Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Integrace skriptovacího jazyka Lua do herního enginu

2015

Ondřej Procházka

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Ondřej Procházka
Název práce:	Integrace skriptovacího jazyka Lua do herního enginu
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	47
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Ondřej Procházka
Title:	Lua scritpting in game engine implemented in C
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	47
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Lua je skriptovací jazyk podporující funkcionální, objektově orientované a imperativní paradigma. Je dostatečně efektivní s nízkými nároky na hardware. Díky těmto vlastnostem se Lua stala populárním skriptovacím jazykem, který se využívá v mnoha oblastech vývoje. Jazyk Lua umožňuje propojení (integraci) s jazyky C, C++ a Java. V práci se zabývám postupem integrace mezi jazyky Lua a C, zejména exportem funkcí a struktur z C do Lua a voláním funkcí z Lua v C. Pomocí těchto mechanismů vytvořím s využitím knihovny SDL 2.0 jednoduchý herní engine v jazyku C, na kterém vystavím v jazyku Lua výslednou hru v izometrické grafice.

Synopsis

Lua is programming language designed as a scripting language and allowing development in multiple styles including object-oriented, functional or imperative paradigm. Language is also powerful enough despite its low hardware requirements. The above and other features made Lua very popular scripting language which is being used in many sectors of development. In this bachelor thesis I analyze integration between Lua and language C. Mainly export of functions and structures from language C to Lua and function call from Lua to C. With those mechanisms and with usage of library SDL 2.0 I have created simple game engine and built game with isometric graphics.

Klíčová slova: skriptovací jazyk; Lua; integrace; SDL 2.0; jazyk C, funkcionální, objektově orientovaný; herní engine

Keywords: scripting language; Lua; integration; SDL 2.0; language C; GUI; game engine; functional; objective oriented; compilation; bytecode; isometric; engine

Poděkování patří zejména mému vedoucímu bakalářské práce, Mgr. Petru Osičkovi, Ph.D, za jeho rady především při tvorbě textu. Dále děkuji rodině a všem, kteří mě podporovali.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **9**
1.1 Lua, C a SDL 2.0 . . . 9
1.1.1 Lua . . . 10
1.1.2 C . . . 10
1.1.3 SDL 2.0 . . . 10
1.2 Popis hry . . . 10

2 Integrace Lua do C **11**
2.1 Rychlokurz Lua . . . 11
2.1.1 Datové typy . . . 11
2.1.2 Moduly . . . 13
2.1.3 Interpret . . . 13
2.2 Základní integrace . . . 14
2.2.1 Inicializace Lua v C . . . 14
2.2.2 Export a import číselných a textových typů do nebo z globálního prostředí v Lua . . . 15
2.2.3 Export a import funkcí v globálním prostředí . . . 16
2.2.4 Manipulace se zásobníkem . . . 17
2.2.5 Export a import tabulek v globálním prostředí . . . 18
2.2.6 Jednoduchý listener Lua v C . . . 20
2.3 Vytvoření modulu pro Lua v C . . . 20
2.4 Tvoření tříd pro Lua v C . . . 21
2.5 Správa paměti Lua v C . . . 24
2.6 Práce s Lua kódy v C a ukázka kompletního programu . . . 26

3 SDL 2.0 a její principy **28**
3.1 Inicializace SDL 2.0 . . . 28
3.2 Grafika . . . 28
3.2.1 Inicializace . . . 29
3.2.2 Textura . . . 29
3.2.3 Renderer . . . 29
3.3 Audio . . . 29
3.4 Události, klávesnice a myš . . . 30

4 Architektura aplikace **31**
4.1 Tvorba knihovny ERPG v jazyku C a propojení s Lua . . . 31
4.1.1 Správa grafiky a vstupních událostí . . . 31
4.1.2 Zpráva zvuku . . . 32
4.1.3 Klávesnice a myš . . . 32
4.1.4 Pomocné funkce . . . 33
4.2 Použití knihovny ERPG v C a propojení s Lua při tvorbě hry . . . 33
4.3 Tvorba GUI v Lua . . . 34
4.3.1 Základní grafické elementy . . . 35

4.3.2 Události 35
4.4 Tvorba herního engine v Lua 36
4.4.1 Animace 36
4.4.2 Dlaždice (Tiles) 36
4.4.3 Statické objekty (Static objects) 37
4.4.4 Dynamické objekty (Dynamic objects) 37
4.4.5 Konzole 37
4.5 Vystavění hry na vybudovaném kódu 38

5 Uživatelská příručka **40**
5.1 Návod ke hře 40
5.2 Návod k editoru 42

Závěr **44**

Conclusions **45**

A Obsah přiloženého CD/DVD **46**

Literatura **47**

Seznam obrázků

1 Ukázka GUI. 35
2 Ukázka dlaždice růžová barva je průhledná 37
3 Ukázka použití konzole 38
4 Animace pavouka 38
5 Aktivace úkolu 39
6 Před spuštěním nové hry 40
7 Herní GUI 41
8 Interakce s truhlou 42
9 Editor GUI 43

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Lua – Syntaxe volání funkce 12
2 Lua – Syntaxe tvorby funkce 12
3 Lua – vytvoření modulu 14
4 Lua – použití modulu 14
5 C: Inicializace a ukončení Lua v C 15
6 Makefile 15
7 Export hodnot do globálního prostředí Lua 16
8 Import datových typů do C z globálního prostředí Lua 16
9 Export funkce 17
10 Import funkce do C 17
11 Tvorba a export tabulky 18
12 Import tabulky z Lua 19
13 Procházení tabulky v C 19
14 Listener Lua 20
15 Vytvoření modulu 21
16 Vytvoření třídy 22
17 Vytvoření třídy 22
18 Vytvoření třídy 23
19 Vytvoření třídy 23
20 Vytvoření třídy 23
21 Vytvoření třídy 24
22 Vytvoření třídy 25
23 Garbage collector 26
24 Vyhodnocení Lua skriptu 27

25 Lua – Jednoduchý Lua skript 27
26 Inicializace SDL2.0 28
27 Makefile SDL2.0 28
28 Použití knihovny ERPG 34

1 Úvod

Programy v nízkoúrovňovém jazyce se těžko udržují. Jedno z řešení, jak se vypořádat s tímto problémem, je integrace skriptovacího jazyka do nízkoúrovňového jazyka. Integrace je propojení dvou jazyků, díky němuž mezi sebou mohou sdílet funkce a hodnoty různých datových typů. Také lze vytvořit v nízkoúrovňovém jazyce knihovnu, a ve skriptovacím jazyce ji využívat (volat funkce a používat definice struktur).

Skriptovací jazyk je většinou dynamický. Tato vlastnost umožňuje programování aplikace takzvaně za běhu, což znamená, že proto abychom aplikaci mohli upravovat, nemusíme ji hned ukončovat. Výhodou u skriptovacího jazyka je navržení pro snadné ovládnutí a rychlý vývoj programu. Nevýhodou skriptovacích jazyků je obvykle nižší rychlost vykonávání kódu za cenu vysoké míry abstrakce.

Nízkoúrovňový jazyk je rychlý, má přístup k funkcím operačního systému. Většinou je nutná kompilace. Dále pak je v některých nízkoúrovňových jazycích nutnost starat se o alokaci a uvolňování paměti. Taková správa paměti je v některých případech výhodou, ovšem jednoduše může dojít k těžko dohledatelné a nebezpečné chybě.

Hry jsou specifické a náročné aplikace. Jeden z důvodů, proč jsou náročné je ten, že kód musí být rychlý a udržitelný, ovšem tyto vlastnosti jsou do jisté míry protichůdné. Kód hry musí být natolik rychlý, aby hra běžela plynule. Pro plynulost hry je vhodné 60 a více snímků za vteřinu. To znamená, že počítač má při 60 snímcích za vteřinu 16 milisekund na vykreslení jednoho snímku, což je vcelku málo. Z toho plyne, že je kladen důraz na rychlost, jsou nutné optimalizace kódu. Optimalizace často znepřehledňují kód a tím pádem se zhoršuje i udržitelnost kódu. Toto znesnadňuje rozšíření funkcionality aplikace, těžší opravování chyb a zhoršení flexibility. Tím je vývoj hry znesnadněn a zpomalen.

Rozhodl jsem se pro integraci skriptovacího jazyka Lua do nízkoúrovňového jazyka C. Jazyk Lua jsem zvolil s ohledem na rychlost, jednoduchost a podporu multiparadigmatového programování. Jako knihovnu, která mi umožnila lepší abstrakci nad nízkoúrovňovými funkcemi a nad OpenGL, jsem zvolil SDL 2.0. Tato knihovna podporuje několik platforem, zejména Mac OSX, Linux, Windows a je rychlá a použitelná při tvorbě her. Integraci budu demonstrovat na tvorbě hry typu RPG. Tedy hra na hrdiny ve stylu známého Diabla nebo Ultimy.

Pozornost budu věnovat především problematice integrace jazyka C se skriptovacím jazykem Lua. Zmíním knihovnu SDL 2.0 a představím základní architekturu aplikace. Na závěr popíšu problematiku tvorby hry.

1.1 Lua, C a SDL 2.0

Pro jádro enginu mé aplikace jsem zvolil, stejně jako vývojáři Lua, jazyk C. Zvolil jsem jej s ohledem na rychlost a pohodlnost integrace jazyka Lua. Pro nízkoúrovňové funkce jsem využil knihovny SDL 2.0, kde pro vykreslování používám abstrakci nad OpenGL.

1.1.1 Lua

Skriptovací jazyk Lua byl vyvinut v roce 1993. Vývojáři tohoto jazyka stále vydávají nové verze. Nejnovější verze je 5.2, která zde bude popisována, již se ale pracuje na verzi 5.3. Při vývoji jazyka Lua byl a stále je kladen důraz na rychlost, velikost a úsporu paměti. Interpret jazyka Lua má velikost kolem 180kB. Velikost přináší některá omezení například u regulárních výrazů, které jsou ochuzené o podmínky `or` a `and`.

Díky svým výhodám, zejména s ohledem na paměťovou nenáročnost, se často vyskytuje v mikropočítačích jako skriptovací jazyk. Lua najdeme v několika herních projektech, z těch nejznámějších například World of Warcraft, Ultima online a také Infinity engine, na kterém běží Icewind Dale, Baldur's gate a další.

Z Lua vzniklo několik dialektů, ty nejznámější jsou LuaJIT, která se používá při integraci s jazykem Java, dále MetaLua, což je nedokončený projekt, který se nachází v alfa verzi. Tento projekt se snaží zakomponovat makra do jazyka Lua.

1.1.2 C

Programovací jazyk C vznikl na počátku 70. let 20. století. Je to jazyk nízkoúrovňový, kompilovaný a vcelku minimalistický. Pro kompilaci se dá využít několik standardů. Já používám standard C99. Jazyk umožňuje zápis assembleru přímo do C. Uživatel jazyka C se musí starat o alokaci a dealokaci paměti. Jazyk C je rychlý a umožňuje, díky množství knihoven, přístup k systémovým funkcím.

1.1.3 SDL 2.0

SDL 2.0 v plném znění Simple DirectMedia Layer je knihovna, která poskytuje jednoduché rozhraní, nad multimediálními nízkoúrovňovými funkcemi. Uživateli poskytuje základní abstrakci nad grafickými funkcemi, které k zobrazovaní používá OpenGL, přičemž výpočty probíhají na grafické kartě. Lze také použít softwarové vykreslování nebo obyčejné OpenGL. Zde budeme uvažovat SDL 2.0 abstrakci nad OpenGL, jelikož je pro tvorbu plánované hry dostačující. SDL 2.0 je pod distribucí zlib. Tato licence umožňuje volné užívání knihovny v jakémkoliv softwaru.

1.2 Popis hry

Hra bude vykreslována izometricky ve 2D pod úhlem 45°. Bylo nutné vystavět pokročilejší GUI systém, s ohledem na množství složitých dialogů. Dále jsem implementoval ukládání hry, načítání, tvoření map, pohyb po mapě, umělou inteligenci postav, systém pro tvoření úkolů a základní mechanismy soubojových systémů.

2 Integrace Lua do C

Skriptovací jazyk Lua umožňuje začlenit Lua skripty do jazyka C a obecně propojit Lua kód s jazykem C. Tímto způsobem lze vykonávat skripty ve více vláknech.

Lua také umožňuje připojit C knihovny do Lua kódu, pokud splňuje určité požadavky. Tudíž Lua může běžet samostatně a být zároveň propojená s jazykem C.

2.1 Rychlokurz Lua

Lua je dynamicky typovaný jazyk s lexikálním rozsahem platnosti. Používá infixovou notaci. Podporuje více paradigmat například funkcionální, objektové a imperativní. Hlavní nevýhoda jazyka Lua je, že nepodporuje makra. Lua se stará sama o správu paměti.

2.1.1 Datové typy

Lua obsahuje několik základních typů:

- nil
- pravdivostní hodnoty (boolean)
- textový řetězec (string)
- číslo (number)
- funkce (function)
- tabulka (table)
- metatabulka (metatable)
- korutina (coroutine)
- userdata

Hodnotu **nil** má každá neinicializovaná proměnná. Lua automaticky při potřebě převádí **číslo**, na textový řetězec a **textový řetězec**, pokud lze převést, na číslo.

Funkce jsou elementy prvního řádu, což umožňuje značnou flexibilitu při programování. Například uživateli je umožněno implementovat uzávěry a generické funkce. Dále pak Lua dovoloje tvorbu rekurzivních funkcí. Funkce může vracet libovolný počet argumentů, pokud nevrací žádný automaticky vrácí `nil`. Při tvorbě funkce lze mít nespecifikovaný počet argumentů. Při volání funkce se může zadat libovolný počet vstupních argumentů. Syntaxe tvorby funkce 2 a její volání 1. Funkce bude mít n vstupních argumentů a m výstupních argumentů.

```
1 ar1, arg2, ... ,argM = name(arg1,arg2, ... , argN)
```

Zdrojový kód 1: Lua – Syntaxe volání funkce

```
1 function name (arg1,arg2, ... , argN) -- name - jméno funkce
2 <body> -- Kód funkce
3 end
```

Zdrojový kód 2: Lua – Syntaxe tvorby funkce

Tabulky jsou hierarchická struktura, kterou Lua používá pro data a zároveň pro program. Pole v tabulkách je číslováno od 1. V tabulce každý klíč, který nemá inicializovanou hodnotu vrací `nil`.

Tvorba tabulky:

- `{}` – Vyhodnotí se na tabulku.
- `{["1"] = 5,...}` – Vyhodnotí se na tabulku s klíčem 1, který obsahuje hodnotu 5.
- `{ ahoj = 5,...}` – Vyhodnotí se na tabulku s klíčem ahoj, obsahuje hodnotu 5. V tomto zápisu nesmí být jako klíč číslo."1"
- `{"ahoj",5,6,...}` – Vyhodnotí se na tabulku. Pod klíčem 1 hodnota "ahoj", pod klíčem 2 hodnota 5, pod klíčem 3 hodnota 6.

Přiřazení do tabulky nebo vypsání hodnoty pod klíčem key.

- `tabulka = {}`
- `tabulka["key"] = 8` – Přiřazení hodnoty 8 za klíč "key"v tabulce.
- `tabulka.key` – Vrácení hodnoty pod klíčem key.
- `tabulka["key"]` – Vrácení hodnoty pod klíčem key.

Tabulky se chovají jako pole známé z C a nebo jako asociativní pole. Lua pomocí rozhodovacích algoritmů mění ve své paměti strukturu tabulky. Rozhodování je řízeno uloženými hodnotami. Pokud jsou klíče celočíselného typu, pak se tabulka reprezentuje polem, zde je přístup k prvku v konstantním čase. Ve všech ostatních případech je uvnitř implementována pomocí B-stromu, kde operace vyhledání prvku je v logaritmickém čase.

Odvozeným datovým typem je **metatabulka**. Metatabulka je tabulka, uložená ve speciálním klíči, se kterým se pracuje výhradně pomocí funkcí `setmetatable` a `getmetatatble`. Na metatabulku se dá nahlížet obdobně jako na implementaci interface z objektově orientovaných jazyků. Díky tomuto

přístupu lze přetěžovat operátory a uživatelsky uvolňovat prostředky. Pro vytvoření metatabulky se musí nejprve vytvořit tabulka. Při tvorbě tabulky, která později bude metatabulkou, se využívá metaklíčů (speciální klíče). Na metaklíče lze napojit metametody, tedy funkce nebo i jiné datové typy. Tyto metametody se aplikují za určitých okolností, například před zničením tabulky garbage collectorem. Pro vytvoření metatabulky z tabulky a napojení na jinou tabulku slouží funkce `setmetatable`. Pro vracení metatabulky s tabulky je funkce `getmetatable`.

Samotná Lua neumožňuje vícevláknové programování. Pro simulaci více vláken používá **korutiny**, při kterých nenastávají problémy jako u vícevláknového programování, protože běží vždy jen jedno vlákno.

Userdata jsou objekty, které si Lua drží ve své paměti. Tyto objekty lze vytvořit jen z jazyka C. Později bude vysvětleno.

2.1.2 Moduly

Lua pro udržování kódu zvolila moduly. Moduly představují možnost, jak rychle zakomponovat určitou funkčnost do Lua bez deformace globálního prostředí. Modul je tabulka, která obsahuje funkce a proměnné, ke kterým má mít uživatel přístup. Lua obsahuje pět základních modulů:

- Math – matematický modul
- Debug – debugovací modul
- Coroutine – modul pro korutiny
- String – modul pro textové řetězce
- Table – modul pro práci s tabulkami

Kód číslo 3 demonstruje vytvoření modulu v Lua. Tento modul bude mít v sobě funkci `print`, která vypíše předaný textový řetězec. Také je v kódu ukázáno vytvoření tabulky `x`, na kterou se napojí metatabulka `mtable`. Při zničení garbage collectorem tabulky `x` se do konzole vypíše „Jsem zničen“.

V kódu 4 je ukázka načtení modulu a vyhodnocení vytvořené funkce `print`.

2.1.3 Interpret

Po spuštění interpretu Lua se vytvoří globální prostředí. Globální prostředí je implicitně uloženo v proměnné `_ENV`. Globální prostředí je tabulka, kde jsou uloženy všechny globální funkce a proměnné. S prostředím může uživatel libovolně manipulovat. Automaticky se prefix `_ENV.` přidává na začátek každého přístupu ke globální funkci a proměnné. Lokální proměnné a funkce se definují pomocí klíčového slova `local`. Pokud klíčové slovo `local` není uvedeno, automaticky se proměnná vytvoří jako globální. Lua si ukládá lokální data do speciálního prostředí. Je k nim umožněn přístup pomocí modulu `debug`.

```lua
local module = {} -- Vytvoření lokální tabulky

local x = {["value"]=5}
local mtable={["__gc"] = function (self)
 print("Jsem zničen")
 end
} --Tento příkaz vytvořil tabulku s~klíčem __gc který obsahuje
   funkci.

setmetatable(x,mtable)  -- Tímto je navázána tabulka mttable na x
   a tím z~mttable se stala metatabulka. Před uvolněním garbage
   collectorem se vypíše na obrazovku Jsem zničen díky navázání na
    gc pomocí klíče "__gc".
x=nil
function module.print(string) -- Vytvoření funkce v~tabulce modulu
   , která vytiskne na obrazovku string.
 print(string)
end

return module -- Vracení tabulky.
```

Zdrojový kód 3: Lua – vytvoření modulu

```lua
Module = require("module") -- načtení modulu. Zpracuje soubor a
   vrací hodnotu, kterou jsme vrátili na konci souboru. Čili
   tabulku module.
Module.print("Hello World")
print(Module.nic) -- při přístup do tabulky k~prvku, kde není
   inicializovaná hodnota nám vypíše nil.
```

Zdrojový kód 4: Lua – použití modulu

2.2 Základní integrace

Lua poskytuje v C rozhraní, které dává uživateli nad Lua velkou moc. Pomocí tohoto rozhraní lze vytvářet několik Lua stavů. Lua stav si udržuje potřebné informace pro běh interpretu Lua. Dále pak lze vytvářet moduly a třídy. Také je umožněno posílat mezi Lua a C hodnoty všech datových typů. V C lze procházet tabulky, vytvářet tabulky a metatabulky. Z jazyka C je možné udělat vše co v Lua. Základní možnosti integrace si v následujících kapitolách ukážeme. Ukázky kódu v dalších kapitolách jsou v jazyce C.

2.2.1 Inicializace Lua v C

Chce-li uživatel integrovat Lua do C musí nejprve vytvořit v jazyku C Lua stav, který si udržuje data a zásobník. Zásobník je struktura, se kterou může uživatel manipulovat z jazyka C. Umožňuje přijímání a odesílání dat mezi Lua a C.

V zásobníku lze přistupovat k libovolné pozici. Indexování zásobníku je následující. Pomocí indexu s prefixem - tedy `-index` se odkazuje na index od vrcholu zásobníku například `-1` se odkáže na vrchol zásobníku a `-2` na druhý prvek od vrcholu zásobníku. S kladným indexem se odkazuje ode dna zásobníku například `1` ukazuje na dno zásobníku, `2` druhý prvek ode dna zásobníku. Uživatel může vytvořit více Lua stavů, které mohou běžet ve více vláknech a sdílet přitom globální prostředí.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<lua5.2/lua.h>
4  #include<lua5.2/lauxlib.h>
5  #include<lua5.2/lualib.h>
6  int main()
7  {
8    lua_State * L = luaL_newState(); //vytvoření vlákna L
9    luaL_openlibs(L); //vložení základních modulů do L
10   //zde se nachází kód který pracuje s~Lua
11   lua_close(L); //korektní ukončení L
12 }
```

Zdrojový kód 5: C: Inicializace a ukončení Lua v C

Při kompilaci je dobré využít Makefile.

```
1  CFLAGS=-Wall -std=c99 -shared -g -O0
2  CLIBS= -llua5.2 -lm
3  OUT_DIR=./
4  OUT=test
5  OBJS=*.o
6  CC=gcc
7
8    $(CC) $(CFLAGS) -c *.c
9    $(CC) $(OBJS) -o ${OUT_DIR}$(OUT) $(CLIBS)
```

Zdrojový kód 6: Makefile

2.2.2 Export a import číselných a textových typů do nebo z globálního prostředí v Lua

Jako první a nejjednodušší způsob představím export hodnot základních datových typů z C do globálního prostředí v Lua. Lua rozhraní v C uživateli poskytuje funkce pro vložení hodnot základních datových typů na zásobník. Dále pak se nabízí funkce, která hodnotě z vrcholu zásobníku přiřadí klíč v tabulce globálního prostředí. Klíče v globální tabulce lze nazývat globální proměnná. Při exportu

datového typu char si Lua automaticky uvnitř své paměti vytvoří kopii stringu.

```
1  lua_pushnumber(L, 5); // Přidání čísla na zásobník.
2  lua_setglobal(L,"_Number"); // Přiřazení vrcholu zásobníku na klíč
      _Number.
3
4  lua_pushstring(L, "text"); // Přidání stringu na zásobník. V~případě
      neúspěchu vrací NULL a na vrchol zásobníku přidá nil.
5  lua_setglobal(L, "_Text");
```

Zdrojový kód 7: Export hodnot do globálního prostředí Lua

Import hodnot z globálního prostředí probíhá analogicky. Nejdříve se musí najít prvek z tabulky v globálním prostředí a pak je zapotřebí funkce, která prvek odebere ze zásobníku a vrátí jeho hodnotu. U čísel funkce vrací strukturu lua_Number, která obsahuje desetinné číslo. Lze je přetypovat na klasické typy v C se zachováním čísla např. (float) (double) (int). U stringů se vrací const char. Aby uživatel mohl libovolně manipulovat s vrácenou hodnotou nebo si ji uchovávat, musí se string překopírovat do vlastní paměti v C. Důvod je prostý, Lua si může v sobě kdykoliv tento string uvolnit a ve chvíli kdyby ho program v C vyžádal, nastal by pravděpodobně neoprávněný přístup do paměti.

```
1  lua_getglobal(L,"_Number"); // Přidá na zásobník obsah prvku tabulky
      pod klíčem _Number.
2  int n = 0;
3  //Pomocí argumentu -1 se odkazujeme na vrchol zásobníku.
4  if(lua_isnumber(L, -1)) //Kontrola pro zjištění jestli je typ number
      .
5    n = (int)lua_tonumber(L,-1); // Vrácení prvku z~vrcholu zásobníku.
6  // Použitím funkce luaL_checknumber(L, -1) dojde ke kontrole a zá
      roveň vrátí hodnotu ze zásobníku.
7  lua_pop(L,1); //Odebrání prvku z~vrcholu zásobníku.
8
9  lua_getglobal(L,"_Text") ;
10 const char * text = lua_tostring(L, -1);
11 lua_pop(L,1);
```

Zdrojový kód 8: Import datových typů do C z globálního prostředí Lua

2.2.3 Export a import funkcí v globálním prostředí

Zásady pro exportování funkce z C do Lua jsou následující. Funkce musí mít jeden argument typu lua_State. Druhý požadavek je, že funkce musí mít návratový typ int, kterým vrací počet návratových argumentů do Lua. Lua předává argumenty do funkce pomocí zásobníku, kde na dně zásobníku se nachází první

argument. K prvnímu argumentu se přistupuje číslem 1, k druhému argumentu číslem 2 a tak dále. Po skončení funkce se argumenty odstraní ze zásobníku.

```
1  int Lua_multiply(lua_State * L)
2  {
3   int first = luaL_checknumber(L, 1); //Vrátí hodnotu prvního
       argumentu funkce.
4   int second = luaL_checknumber(L, 2);//Druhá hodnota argumentu.
5   int result = first * second;
6
7   lua_pushnumber(L, result); //Vložení na zásobník výsledku.
8   return 1; //Informujeme Lua že vracíme jeden argument, tím Lua vrá
       tí první prvek z~vrcholu zásobníku. Po skončení funkce se
       automaticky prvky odeberou ze zásobníku.
9  }
10
11 lua_pushcfunction(L, Lua_multiply); //Přidá se na vrchol zásobníku
     funkce.
12 lua_setglobal(L, "multiply);
```

Zdrojový kód 9: Export funkce

Dále lze exportovat funkci z Lua do C. Lua rozhraní v C poskytuje funkci, která dokáže vyhodnotit Lua funkci v C. Před aplikací této funkce se musí na zásobník vložit Lua funkce a potom argumenty pro Lua funkci. První vložený argument na zásobníku je v Lua funkci jako vstupní argument první, druhý jako druhý atd. Parametry vyhodnocovací funkce jsou počet argumentů a počet návratových argumentů v Lua funkci. Po vyhodnocení se na vrchol zásobníku vloží návratové hodnoty. Pořadí návratových argumentů bude následující, první návratový argument se vloží jako první, druhý jako druhý atd.

```
1 lua_getglobal(L,"multiply"); //Vloží na vrchol zásobníku funkci
    multiply.
2 lua_pushnumber(L,5);
3 lua_pushnumber(L,10);
4 lua_call(L,2,1); //Zavolání vyhodnocovací funkce se 2 argumenty,
    která vrací jednu hodnotu.
5 int n = (int)luaL_checknumber(L,-1);
```

Zdrojový kód 10: Import funkce do C

2.2.4 Manipulace se zásobníkem

Lua rozhraní v C obsahuje několik užitečných funkcí pro manipulaci se zásobníkem.

- `lua_pushnil(L)` Vloží na vrchol zásobníku `nil`.

- `lua_gettop(L)` Vrací číslo vrcholu zásobníku.
- `lua_insert(L,index)` Prvek z vrcholu zásobníku přemístí na pozici index. Prvky se posunou o jednu pozici blíže k vrcholu.
- `lua_pushvalue(L, index)` Vytvoří kopii prvku z vrcholu zásobníku. Kopii přidá na vrchol zásobníku.
- `lua_remove(L, index)` Odstraní prvek na indexu ze zásobníku. Prvky za hodnotou index, blíže k vrcholu zásobníku, se posunou o jednu pozici směrem k odstraňovanému indexu.
- `lua_replace(L, index)` Prvek z vrcholu zásobníku se přesune a nahradí prvek na pozici index.
- `lua_pop(L,n)` Odstraní počet prvků n z vrcholu zásobníku.

2.2.5 Export a import tabulek v globálním prostředí

Lua používá pro data tabulky. Uživatel s pomocí několika funkcí má možnost vytvořit tuto strukturu v C a následně ji exportovat. V následujícím příkladě vytvořím strukturu pro vektor v C a exportuji ji do Lua pomocí tabulky.

```
1  typedef struct Vector{
2    int x;
3    int y;
4  }Vector;
5
6  Vector vec;
7  vec.x = 10;
8  vec.y = 25;
9
10 lua_newtable(L); //Na vrchol zásobníku přidá tabulku.
11
12 lua_pushstring(L, "x"); //Vložíme string na vrchol zásobníku.
13 lua_pushnumber(L, vec.x); //Vložíme číslo na vrchol zásobníku.
14 lua_settable(L, -3); //Tato funkce nám do tabulky, která je na 3
       pozici od vrcholu zásobníku, přiřadí jako klíč vrchol zásobníku
       a jako hodnotu prvek na pozici pod vrcholem zásobníku.
15 //tabulka = {["x"]=10}
16 lua_pushstring(L, "y");
17 lua_pushnumber(L, vec.y);
18 lua_settable(L, -3);
19 //tabulka {["x"]=10, ["y"] = 10}
20 lua_setglobal(L, "_Vector"); //Přiřadili jsme tabulku, která je na
       vrcholu zásobníku, do klíče _Vector.
```

Zdrojový kód 11: Tvorba a export tabulky

Import tabulky z Lua probíhá analogicky. Nyní představím způsob, kdy uživatel zná dopředu klíče. Opět na struktuře `Vector`. Později demonstruji iteraci přes neznámé klíče a přes pole.

```
1  lua_getglobal(L,"_Vector");
2  lua_pushstring(L, "x"); //Vloží se na vrchol zásobníku klíč k~prvku
      v~tabulce.
3  lua_gettable(L, -2); //Funkce vezme z~vrcholu zásobníku klíč k~prvku
       v~tabulce na indexu.
4  vec.x = (int)luaL_checknumber(L, -1);
5  lua_pop(L,1); //Odebrání hodnoty z~vrcholu zásobníku.
6
7  lua_pushstring(L, "y");
8  lua_gettable(L, -2);
9  vec.y = (int)luaL_checknumber(L, -1);
10 lua_pop(L,1);
```

Zdrojový kód 12: Import tabulky z Lua

Rozhraní Lua pro C umožňuje iterovat přes prvky v tabulce, bez nutnosti klíče. Toho se často využívá při procházení pole. Po projití všech prvků v tabulce, vrací funkce `nil` na vrchol zásobníku.

```
1  int i = 0;
2  lua_newtable(L);
3  for(i=0; i<10;i++){
4    lua_pushinteger(L,i);
5    lua_pushinteger(L, i*3);
6    lua_settable(L, -3);
7  }
8  lua_setglobal(L, "_MyTable");
9
10 lua_getglobal(L,"_MyTable"); //Na vrchol zásobníku se přidá tabulka.
11 lua_pushnil(L);//Přidáme na vrchol zásobníku nil, protože je to počá
      teční hodnota pro počátek vyhledávání.
12 while(lua_next(L,-2) != 0){
13 //Vezme z~vrcholu zásobníku poslední klíč a na indexu -2 kde je
      tabulka postoupí na další klíč v~případě, že již prošel všechny
      vrátí 0.
14   printf("Klíč: %d, Hodnota: %d \n", (int)luaL_checknumber(L,-2), (
        int)luaL_checknumber(L, -1));
15   lua_pop(L,1);
16 }
```

Zdrojový kód 13: Procházení tabulky v C

2.2.6 Jednoduchý listener Lua v C

Listener je program, který slouží k vyhodnocení kódu ze vstupu. Lua rozhraní v C uživateli poskytuje funkci `dostring`, která vytvoří z textového řetězce funkci. Tato funkce se pak musí aplikovat pro vyhodnocení vstupu. V následujícím příkladu bude představen listener v C. Tento listener umožní načíst vždy 1000 znaků ze vstupu. Při výskytu chyby se vypíše podrobné hlášení o chybě.

```
1  int main()
2  {
3   lua_State* L = luaL_newstate ();
4   luaL_openlibs(L);
5   char * string = (char*)malloc(sizeof(char)*1001);
6   while( 1 ){
7   printf(">> ");
8   fgets(string,sizeof(char)*1000,stdin);
9
10    lua_getglobal(L, "debug"); // Modul debug na zásobník
11    lua_getfield(L, -1, "traceback"); // Přidání na zásobník funkce
         debug.traceback
12    lua_remove(L, -2); // Odstranění ze zásobníku tabulky debug
13   luaL_loadstring(L,string);// Vytvoření funkce ze stringu a vložení
        na zásobník
14   if( lua_pcall(L,0,0,1) ){ // Zavolání nově vytvořené funkce v~pří
       padě chyby vypíše se chybové hlášení
15     printf("%s", string);
16       printf("runtime error: %s \n", lua_tostring(L, -1));
17       lua_pop(L,1);
18   }
19   }
20   free(string);
21   lua_close(L);
22   return 0;
23 }
```

Zdrojový kód 14: Listener Lua

2.3 Vytvoření modulu pro Lua v C

V Lua se pro strukturovanost kódu využívá takzvaných modulů. V jazyce C, kde je tvoření modulů pro Lua umožněno, je tento proces následující. Nejprve se vytvoří funkce, které by měl modul obsahovat. Potom se tyto funkce zaregistrují do modulu, vytvoří se modul a nakonec se tento modul musí otevřít pro Lua. Rozhraní Lua poskytuje uživateli strukturu pro jednoduché zaregistrování funkce. Tato struktura obsahuje jako první prvek jméno funkce v Lua a druhý prvek je pointer na funkci. Na příkladu demonstruji jak díky této struktuře jednoduše uživatel zaregistruje funkce. Vytvořím 2 funkce pro sčítání a odčítání. Poté je zaregistruji do modulu, který pojmenuji `Example`.

```
int Lua_add(lua_State * L)
{
  lua_pushnumber(L,  luaL_checknumber(L,1)+luaL_checknumber(L,2));
  return 1;
}
int Lua_minus(lua_State * L)
{
  lua_pushnumber(L,  luaL_checknumber(L,1)- luaL_checknumber(L,2));
  return 1;
}

int luaopen_example(lua_State * L)
{
  // Vytvoření pole struktury luaL_Reg, kde na první pozici je jméno funkce v~Lua a na druhé pointer na funkci v~C.
  struct luaL_Reg driver[] = {
    {"add", Lua_add},
    {"minus", Lua_minus},
    {NULL,NULL}
  };

  luaL_newlib(L, driver); //Vytvoření tabulky která si do sebe uloží klíč jako jméno funkce a hodnotu jako C funkci.

  return 1; //Vrácení nově vytvořené tabulky tedy modulu.
}

luaL_requiref(L,"Example", luaopen_example,1); //Vytvoření modulu Example v~Lua. Vytvoří se vazba na tabulku, kterou jsme vytvořili v~předchozí funkci, se jménem Example.
```

Zdrojový kód 15: Vytvoření modulu

Volání funkce, která je umístěna v modulu je stejné jako volání funkce, která je umístěna v tabulce.

2.4 Tvoření tříd pro Lua v C

Tvorba tříd pro Lua v C je jedna z nejužitečnějších věcí při integraci. Umožní uživateli dostatečně velkou abstrakci nad kódem, který se bude používat v Lua. Pomocí této metody může uživatel jednoduše vytvářet objekty, které budou mít své metody a proměnné. Také lze doplnit o dědičnost. Umožňuje přetížit operátory a je možnost napojení na garbage collector. Pro využívání těchto funkcí se musí navázat metatabulka na tabulku, která reprezentuje objekt. Metatabulka obsahuje metametody, na které může programátor navázat funkce. Objekt vytvořený v C nelze nijak modifikovat z Lua. Modifikace probíhá pouze používáním C metod.
Výpis důležitých metametod pro objekty:

- `__index`: Při přístupu na klíč v objektu se interpret nejdříve podívá na klíč v objektu. Pokud klíč neexistuje, tak zavolá funkci navázanou na `__index`. Jako první argument je objekt, druhý klíč a třetí hodnota. Lze také navázat tabulku, přičemž se přistoupí k hledanému klíči v tabulce, pokud existuje vrací se jeho hodnota.

- `__newindex`: Při vytváření nového klíče se zavolá funkce, která je navázána na `__newindex`. První argument je objekt, druhý klíč a třetí hodnota

- `__tostring`: Při nutnosti převedení objektu na string se aplikuje metametoda `__tostring`.

- `__gc`: Při navázání na garbage collector, kde první argument je objekt, se aplikuje metametoda `__gc`.

- `__eq`: Při porovnávání, kde první argument je objekt nalevo od operátoru == a druhý argument objekt napravo se aplikuje metametoda `__eq`.

- operátory které lze přetížit `__unm`: unární mínus, `__add`: sčítání, `__sub`: odčítání, `__mul`: násobení, `__div`: dělení, `__mod`: modulo, `__pow`: mocnina, `__concat`: spojení stringů, `__lt`: menší nebo větší, `__le` menší rovná se nebo větší rovná se.

Pomocí těchto vlastností lze vytvořit i read-only objekt.

Na příkladu demonstruji vytvoření třídy. Objekt vytvořený z této třídy bude navázán na garbage collector. Dále pak bude obsahovat přetíženou operaci pro sčítání, implementovanou metodu pro `tostring`, konstruktor a jednu metodu. Nejprve se vytvoří v C struktura pro vektor.

```
1   typedef struct Vector{
2     int x;
3     int y;
4   }Vector;
```

Zdrojový kód 16: Vytvoření třídy

Dále je standardním postupem vytvoření funkce, která rozpozná, jestli se jedná o typ objektu vektor.

```
1 Vector* Lua_check_vector(lua_State * L, int i)
2 {
3   return luaL_checkudata(L, i, "Vector");
4 }
```

Zdrojový kód 17: Vytvoření třídy

Poté se musí vytvořit funkce pro konstruktor. V ní alokujeme paměť pro objekt, kterou si bude udržovat Lua v paměti. Na objekt navážeme metatabulku `Vector`, která bude vytvořena po zaregistrovaní modulu. Tvorba této tabulky je ve funkci `luaopen_vector`.

```
1 int vector_new(lua_State * L){
2   int x = (int)luaL_checknumber(L,1);
3   int y = (int)luaL_checknumber(L,2);
4   Vector * vec = lua_newuserdata(L, sizeof(Vector)); //Vytvoření
      datového typu userdata. Ukazatel na strukturu, který si drží v~
      paměti Lua. Aplikací této funkce budou userdata na vrcholu zá
      sobníku.
5   vec->x = x;
6   vec->y = y;
7   luaL_setmetatable(L, "Vector"); //Napojení metatabulky Vector na
      objekt.
8   return 1
9 }
```

Zdrojový kód 18: Vytvoření třídy

Funkce, která bude navázána na garbage collector, před uvolněním objektu se zavolá.

```
1 int vector_gc(lua_State * L){
2   printf("Vector je v~garbage collectoru\n");
3   return 0;
4 }
```

Zdrojový kód 19: Vytvoření třídy

Nyní funkce na převedení objektu na string.

```
1 int vector_tostring(lua_State * L){
2   Vector * v~= Lua_check_vector(L,1);
3   lua_pushstring(L, "x: ");
4   lua_pushnumber(L, v->x);
5   lua_pushstring(L, " y: ");
6   lua_pushnumber(L, v->y);
7   lua_concat(L,4);
8   return 1;
9 }
```

Zdrojový kód 20: Vytvoření třídy

Funkce vector_add, která bude navázána na operaci sčítání, sečte dva vektory a z výsledku vytvoří nový vektor. Funkce set_zero bude sloužit jako demonstrativní metoda. Nastavuje prvky vektoru na nula.

```
1  int vector_add(lua_State * L)
2  {
3   Vector * v1 = Lua_check_vector(L,1);
4   Vector * v2 = Lua_check_vector(L,2);
5   int x = v1->x + v2->x;
6   int y = v1->y + v2->y;
7   Vector * vec = lua_newuserdata(L, sizeof(Vector));
8   vec->x = x;
9   vec->y = y;
10  luaL_setmetatable(L,"Vector");
11  return 1;
12 }
13 int vector_set_zero(lua_State * L)
14 {
15  Vector * v~= Lua_check_vector(L,1);
16  v->x = 0;
17  v->y = 0;
18  return 0;
19 }
```

Zdrojový kód 21: Vytvoření třídy

Nakonec uživatel musí zaregistrovat metody, konstruktor a napojit funkce na metametody. Zde se musí vytvořit metatabulka Vector, která byla zmíněna výše u rozpoznávání objektu. Po zaregistrování této třídy, podobně jako u modulu, může uživatel tuto třídu využívat. Tuto část demonstruje kód číslo 22.

2.5 Správa paměti Lua v C

O správu paměti v Lua se stará garbage collector, který je typu mark-and-sweep. Lua bohužel neumožňuje garbage collector navázat na jiné vlákno, které by se staralo o uvolňování paměti. Tento problém se řeší, pokud je nežádoucí náhodné zasekávání, pomocí funkce lua_gc. Tato funkce má několik možných nastavení. Nastavení se předávají jako makra ve druhém parametru funkce. Třetí parametr v této funkci slouží jako dodatečný argument k nastavení. Pokud nastavení nepotřebuje dodatečný argument, je zvykem dosadit 0.
Možnosti nastavení pro funkci lua_gc:

- lua_GCSTOP: Slouží k zastavení garbage collectoru. Pro spuštění lze použít lua_GCRESTART, lua_GCCOLLECT, lua_GCSTEP
- lua_GCRESTART: Restartuje garbage collector, takže garbage collector začne vyhledávání od znovu.

```c
int luaopen_vector(lua_State * L)
{
  struct luaL_Reg method[] = {
    {"set_zero", vector_set_zero},
    {NULL, NULL}
  };
  struct luaL_Reg constructor[] = {
    {"new", vector_new},
    {NULL, NULL}
  };

  luaL_newmetatable(L, "Vector"); //Vytvoření globální metatabulky
      Vector.
  luaL_newlib(L, method);
  lua_setfield(L, -2,"__index");
  lua_pushcfunction(L, vector_add);
  lua_setfield(L, -2,"__add");
  lua_pushcfunction(L, vector_tostring);
  lua_setfield(L, -2,"__tostring");
  lua_pushcfunction(L, vector_gc);
  lua_setfield(L, -2,"__gc");
  luaL_newlib(L, constructor);

  return 1;
}

luaL_requiref(L, "Vector", luaopen_vector,1); //Zaregistrování
    objektu do Lua, stejné jako u~modulu. Nyní lze objekt vytvořit a
     s~vytvořeným objektem manipulovat.
```

Zdrojový kód 22: Vytvoření třídy

- `lua_GCCOLLECT`: Garbage collector dokončí svůj cyklus.
- `lua_GCSTEP`: Garbage collector se spustí na tolik kroků, kolik mu uživatel předá ve třetím argumentu.
- `lua_GCCOUNT`: Vrací počet zabraných kilobytů, který právě používá Lua.
- `lua_GCCOUNTB`: Vrací zbytek v bytech po dělení 1024 zabraných kilobytů, který právě používá Lua.

Lua uživateli umožňuje vytvářet tabulky se slabými vazbami (tzv. weak tables). Uživatel by je měl vytvářet obezřetně. Ve chvíli kdy existuje jen slabá vazba na daný prvek, garbage collector ho, jakmile na něj dojde řada, uvolní. Takové tabulky lze vytvořit pomocí metatabulek. Jako metaklíč do metatabulky se zvolí `__mode` a jako string lze nastavit jedna ze 3 možností:

- `"k"`: Slouží jako slabá vazba na klíč.

- "v": Slouží jako slabá vazba na hodnotu.
- "kv": Slouží jako slabá vazba na klíč i hodnotu.

```
1  int gc_func(lua_State * L){
2   printf("V garbage collectoru");
3  }
4  lua_newtable(L);
5  lua_pushstring(L, "x");
6  lua_pushnumber(L, 20);
7  lua_settable(L, -3);
8  lua_setglobal(L, "_WeakTable");
9  lua_getglobal(L,"_WeakTable");
10 lua_newtable(L);
11 lua_pushstring(L, "__gc");
12 lua_pushcfunction(L, gc_func);
13 lua_settable(L,-3);
14 lua_setmetatable(L,-2); //Navázání metatabulky na vrcholu zásobní
      ku na tabulku, která je na indexu -2.
15 lua_pop(L,1);
16 lua_pushnil(L);
17 lua_setglobal(L,"_WeakTable");
18 lua_gc(L, LUA_GCCOLLECT, 0); //Spuštění garbage collectoru vypíše
      se "V garbage collectoru", protože na tabulku _WeakTable již
      není vazba
```

Zdrojový kód 23: Garbage collector

2.6 Práce s Lua kódy v C a ukázka kompletního programu

V této kapitole představím, jak lze integrovat Lua skripty do C. Lua skript je část kódu napsaný v Lua. Pro integraci Lua skriptů do C existuje mnoho možností. Nejelegantnější způsob je použít funkci luaL_dofile, která vyhodnotí kód skriptu. Tato funkce vrací hodnotu 0, jestliže vyvolání skriptu proběhlo korektně, jinak hodnotu 1. Když obsahuje kód skriptu chybu, na zásobník vloží string s chybovou hláškou.

Pro vyhodnocení funkce z Lua je dobré použít kombinaci funkcí lua_pcall a debug.tracebug z Lua. Funkce lua_pcall v případě chyby aplikuje funkci ze zásobníku. Čtvrtý parametr je pro určení indexu funkce, která se aplikuje v případě chyby, tedy odkáže se na pozici, kde bude funkce debug.traceback. Tato funkce v případě chyby vloží na zásobník podrobné informace o chybě.

Kód 25 bude napsán v jazyce Lua.

```c
#include<stdio.h>
#include<stdlib.h>
#include<lua5.2/lua.h>
#include<lua5.2/lauxlib.h>
#include<lua5.2/lualib.h>
int main(){
  lua_State* L = luaL_newstate ();
  luaL_openlibs(L);
  if(luaL_dofile(L,"./skript.lua")){
    printf("Error: %s", lua_tostring(L,-1));
    return 0;
  }
  int iter = 0;
  int exit = 0;
  while(exit == 0 ){
    iter += 1;
    lua_getglobal(L, "debug");
    lua_getfield(L, -1, "traceback");
      lua_remove(L, -2);
    lua_getglobal(L,"main");
    lua_pushnumber(L,iter);
      if(lua_pcall(L, 1, 1, -3)){
        printf("runtime error: %s \n", lua_tostring(L, -1));
          lua_pop(L,1);
      }else{
        exit = (int)luaL_checknumber(L,-1);
        lua_pop(L,1);
      }
  }
  lua_close(L);
  return 0;
}
```

Zdrojový kód 24: Vyhodnocení Lua skriptu

```lua
--Skript je uložen pod jmenem skript.lua ve stejném adresáři, kde se nachází spouštěč aplikace.
print( "Welcome to Lua skript")
function main(iter)
  print( iter)
  if iter > 20 then
    return 1;
  else
    return 0;
  end
end
```

Zdrojový kód 25: Lua – Jednoduchý Lua skript

3 SDL 2.0 a její principy

SDL je multi-platformní vývojová knihovna, která poskytuje přístup k nízkoúrovňovým funkcím audio, klávesnice, myš, joystick a ke grafické kartě pomocí OpenGL a Direct3D. Nativně podporuje jazyk C a C++.

3.1 Inicializace SDL 2.0

SDL 2.0 dále již jen SDL, obsahuje několik vedlejších knihoven. Vedlejší knihovny poskytují lepší abstrakci nad samotným SDL a novou funkcionalitu. Vedlejší knihovny pro SDL jsou konkrétně pro audio, síťové protokoly (nebudu demonstrovat), načítání obrázků, fonty a vektorovou grafikou.

Pro připojení pomocných knihoven k SDL a samotné knihovny SDL slouží tato hlavička.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<SDL2/SDL.h>
4  #include<SDL2/SDL_image.h>
5  #include<SDL2/SDL2_gfxPrimitives.h>
6  #include<SDL2/SDL_mixer.h>
7  #include<SDL2/SDL_ttf.h>
```

Zdrojový kód 26: Inicializace SDL2.0

Jako Makefile, kde budou přilinkované výše zmíněné knihovny, bude sloužit tento.

```
1  CFLAGS=-Wall -std=c99 -shared -g -O0
2  CLIBS=-lSDL2 -lSDL2_image -lSDL2_ttf -lSDL2_mixer -lSDL2_gfx -lm
3
4  OUT_DIR=./
5  OUT=test
6  OBJS=*.o
7  CC=gcc
8  makebin:
9    $(CC) $(CFLAGS) -c *.c
10   $(CC) $(OBJS) -o ${OUT_DIR}$(OUT) $(CLIBS)
```

Zdrojový kód 27: Makefile SDL2.0

3.2 Grafika

Grafika v SDL lze vykreslovat softwarově nebo pomocí grafické karty. Pro výpočty na grafické kartě lze využít SDL abstrakce, ovšem uživateli pak při ladění

programu nestačí kontrolovat jen klasickou paměť připojenou na základní desce a vytížení procesoru. Musí si dát pozor i na stav paměti v grafické kartě a její celkovou vytíženost.

3.2.1 Inicializace

Nejprve se musí inicializovat `SDL_Window`, což je struktura, která reprezentuje okno. Na toto okno se musí napojit `SDL_Renderer`, který bude do okna vykreslovat svůj obsah. Nastane-li při inicializaci chyba, lze ji zjistit pomocí funkce `SDL_GetError()`. Funkce vrací popis chyby jako textový řetězec.

3.2.2 Textura

SDL jako základní grafický element používá strukturu `SDL_Texture`. `SDL_Texture` obsahuje především informace o pixelech. Tato struktura je uložena v paměti na grafické kartě. SDL umožňuje texturu například zprůhledňovat, měnit modulaci barev nebo přistupovat k jejím pixelům. Přístup k pixelům textury je časově náročný, jelikož se obsah textury musí zkopírovat z grafické karty na paměť RAM a po změně pixelů zkopírovat zpět. Texturu lze vytvořit z obrázku, který je uložen v různých formátech. Postup je následující načteme si obrázek do `SDL_Surface` (základní struktura pro softwarové vykreslování), nastavíme barvu, která bude reprezentovat průhlednost, pomocí funkce zkonvertujeme surface na texturu.

3.2.3 Renderer

Pro vykreslení textury na obrazovku SDL využívá `SDL_Renderer`. Renderer je struktura, která je uložena v paměti na grafické kartě. Do rendereru lze kopírovat textury. Při kopírování lze texturám nastavit velikost, ořezání, natočení a umístění na obrazovce. Po zavolání funkce pro prezentování rendereru se vykreslí jeho obsah na obrazovku. Potom je nutné vyčistit renderer. A tvořit obraz znova pro další snímek. SDL umožňuje do rendereru vkládat čtverce, přímky a body. Přímce a bodu není umožněno měnit tloušťku. Těmto objektům lze nastavit barvu, umístění a průhlednost.

SDL nabízí také texturu, kterou lze nastavit jako cíl pro kopírování textur. Lze tedy zkomponovat několik textur do jedné. Tato textura po vytvoření má libovolné pozadí. Nelze jí nastavit průhledné pozadí, což je značné omezení. Po kompozici se musí zavolat funkce, která nastaví kopírování zpět do hlavního rendereru.

3.3 Audio

Audio v SDL poskytuje nízkoúrovňové funkce. Ve většině případů je proto lepší použít pomocnou audio knihovnu pro SDL. Dále budu uvažovat jen pomocnou knihovnu `SDL_Mixer`.

SDL_Mixer nabízí dva typy přehrávání zvuků. Prvním typem je zvuk Mix_Music, který je určen pro delší přehrávání (většinou hudba na pozadí). Může být spuštěn jen jeden tento zvuk. Dále je pak typ Mix_Chunk pro přehrávání zvuků, které lze míchat do sebe. Většinou se používá jako zvuk k události.

Audio se musí nejprve pomocí funkce inicializovat. Dále pak knihovna poskytuje funkce pro načítání několika formátů hudby pro oba zvukové typy. Pomocí funkcí s těmito typy hudby lze měnit vzdálenost zvuku, polohu, ovládání hlasitosti, pozastavení, vypnutí, opakování a další zvukové efekty.

3.4 Události, klávesnice a myš

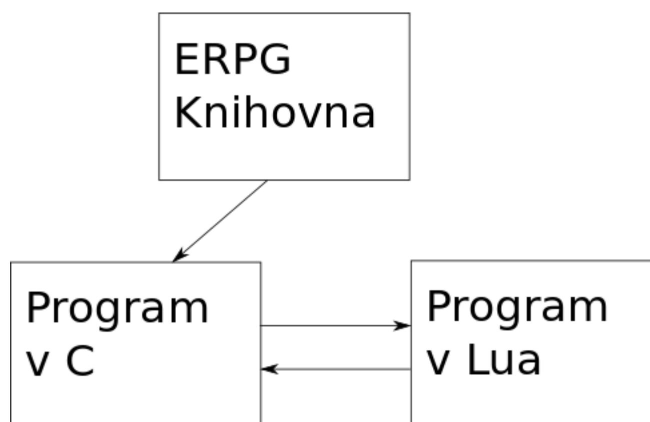
Události v SDL fungují na principu fronty. Při běhu knihovny SDL se vkládají do fronty události ze vstupních zařízení. Potom funkce pro odebírání prvků z fronty vloží popis události na ukazatel na strukturu SDL_Event. Dále pak pomocí funkce a nastavení jejich flagů lze zjistit, o jakou událost se jedná.

Uživatel tímto způsobem v SDL zjistí z klávesnice, která klávesa byla stisknutá, jak dlouho je stisknutá a nebo uvolněná. Také SDL obsahuje událost pro psaní textu. Najde využití při tvoření textového pole.

U myši lze zjistit stisky tlačítek, uvolnění, skrolování pomocí kolečka, pohyb myši, její pozici a také jestli se jedná o dvojklik.

4 Architektura aplikace

Vzhledem k velikosti programu bylo podstatné elegantně navrhnout architekturu aplikace. V C byla vytvořena knihovna s názvem ERPG, která vytvaří dostatečnou abstrakci nad SDL. Tato abstrakce byla exportována do Lua pomocí modulů a tříd. Dále pak byl vytvořen program v C, který otevírá knihovnu ERPG a exportuje z něj moduly do Lua. Program inicializuje knihovnu ERPG, vyvolá vyhodnocení souboru `start_game.lua` a vytváří hlavní smyčku pro vykreslování. V hlavní smyčce volá funkci z Lua s názvem `main`. Dále pak při chybě pozastaví smyčku a spustí konzoli. Nebo lze pomocí klávesy `F2` vyhodnotit soubor `buffer.lua`. Klávesa `F1` slouží pro vyvolání konzole v terminálu. Tyto možnosti jsou vhodné pro inspekci kódu. Opuštění hlavní smyčky probíhá pomocí exportované funkce do Lua. Po ukončení hlavní smyčky se uvolní ERPG knihovna, Lua a SDL.



4.1 Tvorba knihovny ERPG v jazyku C a propojení s Lua

Pro strukturovanější kód bylo zapotřebí vytvořit knihovnu. Knihovnu jsem nazval ERPG. Tato knihovna je vytvořena pro integraci Lua do C. Obsahuje funkce pro jazyk C, které slouží k inicializaci knihovny, otevření knihovny pro Lua, vykreslování a ukončení knihovny. Především jsou, ale funkce tvořeny speciálně pro jazyk Lua. Knihovna je zkompilována jako statická pod standardem C99.

Knihovna má jednu globální strukturu pro udržování svého stavu. Obsahuje odkaz na strukturu `ERPG_Window`, `ERPG_Audio` a proměnnou `exit`. Proměnná `exit` lze měnit z Lua pomocí funkce. Měla by se využívat pro příznak ukončení hlavní smyčky.

4.1.1 Správa grafiky a vstupních událostí

Při vytvoření okna pomocí C funkce `ERPG_make_window` se vytvoří instance `ERPG_Window`. Struktura této instance obsahuje potřebné informace k vykreslování. Dále pak odkaz na strukturu pro myš, klávesnici a dva asociativní seznamy.

První asociativní seznam je pro načtené obrázky ze souborů a druhý pro načtené fonty. Ze struktury lze zjistit velikost okna a nachází-li se okno v režimu celé obrazovky. Do Lua je exportován jako modul `ERPG_Window`.

Pro vyčištění rendereru a obnovení událostí myši a klávesnice je vytvořena C funkce `Lua_prepare` (existuje ekvivalent pro Lua). Tato funkce obnoví události myši, klávesnice a vyčistí renderer. Do globálního prostředí v Lua vytvoří tabulku `mouse` a `keyboard`. V těchto tabulkách se nachází informace o událostech, které nastaly od předchozího zavolání funkce `Lua_prepare`.

Pro vykreslení rendereru je vytvořena C funkce `Lua_update` (existuje ekvivalent pro Lua). Tato funkce vykreslí vše co bylo zkopírováno do rendereru. Co bylo dříve zkopírováno je dříve vykresleno.

Pro grafické objekty je několik základních struktur:

- `ERPG_Sprite`: Struktura pro udržování načtených obrázků.
- `ERPG_Text_Element`: Struktura pro udržování úseků textu s různým grafickým nastavením.
- `ERPG_Rectangle`: Struktura pro tvoření obdélníků.
- `ERPG_Line`: Struktura pro tvoření přímek.

Tyto struktury jsou do Lua exportovány jako třídy. Některé jejich metody jsou stejně pojmenované pro jednoduchou manipulaci. Třídy na tvoření těchto grafických objektů mají společné metody pro kopírování do rendereru, změnu měřítka, posouvání po obrazovce, výřez textury a pro nastavení průhlednosti.

4.1.2 Zpráva zvuku

Při otevření audia pomocí inicializační funkce `ERPG_Audio_create` se vytvoří instance `ERPG_Audio`. Inicializační funkce obsahuje parametry k počtu kanálů pro mixování zvuků. Struktura vytvořené instance obsahuje vše potřebné pro přehrání zvuku. Ve struktuře jsou dva asociativní seznamy s načtenou hudbou. Dva seznamy, protože knihovna `SDL_Mixer` obsahuje dvě struktury hudby pro přehrávání, tedy pro každou strukturu jeden seznam. Dále pak obsahuje seznam hudby, který lze přehrát a proměnnou pro hudbu na pozadí. Pro přehrání seznamu a hudby na pozadí je nutné zavolat funkci. Do Lua je exportován jako modul `ERPG_Audio`.

Pro načtení a manipulaci s hudbou na pozadí existují funkce přímo v modulu `ERPG_Audio`.

Základní struktura pro zvuk je `ERPG_Sound`. Tato struktura je exportována do Lua jako třída. Má několik metod pro manipulaci.

4.1.3 Klávesnice a myš

Do globálního prostředí v Lua jsou vytvořeny dvě tabulky. První je `mouse` a druhá `keyboard`. Tabulky slouží pro zjištění událostí k daným vstupům, které nastaly v předchozí iteraci.

Tabulka `mouse` má následující klíče:

- `x`: x-ová souřadnice myši
- `y`: y-ová souřadnice myši
- `press`: při stisknutém tlačítku se za klíč dosadí název tlačítka, jinak se dosadí `none`
- `press_motion`: při stisknutém tlačítku a pohybu se za klíč dosadí název tlačítka, jinak se dosadí `none`
- `release`: při uvolnění tlačítka se za klíč dosadí název tlačítka, jinak se dosadí `none`
- `on_press`: při stisku tlačítka se za klíč dosadí název tlačítka, jinak se dosadí `none`
- `wheel_y`: při scrollu se scrollovacím tlačítkem se za klíč dosadí `1` nebo `-1` jinak se dosadí `0`

Tabulka `keyboard` má následující klíče:

- `press`: při zmáčknutém tlačítku se přidá do tabulky název tlačítka
- `release`: při uvolnění tlačítka se přidá do tabulky název tlačítka
- `input_key`: při zapnutém módu pro `input text` se do tabulky přidávají znaky stisknuté na klávesnici

4.1.4 Pomocné funkce

Knihovna obsahuje ještě modul `ERPG_Utils`. Tento modul obsahuje pomocné funkce pro Lua. Například funkci pro zjišťování průniku obdélníku s obdélníkem, nebo funkci pro procházení adresářové struktury.

4.2 Použití knihovny ERPG v C a propojení s Lua při tvorbě hry

Pro použití knihovny ERPG se musí vytvořit nový C projekt. Ve vytvořeném projektu je nutné inicializovat knihovnu ERPG a Lua stav. Poté se tvoří okno a inicializuje zvuk. Do Lua stavu se otevřou moduly z ERPG knihovny a základní Lua moduly. Dále se vyhodnotí hlavní skript v Lua, který by měl obsahovat funkci `main`. Pak následuje hlavní cyklus, který lze ukončit z Lua.

Na začátku hlavního cyklu se používá funkce `Lua_prepare`. Pak následuje vyhodnocení funkce `main` z Lua. Tato funkce se vyhodnocuje v každé iteraci. Při chybě se vypíše informace o chybě a otevře se konzole. Dále pak je vhodný

kód pro otevření konzole a vyhodnocení Lua skriptu. Pro vykreslení vybudovaného snímku slouží funkce `Lua_update`. Před novou iterací by měl být kód pro garbage collector a nastavení čekání na další iteraci.

Pro běh garbage collectoru se použije funkce `lua_gc` s nastavením `LUA_GCSTEP` pro realtime hru ideální. Je nutné zvolit takový počet kroků, aby čas iterace nebyl delší než maximální doba snímku.

Po ukončení hlavní smyčky následuje kód pro ukončení Lua a korektního ukončení knihovny ERPG.

```
1  //Inicializace knihoven především ERPG a Lua
2  int main(){
3   lua_State* L = luaL_newstate ();
4   luaopen_main(L);
5   luaL_openlibs(L);
6   ERPG_CREATE_CORE();
7   ERPG_Audio_create(44100, 4096, 16);
8   ERPG_make_window("ENGINE RPG");
9   ERPG_CORE * core = ERPG_get_CORE();
10  //Vyhodnocení hlavního Lua skriptu.
11  while(!core->exit){
12    Lua_prepare(L);
13    //Volání hlavní funkce z~Lua. Případně zkratky na spuštění
        konzole nebo vyhodnocení skriptu
14    Lua_update(L);
15    //Nastavení garbage collectoru + nastavení času pro další iteraci
        .
16  }
17  lua_close(L);
18  ERPG_Destroy_core();
19 }
```

Zdrojový kód 28: Použití knihovny ERPG

4.3 Tvorba GUI v Lua

GUI znamená graphic user interface při přeložení do češtiny grafické uživatelské rozhraní. GUI umožňuje ovládání aplikace pomocí grafických ovládacích elementů.

V Lua jsem vytvořil modul GUI, který umožňuje tvorbu základních elementů a jednoduché přidávání pokročilejších grafických elementů. Pomocí elementů lze vykreslovat určité grafické prvky, dále pak mohou přijímat a odesílat události. Událostí je několik druhů zejména pro zpracování událostí myši, iterace a zničení objektu.

Při tvorbě GUI se jako první vytvoří hlavní objekt. Tento objekt zpravuje události. Objekt má v sobě tabulku `objects`. Tabulka `objects` se prochází stromově a vykresluje všechny obsažené prvky. Pomocí této tabulky se také aplikují funkce na navázané události.

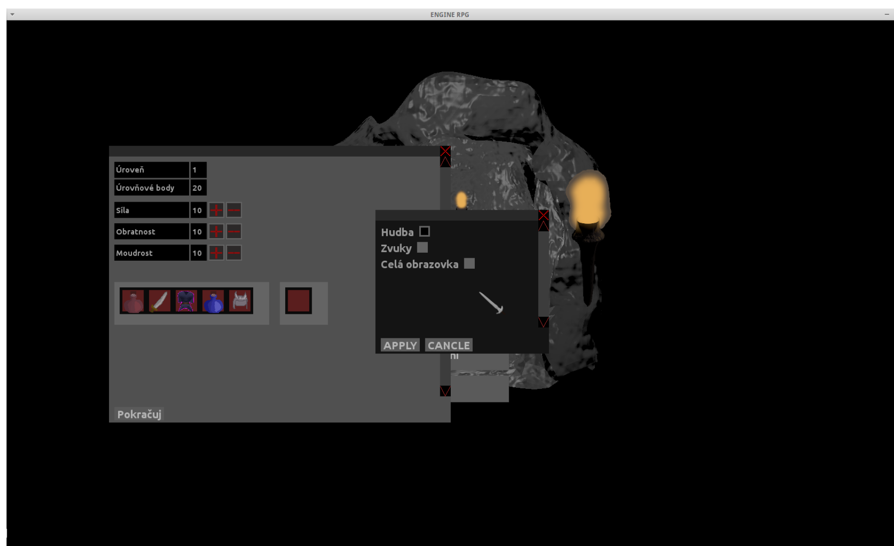
4.3.1 Základní grafické elementy

Tyto grafické objekty jsem zvolil jako základní grafické elementy:

- `frame`: Pro vykreslení obdélníku.
- `text`: Pro vykreslení textu.
- `empty_frame`: Pro vytvoření obdélníku bez grafiky.
- `sprite`: Pro vykreslení obrázku ze souboru.

Zmíněné elementy si udržují obdélník, který určuje jejich pozici a rozměry. Objekty mají implementované metody pro pohyb a pro změnu rozměru.

Děděním a kompozicí těchto prvků se vytváří pokročilejší grafické prvky. V GUI modulu lze najít „checkboxy“, „messageboxy“, „inputboxy“ a také lze tvořit okna s obsahem grafických prvků, které umožňují scroll. GUI modul obsahuje mnoho grafických prvků.



Obrázek 1: Ukázka GUI.

4.3.2 Události

Událost v GUI modulu je reprezentována textovým řetězcem. Na tento řetězec je navázána funkce s argumenty, kde první argument je sám objekt, na který je událost navázána. Objekt si drží události ve své struktuře. Tento modul má několik základních událostí:

- `on_wheel`: Při použití scrollovacího kolečka.
- `on_press_key`: Při zmáčknutí klávesy.

- `on_input_key`: Při psaní textu.
- `on_release_key`: Při uvolnění klávesy.
- `on_motion` Při držení tlačítka myši a zároveň při pohybu myši.
- `on_click`: Při změnách stavu nebo zmáčknutém tlačítku myši.
- `on_iter`: Při každé iteraci.
- `on_destroy`: Při poslání objektu této události.

K navázání události se musí vytvořit funkce, která se naváže na konkrétní událost a grafický element. Pro aktivaci tohoto grafického elementu a jeho vykreslení se musí napojit na grafický element, který je již aktivován. Po napojení si ukládá napojený objekt svého rodiče.

4.4 Tvorba herního engine v Lua

Herní engine se stará o jádro hry a poskytuje rozhraní pro ulehčení tvorby hry. Vytvořil jsem takový engine, který je dostatečně rychlý a zvládá vytvářet poměrně rozsáhlé herní mapy.

Pomocí GUI lze na prvek navázat herní plátno. Herní plátno pak komunikuje s GUI pomocí prvku, na který bylo navázáno. Herní plátno má strukturu, která si udržuje čas hry, obsahuje svůj vlastní systém událostí a vlastní strukturu pro myš a klávesnici. Vzhledem k tomu, že jsem programoval izometrické RPG, tak jsem vytvořil například funkce pro vkládání prvků na mapu, posouvání vykreslovacího plátna po mapě a animace.

Mapa se skládá ze dvou dvourozměrných polí. První pole je určeno pro dlaždice a druhé pro statické a dynamické objekty. V druhém poli lze vytvořit tři vrstvy. Tyto vrstvy slouží k pořadí objektů. Dále pak mapa udržuje rámec, který určuje pozici na mapě. Pomocí rámce lze vykreslit jen určitou část z mapy.

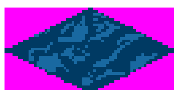
4.4.1 Animace

Animace jsem abstrahoval tak, aby šly vytvářet pomocí Lua konfiguračního souboru. Lua konfigurační soubor je soubor, který je psán v Lua. V tomto souboru jsou uloženy tabulky, které obsahují klíče s navázanými parametry dle vzoru. Tabulka pro animaci se skládá z inicializační a konfigurační části. V inicializační části se musí určit odkud se načítají animace a zvuk. Druhá část konfiguruje animaci. V této části se udává název animace, pozice animace a zvuk, který má vydat při konkrétní animaci.

4.4.2 Dlaždice (Tiles)

Dlaždice jsou grafické prvky které mají stejné rozměry, ale mohou mít rozdílné textury. Tyto prvky se poskládají naspod mapy, aby vytvořily pozadí mapy většinou podlaha. V mé hře má dlaždice rozměry $64x32$. V RPG většinou dlaždice

reprezentuje jeden metr. Dlaždice neumožňuje uchování stavu. Dlaždice je reprezentovaná 2 bajtovým číslem, které reprezentuje index do tabulky dlaždic. Tento způsob umožňuje jednoduché uložení a malou paměťovou náročnost. U dlaždic jsou důležité optimalizace, protože na mapě například 1000x1000 je dlaždic 1000000. Další optimalizace u dlaždic je vykreslování. Dlaždice jsou nejlevnější objekt na mapě. Dlaždice se definují v konfiguračním souboru.



Obrázek 2: Ukázka dlaždice růžová barva je průhledná

4.4.3 Statické objekty (Static objects)

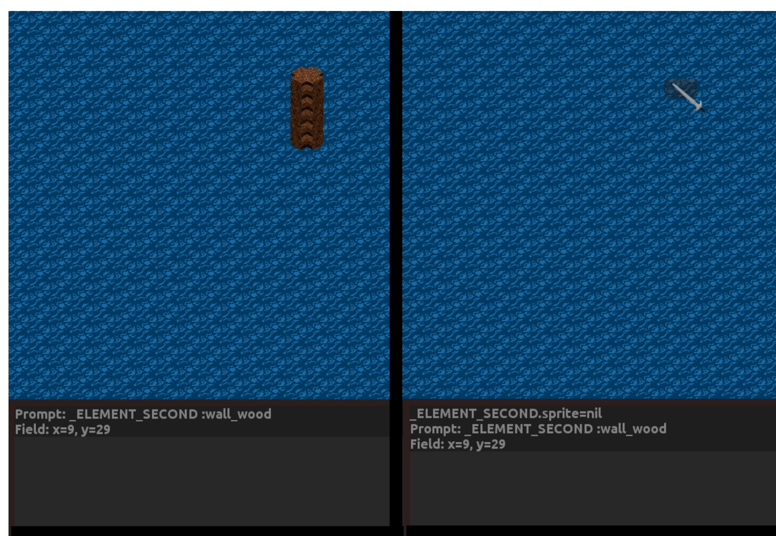
Statické objekty jsou objekty, které jsou určeny pro nepohybující se objekty. Na tyto objekty lze navázat události. Ve hře jsou často používané jako zdi, stromy, truhly atd. Tyto objekty jsou zoptimalizované tak, aby jich mapa mohla pojmout co nejvíce. Uložení statických objektů do souboru je ve formě Lua kódu. Při uložení si pamatují svůj stav, jméno a identifikační číslo. V poli, které si drží struktura pro mapu a přímo ve vrstvě, ve které se nachází je uložen jako odkaz. Statické objekty se definují v konfiguračním souboru.

4.4.4 Dynamické objekty (Dynamic objects)

Dynamické objekty umožňují například přehrávat zvuk a animace. Tyto objekty jsou především pro postavy, monstra, kouzla nebo lze pomocí nich vytvořit například pasti. Dynamické objekty se ukládají společně se statickými objekty. Způsob ukládaní je stejný, pouze dynamické objekty si pamatují navíc informace o pohybu, umístění, v jaké pozici je animace, která událost se vykonává a další důležité informace pro jejich načtení. Dynamický objekt je v poli mapy uložen stejně jako statický objekt. Dynamické objekty jsou nejnáročnějším prvkem na mapě, kvůli množství udržovaných informací a událostem, které jsou na ně automaticky navázané. Proto je není vhodné používat například jako zdi a stromy.

4.4.5 Konzole

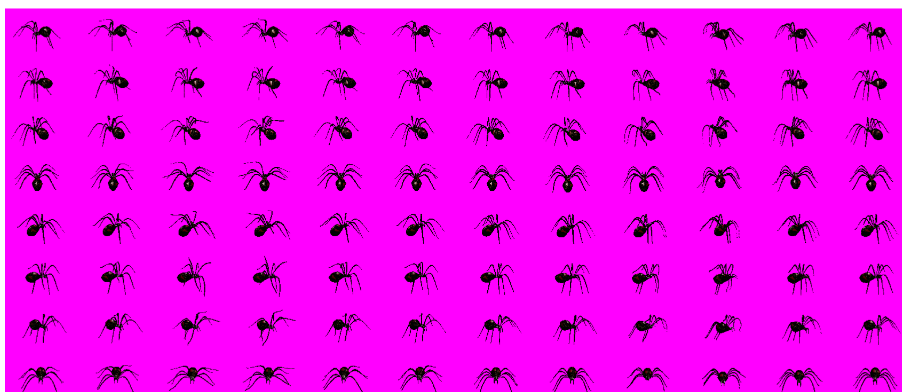
Herní engine nabízí grafickou konzoli. Tato konzole je navržená pro debugování a pro ulehčení práce v editoru. Konzole umožňuje doplňování a vyhledávání z předchozích příkazů. V editoru lze konzoli předat, po zmáčknutí `CTRL` a levého tlačítka myši, informace o konkrétním poli. Toto pole se uloží do proměnné. S proměnnou lze z konzole manipulovat. Konzole je vytvořená pomocí GUI prvků.



Obrázek 3: Ukázka použití konzole

4.5 Vystavění hry na vybudovaném kódu

Při budování hry jsem nejprve musel vytvořit základní testovací grafiku. To znamenalo vytvořit základní animace postav, objekty a dlaždice. Pro tvorbu animací postav a objektů jsem zvolil 3D editor Blender. Tento editor mi umožnil vytvářet 3D objekty. Tyto objekty jsem pomocí skriptu v jazyce Python nafotil z pozic pro izometrickou grafiku. Pozice jsem jednoduše spočítal ze znalosti úhlu natočení kamery a vzdálenosti. Fotky ze skriptu se mi uložili do speciální adresářové struktury a se speciálním názvem. Dále jsem pomocí grafického editoru Gimp seskupil fotky do jednoho souboru. K tomuto úkonu jsem opět využil skriptu, který jsem tentokrát vytvořil v jazyce Scheme. Vytvořený skript mi správně umístil a naškáloval fotky. Jako průhlednou barvu jsem použil růžovou konkrétně zapsanou v RGB `255,0,255`.



Obrázek 4: Animace pavouka

Dalším krokem bylo vytvoření editoru pro mapy. Tento editor byl vcelku jed-

noduchý vytvořit protože GUI a herní engine mi poskytly plno užitečných funkcí. V tomto editoru jsem vytvořil několik druhů štětců a různé panely pro vybírání dlaždic, statických objektů a dynamických objektů. Všechny objekty se registrovaly do editoru pomocí konfiguračních souboru. Tyto soubory byly zároveň sdílené i pro hru. V editoru nelze uložit mapu pokud se spouští z nainstalované aplikace. V nainstalované aplikaci slouží jen pro ukázku postupu vytvoření map.

Při tvorbě hry jsem si musel vytvořit konfigurační soubory, které popisovaly jednotlivé postavy. Z nich se vytvořily postavy jako objekty, které dědily vlastnosti z dynamických objektů. Dále pak postavám se přidala jednoduchá umělá inteligence. Inteligenci jsem vytvořil tak, aby byla výpočetně nenáročná. Nepřátelské postavy si ve většině případů najdou cestu k hrdinovi, kterého ovládá hráč. Potom jsem vytvořil soubojový systém mezi postavami, padání předmětů při smrti, tvoření kouzel, vybavování hrdinu různými předměty a s tím spjatý inventář. Hrdinu jsem musel navázat na ovládání pomocí klávesnice a myši. Dále pak jsem propojil hrdinu s GUI, aby se projevovaly změny na životech nebo například kvůli posílání zpráv do herní konzole.

Nakonec jsem vytvořil systém pro vytváření úkolů (questů). Systém úkolů funguje tak, že se vytvoří textový soubor, do kterého se zapisují otázky, odpovědi a jejich identifikátor. Otázka je věta od postavy, která vede s hrdinou rozhovor, a odpověď je věta od hrdiny tedy od uživatele hry. Postava si pak drží tabulku, která obsahuje informace o tom, který text lze vypsat, jaké odpovědi se mohou vypsat a události po kliknutí na různé odpovědi. Při kliknutí na nějakou odpověď lze například přidat hrdinovi nějaké předměty, úkoly nebo zkušenosti.

Obrázek 5: Aktivace úkolu

Protože aplikace je zatím určená pro běh na Ubuntu 14.04 vytvořil jsem deb balíček. Deb balíček umožňuje standardní instalaci v Ubuntu. Při instalaci se zjistí, jestli jsou nainstalované konkrétní knihovny. Pokud nejsou a má-li příslušný systém odkaz na zvolené knihovny nainstalují se. Po instalaci si spouštěč hry nastaví ikonu hry a přidá se do sekce „Hry“ v menu.

5 Uživatelská příručka

Z pohledu uživatele mé aplikace bude sloužit tato kapitola jako návod ke hře a editoru. Budu se věnovat principům ovládání hry a jejím mechanismů. Dále pak ukážu tvorbu map v editoru. Některá funkčnost je v aplikaci navíc, z důvodu demonstrace vývoje aplikace například konzole v terminálu nebo editor. Konzole se spouští v terminálu při zmáčknutí klávesy `F1`, při spouštění konzole dojde k pozastavení běhu hry dokud se v terminálu nenapíše nějaký příkaz. Silně doporučuji spouštět konzoli jen pokud je hra spuštěná z terminálu, jinak dojde k nenávratnému „zamrznutí“ hry.

5.1 Návod ke hře

Po úspěšném nainstalovaní deb balíčku a následném spuštění hry by se mělo objevit okno. V okně se nachází menu, které má několik tlačítek se zřejmou funkčností. V nastavení lze zvolit „hudba“, „zvuky“ a „celá obrazovka“ vedle těchto možností se nachází čtverec buď prázdný nebo vyplněný. Pro aplikování „checkboxů“ se musí kliknout na čtverec. Rozdíl mezi hudbou a zvuky je, že hudba je pro pozadí a zvuky jsou pro zvuky ve hře.

Pro spuštění nové hry se klikne na tlačítko „Nová hra“, které vyvolá zobrazení okna 6. Před začátkem hry je vhodné rozdat „Úrovňové body“. Pro snadný průběh hry se navolí body dle obrázku. Dále si lze všimnout několika předmětů, kde jeden z těchto předmětů se musí přetáhnout do prázdného políčka. Přetáhnout znamená kliknout na konkrétní předmět a držet levé tlačítko myši a přesunout pomocí myši předmět na pozici prázdného políčka, kde levé tlačítko uvolníme. Na obrázku jsem přesunul předmět „tunika“.

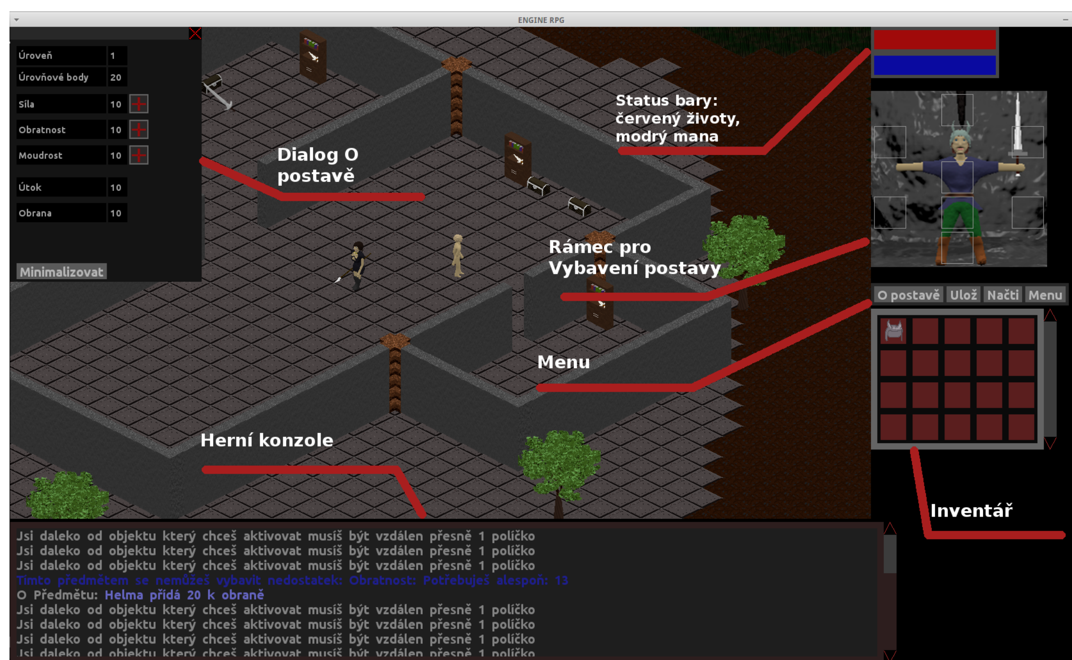
Obrázek 6: Před spuštěním nové hry

Po stisku tlačítka a krátkém „loadingu“ se přepneme do hry 7. Hra se skládá z několika GUI prvků. Vpravo nahoře nalezneme dva „status bary“, první červený slouží pro určení hrdinových životů a druhý modrý určuje hrdinovu „manu“.

Pod „status bary“ nalezneme obrázek postavy s čtverci, které slouží pro vybavení hrdiny. Při přesunutí předmětu, kterým lze vybavit hrdina, do prázdného čtverce v dialogu vybavení hrdiny, se hrdina tímto předmětem vybaví.

Pod zmíněným grafickým prvkem se nachází menu, kde se nachází čtyři tlačítka. První tlačítko „O postavě“ slouží k zobrazení dialogu s hrdinovými dovednostmi, kde lze rozdávat úrovňové body jednotlivým schopnostem. Tlačítko „Ulož“ a „Načti“ se chová dle předpokladu. Tlačítko „Menu“ pausne hru a přepne se do úvodní obrazovky. Pod grafickým prvkem menu se nachází inventář. Inventář slouží pro ukládání nalezených předmětů.

Jako poslední prvek, který je ve spodní části pod herním plátnem je takzvaná herní konzole. Do této konzole se vypisují informace o dění ve hře, například při vybavování předmětem se vypíše, co předmět přidá, nebo proč se jím nelze vybavit.



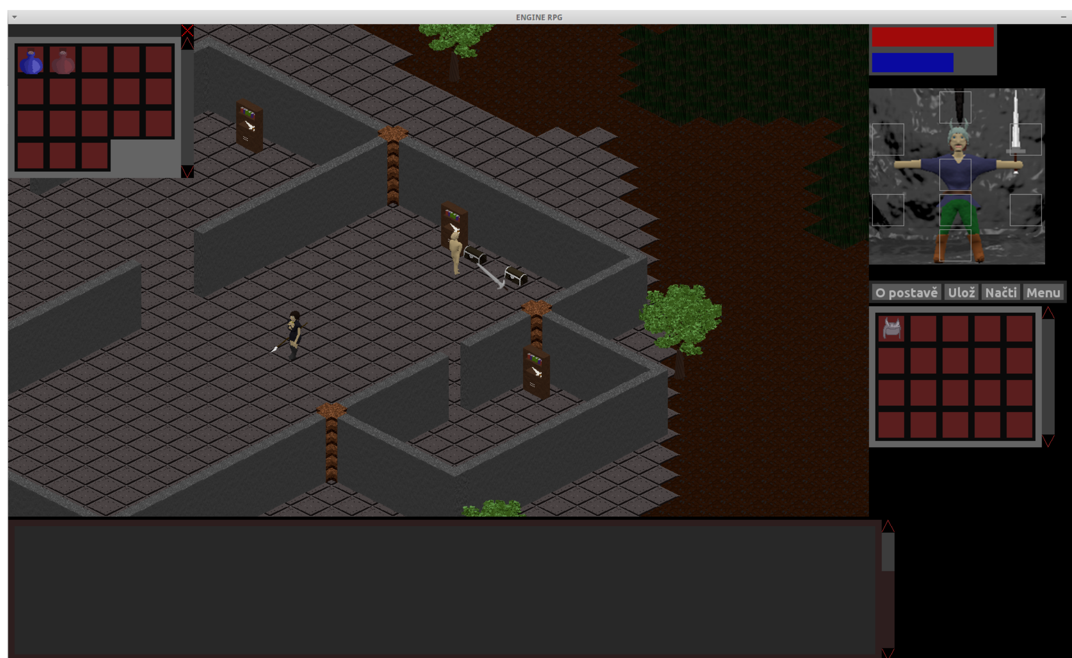
Obrázek 7: Herní GUI

Dále popíši ovládání a interakci s předměty. Pro pohyb po mapě se využívá pravého tlačítka myši a pozice myši na herním platně. Herní plátno je rozděleno na osm částí přičemž každá část určuje směr pohybu. Dle pozice myši se volí kurzor, který ukazuje směr pohybu. Ovládání hry funguje na principu přidržení tlačítka, které reprezentuje danou akci a po kliknutím levého tlačítka myši se akce provede. Při souboji a vyvolání kouzla se určuje směr opět dle pozice myši na herním plátně. Po zmáčknutí tlačítka pro souboj nebo kouzlo se změní kurzor dle směru. Po kliknutí levým tlačítkem se aplikuje souboj či kouzlo v daném směru.

Typy akcí:

- Souboj na blízko klávesa „Left Shift“.
- Vyvolání kouzla klávesa „Left Ctrl“.
- Použití lahvičky s inventáře na životy „S“.
- Použití lahvičky s inventáře na manu „D“.

Při najetí kurzoru na předmět či postavu, se kterou lze interagovat, se ztmaví. Ve vzdálenosti jednoho políčka od tohoto objektu lze po kliknutí myši objekt aktivovat. Po aktivaci se stane nějaká akce například po kliknutí na truhlu se otevře inventář s jejím obsahem nebo po kliknutí na postavu se otevře dialog. Po otevření truhly lze přesunout předměty z truhly do inventáře hrdinu. Mezi jednotlivými inventáři lze přesouvat libovolné předměty. Postavy vám mohou zadávat úkoly.



Obrázek 8: Interakce s truhlou

Při procházení kolem stěn se stěny zprůhledňují, aby bylo vidět co se nachází vedle hrdinu. Ve hře jsou dva úkoly, které vám zadá postavička ženy na obrázku 8. Pro dokončení hry se musí splnit všechny úkoly. Po splnění těchto úkolů vyběhne dialog s nápisem „Zvítězil jsi“. Po tomto dialogu lze volně pokračovat ve hře.

5.2 Návod k editoru

Editor je určen pro vývojáře aplikace. Proto neobsahuje uživatelsky přívětivé rozhraní. Mapa v editoru nelze uložit, pokud je spuštěná aplikace, která je nainstalovaná pomocí deb balíčku. Editor je v tomto případě jen pro demonstraci

vývoje map nikoli pro vytváření nových map. Z tohoto důvodu může v editoru docházet k některým chybám.

Editor se spustí po kliknutí na tlačítko „Spustit editor“. Po spuštění editoru lze vidět několik palet s různými nastaveními. Dále pak vlevo nahoře jsou tlačítka „Save“ a „Load“. Pod herním plátnem se nachází konzole. Do konzole lze zadávat příkazy, které se vyhodnocují v Lua. Při přidržení klávesy „Left Ctrl“ a kliknutím levého tlačítka na mapu se uloží do proměnné obsah políčka a do konzole se vypíše název této proměnné a umístění políčka v poli uvnitř Lua. S proměnnými lze pak libovolně pracovat pomocí příkazů stejně jako v Lua.

Palety v editoru lze libovolně přesouvat. Dále pak je v editoru minimapa. Minimapa slouží jako náhled celé mapy s tím, že lze po kliknutím pravého tlačítka na mapu rychlý přesun na danou pozici.

Obrázek 9: Editor GUI

Pro zvolení objektu, který chceme vkládat na mapu, se musí zakliknout náhled tohoto objektu v paletě. Pro zvolení velikosti štětce se použije paleta pro štětce. Růžový obdélník slouží pro mazání statických objektů, postav i monster. Dlaždice logicky mazat nelze, jde je jen nahrazovat jinými dlaždicemi. Po zvolení příslušných prvků s palety se pod kurzorem na herním plátně ukazuje náhled objektů, aby uživatel věděl kam se umístí konkrétní prvky. Konzole podporuje doplňovaní z historie pomocí klávesy „Tab“. Konzole slouží především pro přidávání předmětů do truhel.

Závěr

Výstupem této bakalářské práce byla jednoduchá RPG hra. Knihovna v C pro Lua byla vytvořená obecně, poskytující silné rozhraní pro tvorbu 2D her. Tvorba grafiky ve 2D je z určité části omezena kvůli SDL, proto by bylo dobré v budoucnu vykreslování grafiky předělat v knihovně do čistého OpenGL za použití shaderů. Dále pak byl vytvořen jednoduchý modul pro GUI systém a herní engine.

Conclusions

Output of this bachelor thesis was simple RPG game. Library in C for Lua was created generally, providing strong API for creating 2D games. Graphics creating is somewhat limited due to SDL nature and could have been later reworked into pure OpenGL with usage of shaders. Also I have created simple GUI modul and game engine.

A Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

bin/
: Instalátor programu RPG_I386.DEB.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní data pro kompilaci programu RPG. V adresáři src/src se nachází adresáře, ve kterých jsou zdrojové kódy programu.

readme.txt
: Instrukce pro instalaci a spuštění programu RPG, včetně všech požadavků pro jeho bezproblémový provoz.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] ROBERTO, Ierusalimschy (comp.). *Programing in Lua - 2nd edition.* Rio de Janeiro: Biblioteca do Departamento de Inform atica, PUC-Rio, 2006. 329 s. ISBN 85-903798-2-5.

[2] BORGMAN Christine L., Dennis M. Ritchie. *The C programing language.* 1978.

[3] *SDL 2.0 Documentation. : dokumentace k SDL 2.0* [online]. 2015 [cit. 2011-02-02]. Dostupný z: ⟨https://wiki.libsdl.org⟩.

[4] *SDL 2.0 Official website. : oficialní web k SDL 2.0* [online]. [cit. 2015-05-17]. Dostupný z: ⟨https://www.libsdl.org/⟩.

[5] *Lua 5.2 Reference Manual. : dokumentace k Lua 5.2* [online]. [cit. 2015-04-03]. Dostupný z: ⟨http://www.lua.org/manual/5.2/⟩.

[6] *Lua 5.2 Reference Manual. : použití Lua* [online]. [cit. 2015-03-24]. Dostupný z: ⟨http://www.lua.org/uses.html⟩.

[7] BELLANGER, Clint (ed.). *Isoemtric Tiles: popis izometrické grafiky.* Dostupný z: ⟨http://flarerpg.org/tutorials/isometric_intro/⟩.

[8] JĘDRZEJEK, Tomáš (ed.). *Tvorba deb balíčku* [online]. [cit. 2012-08-31]. Dostupný z: ⟨http://wiki.ubuntu.cz/vytvo%C5%99en%C3%AD_.deb_bal%C3%ADku⟩.