunikují pomocí REST API a Firebase Cloud Messaging. V návrhu jsem se dále zabýval tvorbou bloků pro `Lua/LÖVE` a generováním kódu z těchto bloků.

V rámci implementace Android aplikace jsem nejprve implementoval uživatelské rozhraní. Komunikace s webovou aplikací probíhá přes REST API, autentizaci uživatele zajišťuje knihovna Firebase Auth. Aby bylo možné posílat zprávy z webové aplikace do Android aplikace, využil jsem služby Firebase Cloud Messaging a díky tomu uživatel může automaticky stáhnout a spustit svůj vytvořený projekt do Android zařízení. Spouštění vytvořených projektů probíhá pomocí aplikace `LÖVE for Android`, což je oficiální aplikace od tvůrců framework `LÖVE`.

Pro implementaci webové aplikace jsem využil pro část backend Framework Laravel a pro část frontend VueJS. Blokový editor jsem do webové aplikace implementoval pomocí Blockly a pro textový editor jsem využil Monaco code editor. K Monaco editoru jsem připojil Lua Language Server pro zlepšení funkcí textového editoru (zvýraznění syntaxe, automatické doplňování, dokumentace). Důležitou částí webové aplikace je možnost spouštění her v prohlížeči, proto používám balíčku `love.js`. Pro webovou aplikaci bylo nutné vytvořit bloky, které odpovídají funkcím z `Lua/LÖVE`, aby uživatel mohl pomocí nich programovat. Ke každému bloku jsem přidal generátor kódu a jeho výchozí hodnoty. Nakonec jsem do webové aplikace implementoval jednotlivé tutoriály, na kterých si uživatel může vyzkoušet programování. Aplikace je vhodná pro ty, co začínají s programováním v `Lua/LÖVE`. Blokové programování je vhodné pro seznámení se s programováním, pro pokročilejší projekty už lze využít textový editor.

Obě dvě aplikace jsem publikoval jako open-source a jsou volně dostupné na Github. Android aplikace je také ke stažení v Google Play Store[1].

[1] https://play.google.com/store/apps/details?id=blocks.love

Literatura

[1] *Config Files* [online]. 2021 [cit. 2021-18-12]. Dostupné z: https://love2d.org/wiki/Config_Files.

[2] *Programming in DRAKON* [online]. 2021 [cit. 2021-23-12]. Dostupné z: https://drakon.tech/read/programming_in_drakon.

[3] *Storage updates in Android 11* [online]. 2021 [cit. 2021-12-4]. Dostupné z: https://developer.android.com/about/versions/11/privacy/storage#test-scoped-storage.

[4] Akinlaja, D. D. *LÖVE2d for Lua Game Programming*. Packt Publishing Ltd, 2013. ISBN 9781782161608.

[5] Firebase. *Introducing Firebase Cloud Messaging* [online]. 2021 [cit. 2021-19-4]. Dostupné z: https://www.youtube.com/watch?v=sioEY4tWmLI.

[6] Fraser, N. Ten things we've learned from Blockly. In: IEEE. *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 2015, s. 49–50.

[7] Ierusalimschy, R. Programming in Lua. 2016.

[8] Johnston, W. M., Hanna, J. P. a Millar, R. J. Advances in dataflow programming languages. *ACM computing surveys (CSUR)*. ACM New York, NY, USA. 2004, sv. 36, č. 1, s. 1–34.

[9] Kuhail, M. A., Farooq, S., Hammad, R. a Bahja, M. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. *IEEE Access*. IEEE. 2021.

[10] Kurihara, A., Sasaki, A., Wakita, K. a Hosobe, H. A programming environment for visual block-based domain-specific languages. *Procedia Computer Science*. Elsevier. 2015, sv. 62, s. 287–296.

[11] Leiva, A. *Kotlin for Android Developers*. Leanpub, 2017. ISBN 9781530075614.

[12] Maloney, J., Resnick, M., Rusk, N., Silverman, B. a Eastmond, E. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*. ACM New York, NY, USA. 2010, sv. 10, č. 4, s. 1–15.

[13] Pall, M. *Subject: Re: Lua on cell phones* [online]. 2021 [cit. 2021-18-12]. Dostupné z: http://lua-users.org/lists/lua-l/2007-11/msg00248.html.

[14] Pasternak, E., Fenichel, R. a Marshall, A. N. Tips for creating a block language with blockly. In: IEEE. *2017 IEEE Blocks and Beyond Workshop (B&B)*. 2017, s. 21–24.

[15] Pokress, S. C. a Veiga, J. J. D. MIT App Inventor: Enabling personal mobile computing. *ArXiv preprint arXiv:1310.2830*. 2013.

[16] Price, T. W. a Barnes, T. Comparing textual and block interfaces in a novice programming environment. In: *Proceedings of the eleventh annual international conference on international computing education research*. 2015, s. 91–99.

[17] Repenning, A. Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *J. Vis. Lang. Sentient Syst.* 2017, sv. 3, č. 1, s. 68–91.

[18] Resnick, M., Maloney, J., Monroy Hernández, A., Rusk, N., Eastmond, E. et al. Scratch: programming for all. *Communications of the ACM*. ACM New York, NY, USA. 2009, sv. 52, č. 11, s. 60–67.

[19] Selby, C. Four approaches to teaching programming. 2011.

[20] Svirca, Z. *Everything you need to know about MVC architecture* [online]. 2021 [cit. 2021-29-12]. Dostupné z: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1.

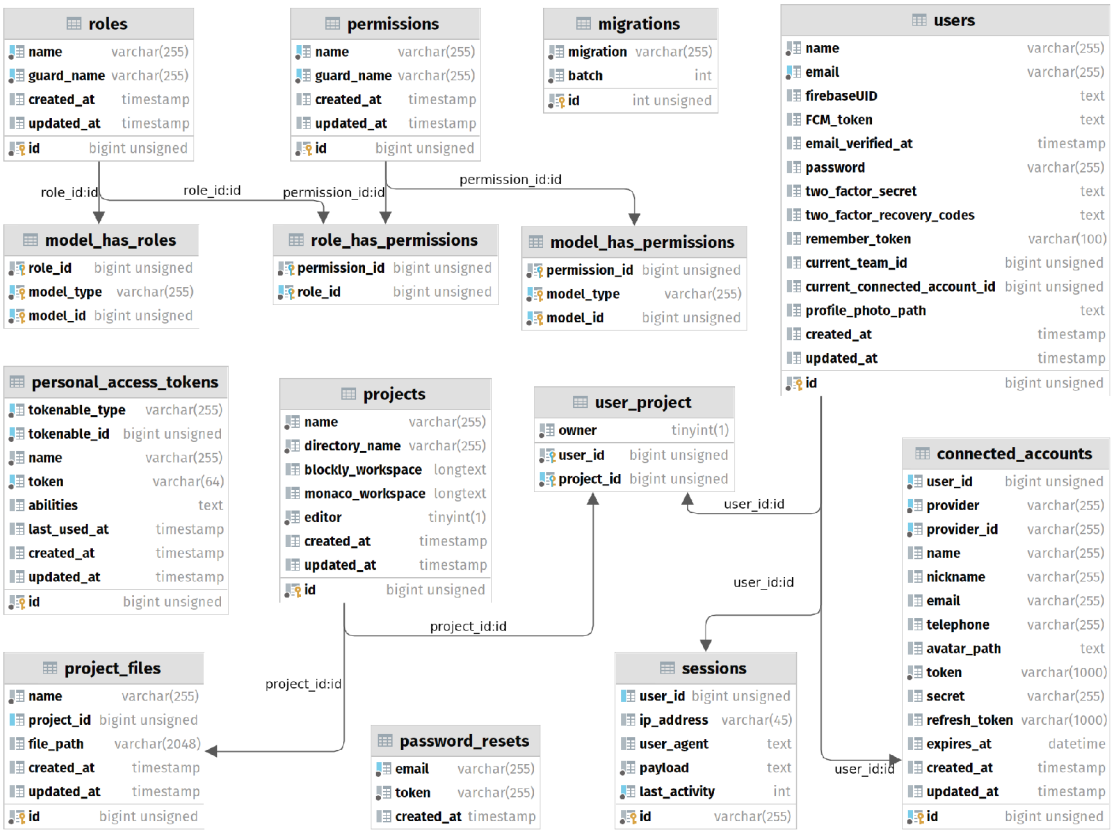
[21] Varma, J. *Learn Lua for iOS Game Development*. Apress, 2013. ISBN 9781430246633.

[22] Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., Cruz Alves, N. da et al. CodeMaster–Automatic Assessment and Grading of App Inventor and Snap! Programs. *Informatics in Education*. Vilnius University Institute of Mathematics and Informatics, Lithuanian .... 2018, sv. 17, č. 1, s. 117–150.

[23] Weintrop, D. a Wilensky, U. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*. ACM New York, NY, USA. 2017, sv. 18, č. 1, s. 1–25.

Příloha A

ER diagram databáze



Obrázek A.1: Finální ER diagram databáze.

Příloha B

Monaco editor

main.lua conf.lua Edit Blocks

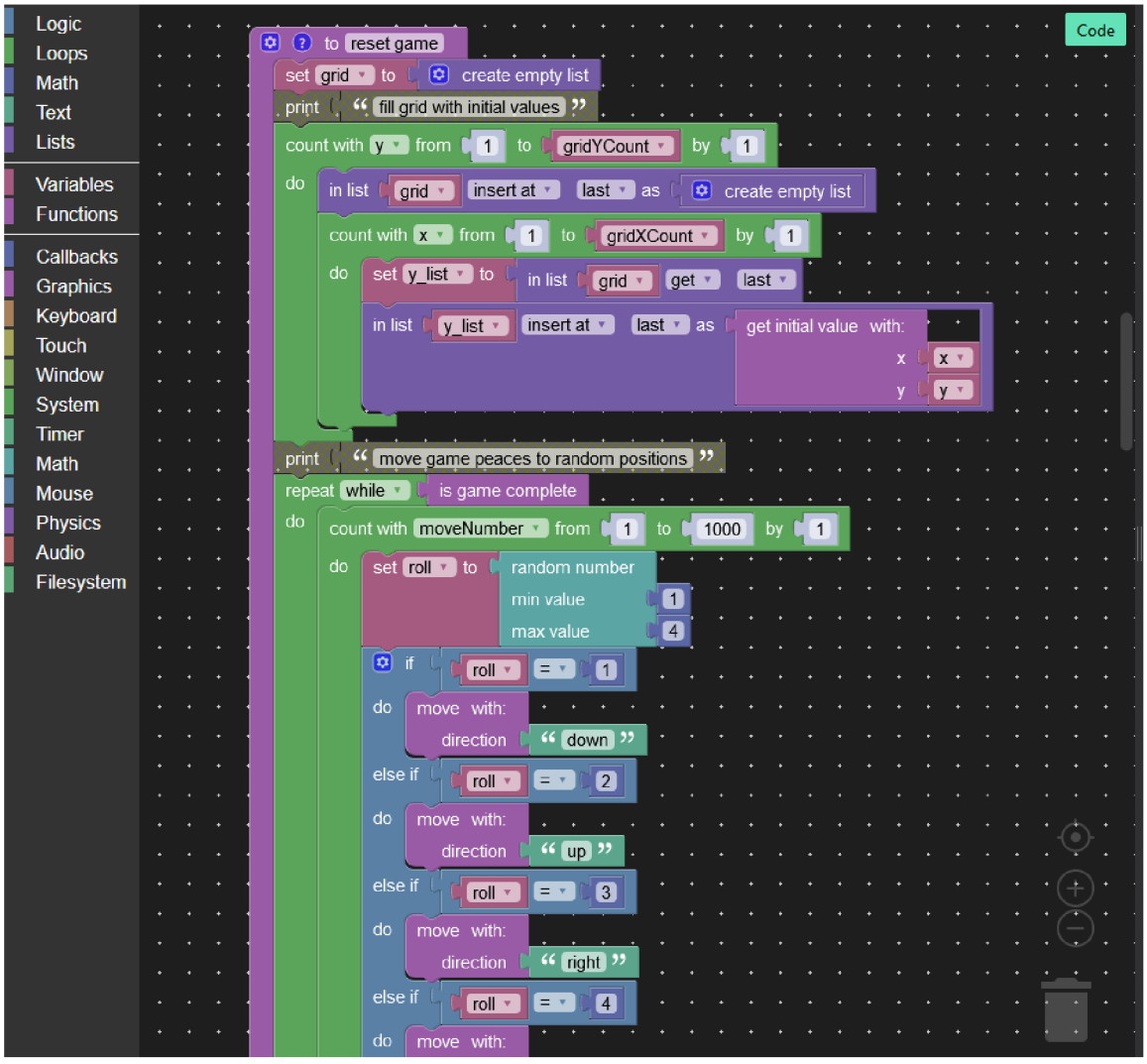
```lua
-- Describe this function...
function get_initial_value(x, y)
  return (y - 1) * gridXCount + x
end

-- Describe this function...
function move(direction)
  y_inc = 1
  if (1) > (gridYCount) then
    y_inc = -y_inc
  end
  for y = 1, gridYCount, y_inc do
    x_inc = 1
    if (1) > (gridXCount) then
      x_inc = -x_inc
    end
    for x = 1, gridXCount, x_inc do
      y_list = grid[y]
      if y_list[x] == gridXCount * gridYCount then
        emptyX = x
        emptyY = y
      end
    end
  end
  newEmptyX = emptyX
  newEmptyY = emptyY
  if direction == 'down' then
    newEmptyY = emptyY - 1
  elseif direction == 'up' then
    newEmptyY = emptyY + 1
  elseif direction == 'right' then
    newEmptyX = emptyX - 1
  elseif direction == 'left' then
    newEmptyX = emptyX + 1
  end
  if grid[newEmptyY] then
    new_empty_y_list = grid[newEmptyY]
    empty_y_list = grid[emptyY]
    if new_empty_y_list[newEmptyX] then
      j = new_empty_y_list[newEmptyX]
      i = empty_y_list[emptyX]
      empty_y_list[emptyX] = j
      new_empty_y_list[newEmptyX] = i
    end
```

Obrázek B.1: Uživatelské rozhraní s Monaco editorem.

Příloha C

Blockly editor



Obrázek C.1: Uživatelské rozhraní s Blockly editorem.