VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

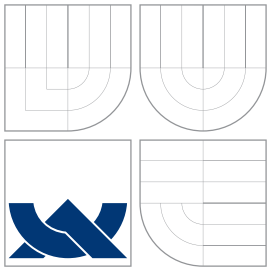
MOTIVAČNÍ VÝUKOVÁ HRA NA PLATFORMĚ ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

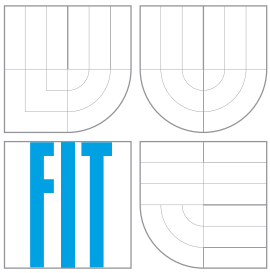
AUTOR PRÁCE
AUTHOR

Bc. MARTIN KUČERA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOTIVAČNÍ VÝUKOVÁ HRA NA PLATFORMĚ ANDROID

EDUCATIONAL GAME OF MOTIVATION ON THE ANDROID PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KUČERA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. JITKA KRESLÍKOVÁ, CSc.

BRNO 2015

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2014/2015

Zadání diplomové práce

Řešitel: **Kučera Martin, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Motivační výuková hra na platformě Android**
Educational Game of Motivation on the Android Platform
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte literaturu zabývající se logickými členy a logickými obvody, zaměřte se na simulaci logických obvodů.
2. Seznamte se s metodami hraní her s využitím konceptů rozvíjení lidské vynalézavosti.
3. Seznamte se s problematikou multiplatformních aplikací.
4. Zhodnoťte existující aplikace pro podporu výuky nebo jako motivační faktor rozhodování o budoucím oboru studia, které se zabývají úvodem do logických obvodů.
5. Specifikujte požadavky na počítačovou hru pro podporu intuitivního uvažování a navrhněte její architekturu.
6. Zvolte vhodné vývojové prostředí a po dohodě s vedoucí implementujte část navržené aplikace na platformě Android.
7. Použitelnost aplikace demonstrujte na vhodném vzorku dat vybraném po dohodě s vedoucí.
8. Zhodnoťte výslednou aplikaci a navrhněte možná rozšíření.

Literatura:

- Vingron, S.:Logic Circuit Design. Springer Heidelberg Dordrecht London New York, 2012.
- Choi, K., Kang, D.: Modeling and simulation of discrete-event systems. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013.
- Ross, S.: Simulation. Academic Press, 2002, ISBN 0-12-598053-1.
- Hashimi, S., Y., Komatineni, S., MacLean, D.: Pro Android 2, Apress, 2010, ISBN 13-978-1-4302-2659-8.
- Další dle pokynů vedoucí.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 5 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kreslíková Jitka, doc. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2014
Datum odevzdání: 27. května 2015

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je vytvoření prototypu výukové hry, která by měla usnadnit rozhodování při výběru zaměření budoucího studia nebo záliby. Záměrem hry je poskytnout názorný a především velmi pozvolný úvod do problematiky logických systémů.

Teoretická část práce se zabývá tématy potřebnými pro návrh hry. Vysvětluje logické systémy, hradla, klopné obvody, věnuje se žánrům počítačových her a zaměřuje se zejména na puzzle hry včetně jejich historie.

Dále diplomová práce představuje problematiku multiplatformního vývoje pro mobilní zařízení i počítače, stručně popisuje několik známých multiplatformních knihoven. K závěru předvádí a srovnává několik vybraných aplikací, které slouží pro podporu výuky logických systémů.

Abstract

The aim of the dissertation is to create a prototype of an educational game which should help students with the choice of their future studies. To this end, the game introduces the topic of logic systems in a non-threating interactive way with clear illustrative examples.

The theoretical part of the work deals with the background knowledge necessary in order to design the game. It explains logic systems, gates, latches, and explores various aspects of game genres with primary focus on puzzle games and their history.

In addition, the dissertation presents some of the issues related to cross-platform development of games for both mobile and computer platforms, and provides brief descriptions of several popular cross-platform libraries. Lastly, the work introduces and compares several chosen applications created as a supplementary material for teaching logic systems.

Klíčová slova

logický systém, logický obvod, hradlo, klopný obvod, podpora výuky, počítačová hra, puzzle, Android, Java, Scala

Keywords

logic system, logic circuit, gate, latch, teaching aid, computer game, puzzle, Android, Java, Scala

Citace

Martin Kučera: Motivační výuková hra na platformě Android, diplomová práce, Brno, FIT VUT v Brně, 2015

Motivační výuková hra na platformě Android

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc.

.....
Martin Kučera
19. května 2015

Poděkování

Na tomto místě bych rád poděkoval vedoucí diplomové práce paní doc. RNDr. Jitce Kreslíkové, CSc., za přínosné rady a vstřícný přístup.

Obsah

Kapitola 1

Úvod

Práce se zabývá vývojem aplikace, která má, ideálně nenásilnou formou, představit základy logických systémů.

Proto se nejdříve věnuji úvodu do logických systémů, obvodů, hradel, klopných obvodů a formálnímu popisu systémů pomocí booleovských funkcí. Ve stejné části také nastíním téma simulace logických systémů.

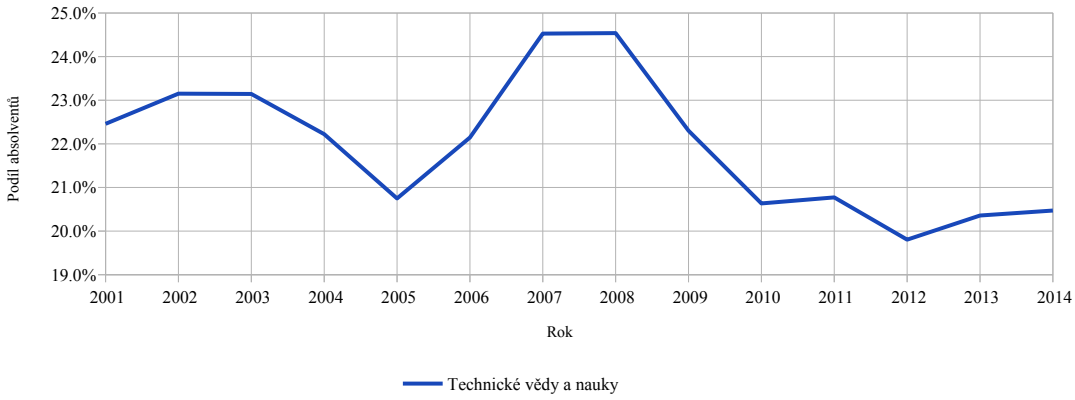
Poté následuje seznámení s počítačovými hrami a představení jejich žánrů se zaměřením na puzzle hry. Práce prozkoumá i historii puzzle her a u každé hry zdůrazní její důležité či originální vlastnosti.

V další kapitole budou popsány základy vývoje multiplatformních aplikací. Položím otázku, zda vůbec multiplatformně vyvíjet. Představím několik aktuálně populárních platforem a uvedu možnosti vývoje pro tyto platformy.

Následující kapitola se zabývá analýzou a srovnáním několika aplikací, především her, které se prezentují jako výukové aplikace zaměřené na logické operace, členy nebo celé obvody.

Zbývající kapitoly se zaměřují na prototyp aplikace, který je výstupem této práce. Postupně se rozepíši o specifikaci požadavků, návrhu, implementaci, ověření funkčnosti, zhodnocení a také o možných rozšířeních.

Dlouhou dobu pomalu klesá podíl absolventů technických oborů, viz obrázek 1.1. Je to jeden z důvodů volby tématu této práce. Pokud bude prototyp úspěšný, mohlo by se pokračovat ve vývoji a následně aplikaci uvolnit zdarma ke stažení.



Obrázek 1.1: Graf podílu absolventů technických oborů, zdroj dat: MŠMT ČR[10]

Kapitola 2

Logické systémy a obvody, možnosti popisu a simulace

Kapitola se zabývá představením logických systémů, obvodů, popisem jejich fungování a následně i hradly a klopnými obvody. Nakonec se dostane řada na možnosti simulace celých logických obvodů.

2.1 Logické systémy a logické obvody

2.1.1 Systém

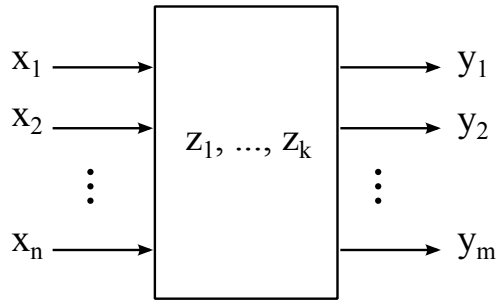
Do systému vstupuje n vstupních veličin $x_1, \ldots, x_n$ a vystupuje z něj m výstupních veličin $y_1, \ldots, y_m$. Vstupní veličiny x mění svoji hodnotu nezávisle na systému. Naopak výstupní veličiny závisí na vstupních veličinách. Tento vztah je umožněn dalším typem veličin – $z_1, \ldots, z_k$. Jedná se o vnitřní veličiny, které jsou většinou označovány jako **stavové**. Všechny tyto veličiny systému označujeme jako vstupní, výstupní nebo stavové proměnné.

2.1.2 Logický systém

Pokud hovoříme o *logickém* systému, pak všechny proměnné mohou nabývat pouze dvou hodnot, a to **0** a **1**. Když mluvíme o těchto hodnotách, tak jsou často označovány jako logické nebo booleovské hodnoty. V různých materiálech se můžeme setkat i s odlišných značením, např. **L** a **H** (z anglického low a high).

2.1.3 Prostředky pro popis vstupu, výstupu a stavu systému

K popisu vztahu mezi vstupy a výstupy obyčejně používáme vektory vstupních, výstupních a stavových proměnných. Uspořádaná n-tice n logických hodnot tvoří vektor. N-tici hodnot tvořenou hodnotami vstupních proměnných $x_1, \ldots, x_n$ označujeme jako *vstupní vektor* a o m-tici hodnot výstupních proměnných $y_1, \ldots, y_m$ mluvíme jako o *výstupním vektoru*.



Obrázek 2.1: Logický systém

Ku příkladu pro tři hodnoty na vstupu jsou vstupní proměnné značené x_1, x_2, x_3. Pak jeden vstupní vektor má takovouto podobu:

$$a = (a_1, a_2, a_3)$$

kde $a_i \in \{0, 1\}$ je specifická logická hodnota proměnné x_i, $i = 1, 2, 3$.

Pro n vstupních a m výstupních proměnných můžeme množinu $A = \{(a_1, \ldots, a_n) \mid a_i \in \{0, 1\}\}$ všech vstupních vektorů a množinu $U = \{(u_1, \ldots, u_n) \mid u_i \in \{0, 1\}\}$ všech výstupních vektorů vyjádřit jako následující kartézský součin

$$\mathbf{A} = \underbrace{\{0, 1\} \times \cdots \times \{0, 1\}}_{n\text{-krát}} = \{0, 1\}^n$$

$$\mathbf{U} = \{0, 1\}^m$$

kde $\{0, 1\}$ značí množinu logických hodnot.

Mějme následující příklad – tři vstupní proměnné x_1, x_2, x_3 a dvě výstupní proměnné y_1, y_2. Pak množiny všech vstupních a výstupních vektorů jsou následující:

$$\begin{aligned}\mathbf{A} =& \{0, 1\}^3 = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1),\\ & (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}\\ \mathbf{U} =& \{0, 1\}^2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}\end{aligned}$$

Vidíme, že v množinách $\mathbf{A}$ je 2^n a v $\mathbf{U}$ je 2^m různých vstupních či výstupních vektorů. V praxi však často nebudou množiny $\mathbf{A}$ a $\mathbf{U}$ úplné, protože některé kombinace vstupních nebo výstupních hodnot nebude okolí nebo systém nikdy generovat.

2.1.4 Stav systému

Stav systému udržují stavové proměnné. Hrají nenahraditelnou roli při vyjadřování vztahů mezi vstupy a výstupy, vyjadřují chování systému. Nechť je $\mathbf{S}$ množina všech možných stavů systému při k stavových proměnných $z_1, \ldots, z_k$, pak $\mathbf{S}$ bude splňovat

$$\mathbf{S} \subseteq \{0, 1\}^k$$

Je také možné se setkat s poněkud matoucí terminologií, jako například vstupní a výstupní *stavy* či *kombinace* namísto *vektorů* (či *hodnot proměnných*).

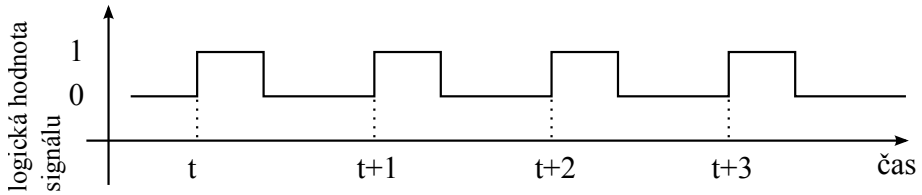
2.1.5 Čas v logickém systému

Hodnoty výstupních a stavových proměnných se v průběhu času mění v závislosti na vstupních proměnných, logické systémy jsou tedy dynamické. Při popisu logického systému se používá diskrétní čas, který je definován jako posloupnost bodů ve spojitém čase

$$t_0, t_1, t_2, \ldots \text{ kde } t_0 < t_1 < t_2 \ \ldots$$

V každém bodě času t_i je pro systém definován určitý vstupní vektor, vnitřní stav a výstupní vektor. Konkrétní hodnoty t_i nejsou běžně významné, co je ale důležité je pořadí těchto bodů. Proto nám pro popis bodů v diskrétním čase většinou stačí celá nebo přirozená čísla. Diskrétní proměnná t vyjadřující aktuální čas tedy může nabývat například hodnot 0, 1, 2, 3, 4, ...

Typicky jsou body v diskrétním čase určeny tak, aby se kryly se změnami proměnných v systému. U synchronních systémů jsou tyto body určeny přímo změnami specifické proměnné, která se běžně nazývá hodinová nebo synchronizační proměnná. Synchronní systém může reagovat na změnu hodinové proměnné $0 \rightarrow 1$ (ukázka na obr. 2.2), $1 \rightarrow 0$, nebo na obě varianty zároveň. Tuto proměnnou generuje zařízení pojmenované jako *generátor hodin*, hodinová proměnná se typicky nepovažuje za vstupní proměnnou systému.



Obrázek 2.2: Příklad hodinového signálu

2.1.6 Chování a stavba systému

V každém logickém systému existují vztahy mezi hodnotami vstupních a výstupních proměnných. Množinu všech těchto vztahů, které jsou charakteristické pro daný systém, nazýváme *chování systému*.

Logické systémy, pomineme-li ty elementární, jsou tvořeny kompozicí vzájemně propojených jednodušších logických systémů. Tato kompozice se běžně označuje jako *struktura systému*. Formálně je možné strukturu systému vyjádřit dvojicí $(\mathscr{P}, \mathscr{V})$, kde $\mathscr{P} = \{P_1, \ldots, P_i\}$ je množina prvků struktury (logické pod-systémy) a $\mathscr{V}$ je množina vazeb mezi prvky systému. Způsob formálního popisu je možno volit z více přístupů. Základním prostředkem pro popis systému je orientovaný ohodnocený multigraf, kde vrcholy vyjadřují konkrétní prvky systému a orientované hrany naznačují vazby mezi prvky.

U logického systému je nejnižší rozlišovací úrovní úroveň logických proměnných v diskrétním čase (jejich hodnoty jsou mapovány na fyzikální veličiny ve spojitém čase). V popisu systému nepracujeme se vztahy mezi fyzikálními veličinami ve spojitém čase, přestože tyto vztahy existují a jsou důležité pro následnou implementaci v technických zařízeních.

Naopak nejvyšší rozlišovací úrovní jsou vektory hodnot logických proměnných. V této práci se budeme zabývat primárně vztahy mezi logickými hodnotami, ne mezi vektory logických hodnot (ty jsou stěžejní pro systémy pracující na vyšší rozlišovací úrovni, tam slouží k vyjádření např. čísla, instrukce nebo znaku). S vektory logických hodnot pracují složitější číslicové (digitální) systémy. Tyto systémy pracují s diskrétní formou informace, která je nazývaná *bit* (z anglického binary digit). Pak o vektoru hodnot mluvíme jako o n-bitovém vektoru, slovu nebo řetězci.

2.1.7 Logický obvod

Implementaci logického systému na konkrétním technickém zařízení (běžně elektronické zařízení) nazýváme *logický obvod*. Struktura a chování logického systému v diskrétním čase popisuje stavbu a chování logického obvodu. Z logických obvodů, základních stavebních bloků, se budují číslicové počítače a další číslicová zařízení.

2.1.8 Rozdělení logických systémů

Podle toho, jak se logické systémy chovají, je rozdělujeme na dvě skupiny: kombinační a sekvenční systémy (na stejné skupiny se dělí i obvody).

Kombinační logický systém (nebo obvod) je jednodušší z dvojice. Chování kombinačního systému je možno vyjádřit následující funkcí, běžně označovanou jako *funkce kombinačního obvodu*:

$$f : \mathbf{A} \to \mathbf{U}$$

kde $\mathbf{A}$ je množina vstupních a $\mathbf{U}$ množina výstupních vektorů.

Funkci f lze popsat tak, že každému vstupnímu vektoru $a \in \mathbf{A}$ přiřazuje jeden výstupní vektor $u = f(a), u \in \mathbf{U}$. Povšimněme si, že výstupní vektor závisí pouze na vstupním vektoru.

Sekvenční logický systém je typický tím, že jeho výstupní vektor (na rozdíl od kombinačního systému) závisí nejen na vstupním vektoru, ale i na posloupnosti (sekvenci) vstupních vektorů v minulosti. Z toho plyne, že pro stejné vstupní vektory (ne nutně stejnou historii) může tento typ systému generovat různé výstupní vektory. Aby tohoto chování mohl sekvenční logický systém dosáhnout, musí mít schopnost uchování informace – schopnost pamatovat si data z minulosti. Uloženou informaci následně použije při výpočtu výstupu.

Formálně lze popsat chování sekvenčního logického systému funkcí F_a:

$$F_a : \mathbf{A}^* \to \mathbf{U}^*$$

kde $\mathbf{A}^*$ a $\mathbf{U}^*$ jsou nekonečné množiny všech posloupností s konečnou délkou, jsou sestaveny z množiny vektorů $\mathbf{A}$ resp. $\mathbf{U}$. Tyto posloupnosti nazýváme *vstupní* či *výstupní slova*.

Funkce F_a přiřazuje každému vstupnímu slovu jedno slovo výstupní se stejnou délkou. Posloupnost $a_0 a_1 a_2 \dots a_r$ je slovo složené z vektorů, kde dolní indexy značí bod v diskrétním čase. Pro nějaké vstupní slovo e (ze startovního času $t = 0$) generuje výstupní slovo $w = F_a(e)$.

Funkci F_a označujeme jako *funkci sekvenčního systému*. Zápis funkce je zkomplikován tím, že definiční obor a obor hodnot (koobor) F_a jsou nekonečné množiny. Tato nepříjemnost je vyřešena zavedením *stavu systému*, pak lze zapsat chování pomocí dvou funkcí p a v, které mají konečné definiční obory i obory hodnot.

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{p}(\mathbf{s}(t), \mathbf{a}(t)) \\ \mathbf{u}(t) &= \mathbf{v}(\mathbf{s}(t), \mathbf{a}(t)) \end{aligned}$$

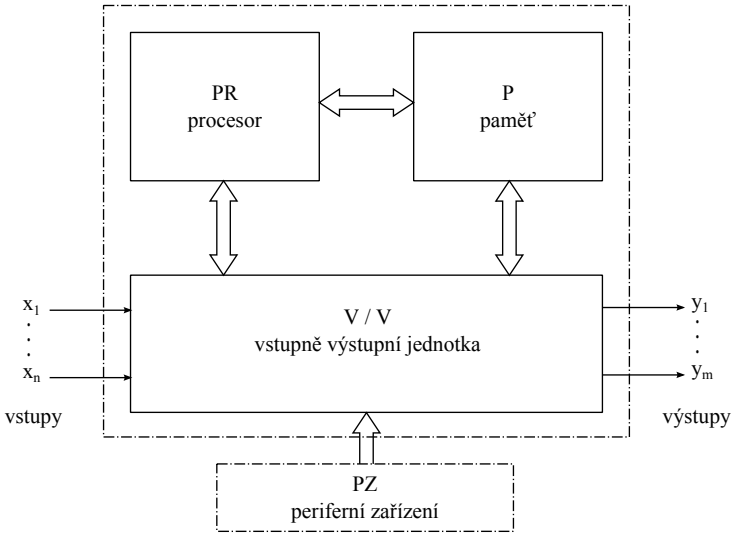
kde $\mathbf{a}(t)$, $\mathbf{s}(t)$ a $\mathbf{u}(t)$ odpovídají vstupnímu vektoru, stavovému vektoru a výstupnímu vektoru v čase t, $\mathbf{s}(t+1)$ popisuje *následující stav* v čase $t+1$. Nově zavedená funkce p předepisuje následující stav systému a v určuje aktuální výstupní vektor. Tyto dvě funkce, které užívají vektor popisující stav systému, výrazně zjednodušily zápis – není už potřeba historie vstupů A^*.

2.1.9 Pevná a programovatelná funkce

Logické systémy (a logické obvody) lze rozdělit podle způsobu, jakým je implementována funkce popisující chování.

Systémy s pevnou funkcí jsou charakteristické tím, že jejich chování je dáno jejich konkrétní strukturou navrženou speciálně pro danou funkčnost. Pokud chceme změnit chování, musíme změnit strukturu systému.

Pro *systémy s programovatelnou funkcí* je typické, že jejich funkce je řízená *programem* uloženým v paměťovém podsystému. Program lze přepsat a tím upravit i chování celého systému beze změny struktury systému. Program se skládá z po sobě jdoucích příkazů kódovaných v booleovských vektorech. Takto zapsané příkazy se označují jako *strojové instrukce* nebo zkráceně pouze *instrukce*.



Obrázek 2.3: Typická obecná struktura logického systému s programovatelným chováním

Systém s programovatelnou funkcí má svou strukturu univerzální. Typické rozdělení systému si můžeme prohlédnout na obr. 2.3. **P** je paměť s přímým přístupem – RAM (**R**andom **A**ccess **M**emory). Procesor **PR** je zodpovědný za provádění programu, který je uložen v paměti. Provádění programu (také označované jako *interpretace*) je proces, ve kterém procesor postupně načítá instrukce z paměti a vykonává je. Instrukce nemusí nutně ležet za sebou – např. instrukce skoku zapříčiní, že adresa následující instrukce nemusí být adresou instrukce, která se nachází za instrukcí skoku. Vstupně-výstupní jednotka má na starosti přijímaní a dekódování vstupu (získává vstupní vektor) a kódování a odesílání výstupu (výsledku).

Velmi důležitá skupina implementující logické systémy s programovatelnou funkcí jsou *univerzální mikropočítače* a *mikrokontroléry* často označované jako *jednočipové počítače*. Univerzální mikropočítače a mikrokontroléry nelze použít na všechny úlohy, které mohou řešit logické systémy s pevnou funkcí. Pouze na množinu úloh, kde není doba odezvy kritická, univerzální mikrokontroléry a mikropočítače jsou totiž podstatně pomalejší než obvod speciálně navržený na danou úlohu. Tento nedostatek lze částečně zmírnit užitím specializovaného procesoru, který má instrukční sadu vhodnou pro řešení specifického problému.

2.1.10 Hlavní úlohy návrhu logických obvodů

Prvním úkolem je nalezení a popis chování logického obvodu ze struktury obvodu. Tento úkol se nazývá *analýza logických obvodů*.

Druhý úkol představuje *syntéza logických obvodů*, kde na vstupu máme formálně nebo slovně formulované zadání a výstupem očekáváme strukturu systému, která splňuje požadavky na chování a parametry systému.

Syntéza není obecně jednoznačná, pro ne úplně jednoduché problémy totiž většinou existuje velké množství řešení splňující zadání. Pak se hledá tzv. optimální řešení podle zadaných kriteriálních funkcí, proces se označuje jako *optimální syntéza*. Úkol optimální syntézy je velmi komplikovaný, přesné algoritmické řešení je známé pouze pro několik vybraných obvodů a některá kritéria optimálnosti. Složitost řešení tohoto typu úloh na počítači je exponenciální vzhledem k počtu prvků obvodu.

U systémů s programovatelnou funkcí se běžně nenavrhuje univerzální mikropočítač, úkolem

je napsat program pro již existující zařízení.

Další poměrně významná činnost při návrhu logických obvodů je *modelování a simulace logických systémů*. Jde o navržení odpovídajícího matematického modelu obvodu a následném experimentování nad tímto modelem – sledujeme reakce na různé vstupy (často v jiném časovém měřítku) anebo procházíme historii chování systému.

2.1.11 Zdroje

Hlavním zdrojem této části o logických systémech byla kniha [16], podstatně méně informací bylo čerpáno z [48] a [45].

2.2 Logické funkce

Následující text je založen především na [48] a [16], dále bylo čerpáno také z [26].

2.2.1 Typy booleovských funkcí

Obecná logická funkce je popsána tzv. booleovskou funkcí (někdy označovaná jako B-funkce) $f(x_1, x_2, \ldots, x_n)$, která je zobrazením z n booleovských hodnot do jedné booleovské hodnoty.

$$f : \{0,1\}^n \to \{0,1\}$$

Definičním oborem výše uvedené booleovské funkce je množina

$$\{0,1\}^n = \bigtimes_{i=1}^{n} \{0,1\}$$

což je množina všech možných n-tic s prvky 0 a 1. Oborem hodnot funkce je množina $\{0,1\}$, tj. *booleovská* nebo také *logická hodnota*. Obecná booleovská (logická) funkce tedy přiřazuje každé vstupní n-tici booleovských hodnot právě jednu výstupní booleovskou hodnotu (tj. 0 nebo 1). Příklad je vyveden v tabulce 2.1a.

x_1	x_2	f
0	0	0
1	0	1
0	1	1
1	1	1

(a)

x_1	x_2	f
0	0	0
1	0	1
0	1	1

(b) Neúplná b. funkce

x_1	x_2	f
0	0	0
1	0	1
0	1	1
1	1	x

(c) Rozšířená b. funkce

Tabulka 2.1: Boolovské (logické) funkce

V praktických úlohách se objevují tzv. neúplné booleovské funkce. Takové funkce nemají definiční obor odpovídající $\{0,1\}^n$, ale pouze jeho podmnožině. Neúplná booleovská funkce $f(x_1, x_2, \ldots, x_n)$ je zobrazení z $\mathbf{Q}$ do $\{0,1\}$, které je definované následovně

$$f : \mathbf{Q} \to \{0,1\}, \text{kde } \mathbf{Q} \subset \{0,1\}^n$$

V tabulce 2.1b vidíme příklad neúplné booleovské funkce. V bodě 11 není definována, takovéto body označujeme jako *nedefinované body* funkce.

V praxi se nejčastěji užívají tzv. *rozšířené booleovské funkce*. Jde o úplné booleovské funkce mající v oboru hodnot kromě 0 a 1 také hodnotu „x“, která popisuje hodnotu neurčitou – funkce v tomto bodě může nabývat hodnoty 0 nebo 1.

$$f(x_1, x_2, \ldots, x_n) : \{0,1\}^n \to \{0,1,x\}$$

V tabulce 2.1c je ukázán příklad rozšířené funkce (shodná s funkcí OR kromě hodnoty v 11).

Funkci f' (úplnou, neúplnou nebo rozšířenou) nazýváme *rovnocennou* s funkcí f právě tehdy, když mají kromě neurčených bodů stejné hodnoty 0 či 1. Můžeme si zkontrolovat, že funkce z tabulek 2.1a, 2.1b a 2.1c jsou rovnocenné.

Pro jednu určitou booleovskou funkci f je 2^k totálních funkcí, které jsou s ní rovnocenné. Proměnná k značí počet neurčených bodů v oboru funkce f.

Ku příkladu k funkci f z tabulky 2.1c je $2^1 = 2$ různých úplných funkcí, které se jedna od druhé liší v různých hodnotách 0 a 1 v onom jednom neurčeném bodě funkce f.

2.2.2 Vzdálenost booleovských funkcí

Při návrhu logických systémů je vhodné mít možnost, jak hodnotit míru rozdílů mezi dvěma booleovskými funkcemi. Proto definujeme vzdálenost $d(l, j)$ dvou n-tic (ekvivalentní s vektory či body funkce) l a j z množiny $\{0,1\}^n$ jako počet míst, ve kterých se tyto dvě n-tice odlišují. Zavedená vzdálenost se jmenuje *Hammingova vzdálenost*. Právě tehdy, když dvě n-tice mají vzdálenost rovnu jedné, tak se označují jako *sousedící*.

V dalším textu budeme používat následující konvenci: vektory logických (booleovských) hodnot budou reprezentovány jako řetězec symbolů 0 a 1. Například vektor $(0,1,0)$ budeme zkracovat na 010.

Nechť máme dva vektory 1010 a 0110, pak jejich vzdálenost odpovídá $d(1010, 0110) = 2$ a nejsou proto sousedící.

Spolu s množinou $\{0,1\}^n$ tvoří zavedená funkce vzdálenosti tzv. *metrický prostor*, protože splňuje následující tvrzení:

1. $d(l, j) \geq 0$
2. $d(l, j) = d(j, l)$
3. $d(l, j) = 0 \iff l = j$
4. $d(l, k) \leq d(l, j) + d(j, k)$ (trojúhelníková nerovnost)

Více o metrikách a metrických prostorech včetně příkladů může čtenář nalézt ve skriptech [26].

2.2.3 Užití booleovských funkcí

Logické funkce se používají pro popis kombinačních obvodů (tj. obvodů, kde nezáleží na historii stavů). Chování logického systému, který popisuje nějaký logický obvod, je tedy možno popsat uspořádanou m-ticí booleovských funkcí $f_1, f_2, \ldots, f_m$:

$$y_j = f_j(x_1, x_2, \ldots, x_n), \text{ kde } j = 1, 2, \ldots, m$$

Mějme logický systém se dvěma vstupy a dvěma výstupy (vyobrazen na obrázku 2.4). Obecně budou funkce vypadat následovně

$$y_1 = f_1(x_1, x_2)$$
$$y_2 = f_2(x_1, x_2)$$

Obrázek 2.4: Logický systém se dvěma vstupy a dvěma výstupy

Řekněme, že první výstup bude ekvivalentní s funkcí XOR a druhý s funkcí AND. Tuto skutečnost můžeme zapsat takto

$$y_1 = f_1(x_1, x_2),\ f_1(x_1, x_2) = x_1 \oplus x_2$$
$$y_2 = f_2(x_1, x_2),\ f_2(x_1, x_2) = x_1 \wedge x_2$$

V tabulce 2.2a vidíme všechny kombinace vstupních hodnot s patřičnými výstupy, tabulka 2.2b ukazuje zápis pomocí vektorů.

$$f_j = \mathbf{A} \to \{0, 1\}, j = 1, 2$$

x_1	x_2	f_1	f_2
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

(a) Funkční

$(x_1,\ x_2)$	$(y_1,\ y_2)$
0 0	0 0
1 0	1 0
0 1	1 0
1 1	0 1

(b) Vektorový

Tabulka 2.2: Popis chování pomocí B-funkcí

2.3 Logické výrazy

V předchozí části jsme se setkali s popisem chování logické funkce $x_1 \wedge x_2$, obecně se tato notace nazývá *logický výraz*. Jde o řetězec symbolů složený z konstant $\{0, 1\}$, proměnných a operací $\{\neg, \wedge, >, <, \oplus, \vee, \widetilde{\vee}, \leftrightarrow, \to, \leftarrow, \widetilde{\wedge}\}$. Pojem logický výraz budeme v dalším textu zkracovat pouze na výraz.

2.3.1 Logické spojky

Operátory nad logickými výrazy se nazývají *logické spojky* (anglicky *logic connectives*). Pokud mají dva vstupní parametry, pak se nazývají *binární logické spojky* (nebo také dyadické logické spojky). Jediná spojka v seznamu, která očekává pouze jeden vstupní parametr, je negace $\neg$. Spojky s jedním vstupním parametrem jsou označovány jako *unární logické spojky* (méně často nazývané jako monadické logické spojky). Symboly $>$ a $<$ značí inhibici a transponovanou inhibici, $\widetilde{\vee}$ NOR a $\widetilde{\wedge}$ NAND. Značení inhibice se v literatuře různí, zde použitá pochází z jazyka APL. Značky pro NOR a NAND jsou původem také z APL z dílny pana Iversona. V knize "Logic Circuit Design" [48] se rozhodl autor nepoužívat obvyklou notaci především proto, že Iversonovůs zápis se lépe pamatuje a působí logičtěji, s tím plně souhlasím a proto také budu používat tento nový zápis. Posuďte sami. *Shefferova funkce*:

$$x_1 \downarrow x_2 \Leftrightarrow x_1 \widetilde{\wedge} x_2 \Leftrightarrow \neg(x_1 \wedge x_2)$$

Nicodova funkce (někdy také označována jako Piercova funkce):

$$x_1 | \, x_2 \Leftrightarrow x_1 \uparrow x_2 \Leftrightarrow x_1 \mathbin{\widetilde{\vee}} x_2 \Leftrightarrow \neg(x_1 \vee x_2)$$

2.3.2 Elementární logické funkce

Nyní, když jsme zavedli operace nad logickými proměnnými, můžeme sepsat kompletní tabulku tzv. elementárních logických funkcí. Jde o takové logické funkce, které mají nejvíce dvě vstupní

Definice $X_1 = 0011$ $X_2 = 0101$	Zápis	Jméno logické funkce
$Y_0 = 0000$	$Y_0 \Leftrightarrow 0$	kontradikce, nepravda
$Y_1 = 0001$	$Y_1 \Leftrightarrow X_1 \wedge X_2$	konjunkce, AND
$Y_2 = 0010$	$Y_2 \Leftrightarrow X_1 > X_2$	inhibice
$Y_3 = 0011$	$Y_3 \Leftrightarrow X_1$	identita
$Y_4 = 0100$	$Y_4 \Leftrightarrow X_1 < X_2$	transponovaná inhibice
$Y_5 = 0101$	$Y_5 \Leftrightarrow X_2$	identita
$Y_6 = 0110$	$Y_6 \Leftrightarrow X_1 \oplus X_2$	antivalence, exkluzivní OR, XOR
$Y_7 = 0111$	$Y_7 \Leftrightarrow X_1 \vee X_2$	disjunkce, OR
$Y_8 = 1000$	$Y_8 \Leftrightarrow X_1 \mathbin{\widetilde{\vee}} X_2$	NOR, Nicodova (Piercova) funkce
$Y_9 = 1001$	$Y_9 \Leftrightarrow X_1 \leftrightarrow X_2$	ekvivalence
$Y_{10} = 1010$	$Y_{10} \Leftrightarrow \overline{X_2}$ (také $\neg X_2$)	negace, NOT
$Y_{11} = 1011$	$Y_{11} \Leftrightarrow X_1 \leftarrow X_2$	transponovaná implikace
$Y_{12} = 1100$	$Y_{12} \Leftrightarrow \overline{X_1}$	negace, NOT
$Y_{13} = 1101$	$Y_{13} \Leftrightarrow X_1 \rightarrow X_2$	implikace
$Y_{14} = 1110$	$Y_{14} \Leftrightarrow X_1 \mathbin{\widetilde{\wedge}} X_2$	NAND, Shefferova funkce
$Y_{15} = 1111$	$Y_{15} \Leftrightarrow 1$	tautologie, pravda

Tabulka 2.3: Elementární logické funkce

proměnné a právě jednu výstupní proměnnou. Přitom vstupní (pak také výstupní) proměnná může být buďto logická hodnota, nebo vektor logických hodnot. V tabulce 2.3 jsou vypsány všechny elementární logické funkce.

2.3.3 Funkční úplnost

Z tabulky je zřejmé, že existuje poměrně velký počet elementárních logických funkcí. Běžně se ale zdaleka nepoužívají všechny. Naopak se často používají pouze dvě nebo tři logické funkce. Pokud pomocí množiny logických operátorů je možno vyjádřit jakoukoliv logickou funkci, pak tuto množinu adresujeme jako *funkčně úplná množina (logických spojek)*. V tabulce 2.4 se nachází v nejlevějším a nejpravějším sloupci. Každá logická funkce může být popsána za užití pouze AND, OR a NOT spojek. Množina $\{\wedge, \vee, \neg\}$ je tedy funkčně úplná množina. To znamená, že pokud můžeme množinou spojek S vyjádřit všechny spojky jiné funkčně úplné množiny, pak je i daná množina S funkčně úplná. V tabulce vidíme v nekrajových sloupcích vyjádření AND, OR a NOT pomocí spojek z prvního sloupce.

V pravém sloupci se nachází tzv. duální množiny, které obsahují duální spojky k původní množině. Přídavné jméno „duální“ znamená, že pokud nahradíme pravdivostní hodnoty a pů-

množina	$\neg A$	$A \wedge B$	$A \vee B$	duální množina
$\{\widetilde{\wedge}\}$	$A \mathbin{\widetilde{\wedge}} A$	$(A \mathbin{\widetilde{\wedge}} B) \mathbin{\widetilde{\wedge}} (A \mathbin{\widetilde{\wedge}} B)$	$(A \mathbin{\widetilde{\wedge}} A) \mathbin{\widetilde{\wedge}} (B \mathbin{\widetilde{\wedge}} B)$	$\{\widetilde{\vee}\}$
$\{\wedge, \neg\}$	$\neg A$	$A \wedge B$	$\neg(\neg A \wedge \neg B)$	$\{\vee, \neg\}$
$\{\rightarrow, 0\}$	$A \rightarrow 0$	$(A \rightarrow (B \rightarrow 0)) \rightarrow 0$	$(A \rightarrow B) \rightarrow B$	$\{>, 1\}$
$\{\rightarrow, \neg\}$	$\neg A$	$\neg(A \rightarrow \neg B)$	$\neg A \rightarrow B$	$\{>, \neg\}$
$\{\oplus, \wedge, 1\}$	$A \oplus 1$	$A \wedge B$	$A \oplus B \oplus (A \wedge B)$	$\{\leftrightarrow, \vee, 0\}$
$\{\oplus, \vee, 1\}$	$A \oplus 1$	$A \oplus B \oplus (A \vee B)$	$A \vee B$	$\{\leftrightarrow, \wedge, 0\}$

Tabulka 2.4: Vybrané funkčně úplné množiny logických spojek

vodní spojky jejich duálními protějšky, pak dostaneme výrazy, které jsou ekvivalentní s výrazy původními. Tabulka 2.5 přehledně zobrazuje základní logické funkce s jejich duálními tvary.

originální	0	$\wedge$	$>$	$<$	$\oplus$	$\vee$	$\widetilde{\vee}$	$\leftrightarrow$	$\neg$	$\leftarrow$	$\rightarrow$	$\widetilde{\wedge}$	1
duální	1	$\vee$	$\leftarrow$	$\rightarrow$	$\leftrightarrow$	$\wedge$	$\widetilde{\wedge}$	$\oplus$	$\neg$	$>$	$<$	$\widetilde{\vee}$	0

Tabulka 2.5: Duální hodnoty a spojky

2.3.4 Vlastnosti spojek

V prvé řadě si zavedeme často používanou konvenci priorit logických spojek. Tabulka 2.6 zobrazuje spojky podle jejich priority. Díky této konvenci můžeme vynechávat závorky a zpřehlednit

spojka	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
priorita	1	2	3	4	5

Tabulka 2.6: Priorita logických spojek

tak zápis výrazu. Ukažme si to na příkladu:

$$\Big((A \wedge B) \vee \big((\neg A) \wedge B\big)\Big) \leftrightarrow B \Leftrightarrow$$
$$(A \wedge B) \vee \big(\neg A \wedge B\big) \leftrightarrow B \Leftrightarrow$$
$$A \wedge B \vee \neg A \wedge B \leftrightarrow B$$

Další obvyklá konvence je vynechávání $\wedge$ členu, nyní dostáváme tento výraz:

$$AB \vee \neg AB \leftrightarrow B$$

který může být zapsán i takto (jiná notace negace):

$$AB \vee \bar{A}B \leftrightarrow B$$

Představme si dvě binární spojky $\bullet$ a $\circ$, které použijeme pro popis několika vlastností.

$$\bullet, \circ \in \{\wedge, <, >, \oplus, \vee, \widetilde{\vee}, \leftrightarrow, \leftarrow, \rightarrow, \widetilde{\wedge}\}$$

Spojka $\bullet$ je *komutativní* pokud platí

$$A \bullet B \Leftrightarrow B \bullet A$$

Spojka $\bullet$ je *asociativní* pokud platí

$$A \bullet (B \bullet C) \Leftrightarrow (A \bullet B) \bullet C$$

Spojka $\bullet$ je *zleva distributivní* vůči spojce $\circ$ pokud platí

$$A \bullet (B \circ C) \Leftrightarrow (A \bullet B) \circ (A \bullet C)$$

Podobně platí, že spojka $\bullet$ je *zprava distributivní* ke spojce $\circ$ pokud

$$(B \circ C) \bullet A \Leftrightarrow (B \bullet A) \circ (C \bullet A)$$

V případě, že spojka $\bullet$ je distributivní zleva i zprava vůči $\circ$, pak říkáme, že spojka $\bullet$ je distributivní vůči $\circ$.

Další zajímavou vlastností je *idempotence*, kterou lze popsat tak, že ze vstupu (vstupů) A po aplikaci operace dostáváme opět vstup A. Logické funkce AND i OR jsou idempotentní, protože platí:

$$A \wedge A \Leftrightarrow A$$
$$A \vee A \Leftrightarrow A$$

Pro funkce AND a OR existuje tzv. *neutrální prvek*. Mějme na vstupu neutrální prvek a libovolnou hodnotu A, pak po vyčíslení operace dostaneme opět hodnotu A. Pro sčítaní a OR platí, že neutrálním prvkem je 0, protože

$$A + 0 = A$$
$$A \vee 0 \Leftrightarrow A$$

Neutrální prvek pro násobení a AND je, jak můžeme tušit, 1, protože následující výroky jsou pravdivé:

$$A \cdot 1 = A$$
$$A \wedge 1 \Leftrightarrow A$$

Ekvivalence a XOR mají neutrální prvky 1 a 0. Ani jedna z operací z předchozí věty ale není idempotentní.

$$A \leftrightarrow 1 \Leftrightarrow A$$
$$A \oplus 0 \Leftrightarrow A$$

V praxi se upřednostňují komutativní logické spojky, protože nezáleží na pořadí vstupů. NAND a NOR jsou také komutativní, jako všechny spojky zmíněné v této podkapitole, ale nemají neutrální prvek ani nejsou idempotentní. Implikace ani inhibice nejsou komutativní.

Asociativita platí jen pro podmnožinu komutativních logických funkcí. Vyjmenujme několik nejznámější logických funkcí, které jsou zároveň komutativní i asociativní: AND $\wedge$, OR $\vee$, ekvivalence $\leftrightarrow$ a XOR $\oplus$.

Při úpravách logického výrazu jsou velmi užitečné *De Morganovy* zákony, které popisují vztahy mezi sjednocením, průnikem a doplňkem. Nejčastěji tyto zákony vidíme aplikované na spojky AND a OR. Jak vidíme v tabulce 2.7, lze je použít na velké množství spojek.

$$\begin{array}{l|l}
\overline{A \wedge B} \Leftrightarrow \overline{A} \vee \overline{B} & \overline{A \vee B} \Leftrightarrow \overline{A} \wedge \overline{B} \\
\overline{A > B} \Leftrightarrow \overline{A} \leftarrow \overline{B} & \overline{A \leftarrow B} \Leftrightarrow \overline{A} > \overline{B} \\
\overline{A < B} \Leftrightarrow \overline{A} \rightarrow \overline{B} & \overline{A \rightarrow B} \Leftrightarrow \overline{A} < \overline{B} \\
\overline{A \oplus B} \Leftrightarrow \overline{A} \leftrightarrow \overline{B} & \overline{A \leftrightarrow B} \Leftrightarrow \overline{A} \oplus \overline{B} \\
\overline{A \mathbin{\tilde{\vee}} B} \Leftrightarrow \overline{A} \mathbin{\tilde{\wedge}} \overline{B} & \overline{A \mathbin{\tilde{\wedge}} B} \Leftrightarrow \overline{A} \mathbin{\tilde{\vee}} \overline{B}
\end{array}$$

Tabulka 2.7: De Morganovy zákony

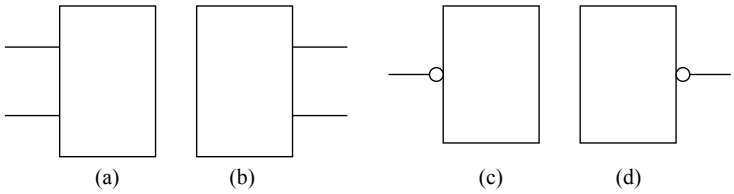
Pravidlo negace popisuje, jak vytvořit doplněk nebo znegovat libovolný logický výraz. Zní následovně: nahraď všechny proměnné a konstanty jejich doplňkem a každou spojku její duální variantou. Níže se nachází detailní příklad použití pravidla negace.

$$\begin{aligned}
\neg\Big(\big((A \ \vee \neg CD)AB \vee C\big)\neg D\Big) \Leftrightarrow \quad & \overline{\big((A \ \vee \overline{C}D)AB \vee C\big)\overline{D}} \Leftrightarrow \\
\overline{(A \ \vee \overline{C}D)AB \vee C} \vee \overline{\overline{D}} \Leftrightarrow \quad & \overline{(A \ \vee \overline{C}D)AB \vee C} \vee D \Leftrightarrow \\
\Big(\overline{(A \vee \overline{C}D)AB} \wedge \overline{C}\Big) \vee D \Leftrightarrow \quad & \big(\overline{A \vee \overline{C}D} \vee \overline{AB}\big)\overline{C} \vee D \Leftrightarrow \\
\big((\overline{A} \wedge \overline{\overline{C}D}) \vee \overline{A} \vee \overline{B}\big)\overline{C} \vee D \Leftrightarrow & \big(\overline{A}(C \vee \overline{D}) \vee \overline{A} \vee \overline{B}\big)\overline{C} \vee D
\end{aligned}$$

2.4 Hradla

Pojem *hradlo* značí grafický symbol, který se váže k určité logické funkci. Chování hradel je ideální. Zařízení reálně vyráběná se snaží co nejvíce přiblížit jejich fungování. Ne všechny logické funkce existují ve formě hradla, nejvíce se používají hradla reprezentující základní logické funkce.

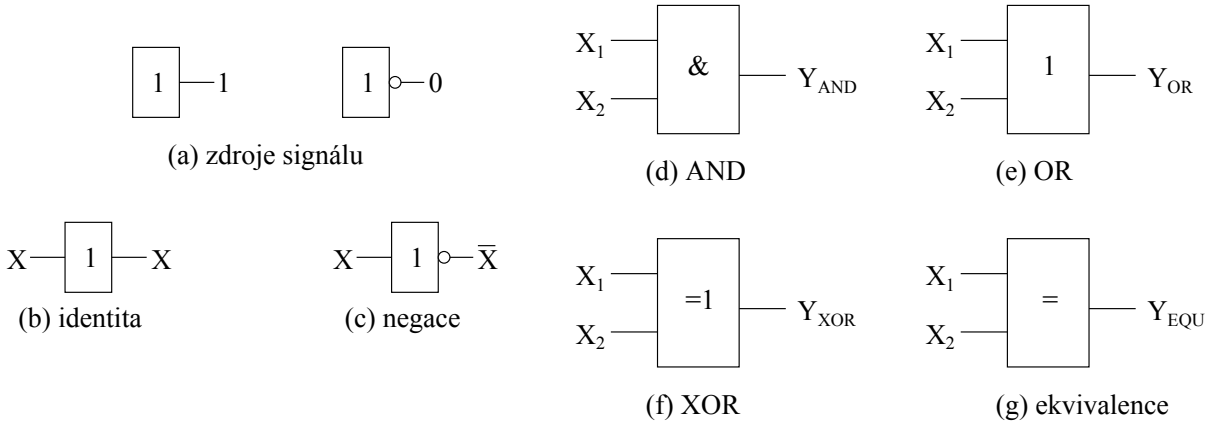
V textu práce i v aplikaci používám značení dané normou IEC 60617-12. Standardní grafickou podobu hradla, především jeho vstupů a výstupů, si můžeme prohlédnout na obrázku 2.5 [13]. Část (a) vyobrazuje vstupy a (b) výstupy. Negace se znázorňuje kolečkem na konci vstupu či na začátku výstupu (výrazy začátek a konec chápeme ve směru toku informace). Pak tedy část (c) představuje negovaný vstup a (d) negovaný výstup.



Obrázek 2.5: Grafické znázornění vstupů a výstupů hradel

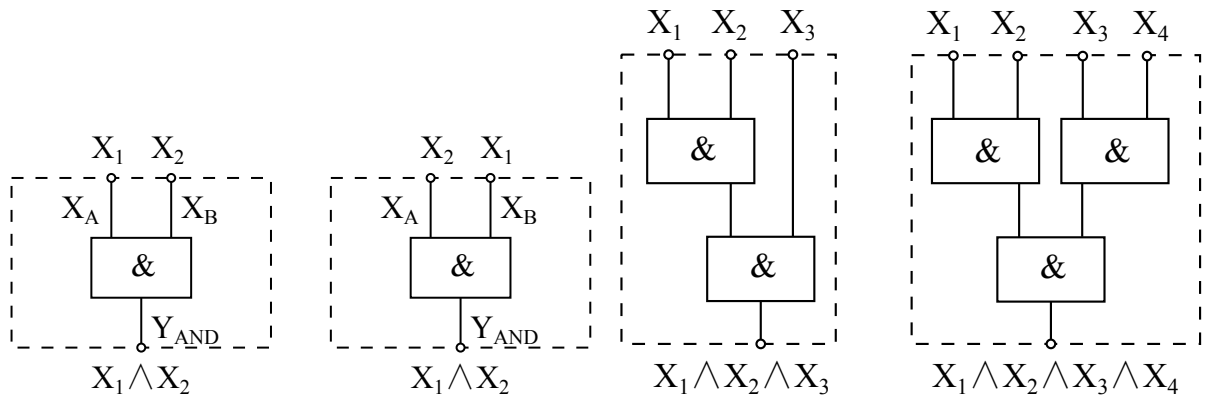
Na obrázku 2.6 vidíme základní logická hradla. První jsou tzv. zdroje nebo také generátory signálu. Další člen je identita, která se někdy označuje jako sledovač. Následuje negace, v literatuře také pojmenovaná jako invertor, negátor nebo negační obvod. (d) je AND člen, značený také jako součinový obvod nebo konjunktor. OR vystupuje pod štítkem (e). Tento člen je občas pojmenován jako součtový obvod nebo disjunktor. Text na hradlu nemusí být jen „1“, lze se setkat i s „≥ 1“. Následuje XOR – exkluzivní součet. Posledním základním hradlem je ekvivalence neboli shodnost. Grafická podoba se může lehce lišit, někdy jsou používaná dvě rovnítka „==“ místo jednoho „=“.

Jak bylo zmíněno dříve, nejvíce se užívají hradla těchto základních logických funkcí. Je to dáno tím, že s funkcemi, které jsou asociativní a komutativní, se pracuje nejsnadněji. V praxi se



Obrázek 2.6: Hradla základních logických funkcí

nepoužívají jen nejjednodušší varianty se dvěma vstupy, ale běžně vidíme hradla se třemi a více vstupy.



(a) Komutativita (b) Asociativita

Obrázek 2.7: Důležité vlastnosti hradel

Díky *komutativitě* je možné libovolně přeskládávat pořadí vstupů bez jakéhokoliv vlivu na výstup. Příklad vidíme na obrázku 2.7a.

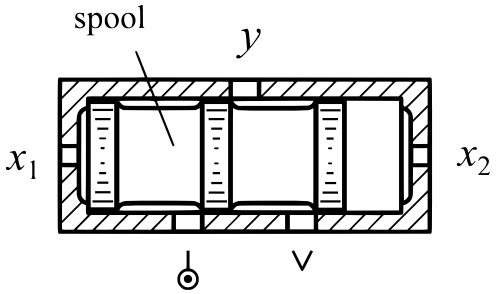
Asociativita umožňuje jednoduché vytváření vícevstupých variant konkrétního hradla. Obrázek 2.7b zachycuje třívstupé a čtyřvstupé hradlo AND.

2.5 Klopné obvody

2.5.1 Základy

Klopné obvody (anglicky *latch* nebo *flip-flop*, podle typu) se na rozdíl od jednodušších hradel liší především tím, že obsahují vnitřní paměť. Hradla představují kombinační obvody a klopné obvody patří mezi obvody sekvenční.

Přestože pojem „klopný obvod“ je v českém jazyce svázán s elektronickým provedením, vzor překladu – latch – může být klidně v provedení mechanickém. Dobrý příklad je prezentován v [48], můžete si jej prohlédnout na obrázku 2.8. Ventil si pamatuje poslední pulz z jednoho ze vstupů x_1, x_2 a podle této paměti je tekutina z y směrována buďto do výstupu $\vee$ nebo $\odot$.



Obrázek 2.8: Speciální ventil jako příklad mechanické paměti, zdroj [48]

Základní dělení kombinačních obvodů je podle hodinového vstupu. *Synchronní* obvody jsou synchronizovány hodinovým signálem. Stav synchronních obvodů se mění pouze jako reakce na hodinový signál (často na náběžnou hranu nebo aktivní stav, méně často na sestupnou (úběžnou) hranu nebo změnu logického signálu). *Asynchronní* obvody mění svůj stav okamžitě[1] při změně vstupních signálů, nepotřebují hodinový vstup.

Zaměřím se především na asynchronní klopné obvody, protože ve výsledné aplikaci není plánována podpora synchronních variant.

Schémata v této podkapitole jsou především ilustrativního charakteru. U většiny platí, že vstupní pulzy musí trvat dostatečně krátkou dobu, jinak nebude dosaženo předpokládaného chování. V praxi se užívají mnohem sofistikovanější obvody, často vícečinné.

2.5.2 Klopný obvod RS

RS obvod je nejjednodušší základní obvod, který má schopnost si udržet určitý stav bez přivedení vnějšího signálu. Jméno RS (také R-S) je zkratka jeho dvou vstupů – **r**eset a **s**et. Obvod je tvořen dvěma NOR hradly (členy), která jsou křížem propojena vstupy a výstupy. V tabulce 2.8 vidíme pravdivostní hodnoty obvodu a na obrázku 2.9 realizaci v asynchronní variantě.

R	S	Q_i	$\overline{Q_i}$
0	1	1	0
1	0	0	1
0	0	Q_{i-1}	$\overline{Q_{i-1}}$
1	1	zakázaný stav	

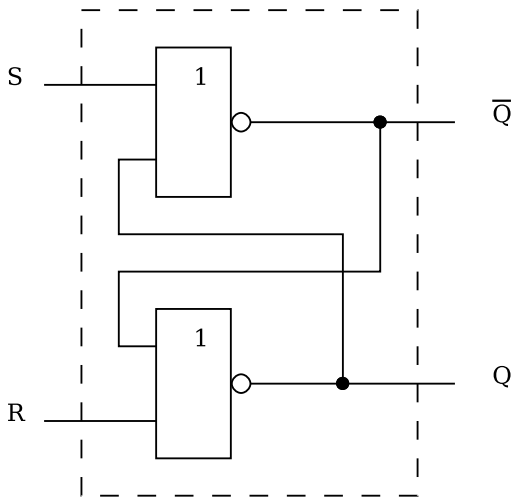
Tabulka 2.8: Pravdivostní tabulka RS obvodu

Neformálně lze obvod popsat tak, že při aktivaci vstupu S se obvod přepne do $Q = 1$ a při aktivaci vstupu R se navrátí do původního stavu $Q = 0$. Klidový stav, tj. vstupy $(S, R) = (0, 0)$, vrací uloženou hodnotu Q. Stav, kdy jsou oba vstupy na logické 1 není povolen.

2.5.3 Klopný obvod JK

Po prostudování tabulky 2.9 je patrné, že JK obvod funguje skoro stejně jako RS. Jediný rozdíl je, že nemá zakázanou kombinaci vstupních hodnot 11. Na tento vstup reaguje překlopením (toggle) uložené hodnoty. Pro označení JK existuje v angličtině mnemotechnická pomůcka – „**j**ump-**k**ill“.

[1] Pro jednoduchost zanedbáváme zpoždění prvku.



Obrázek 2.9: Asynchronní RS obvod

J	K	Q_i	$\overline{Q_i}$
0	1	0	1
1	0	1	0
0	0	Q_{i-1}	$\overline{Q_{i-1}}$
1	1	$\overline{Q_{i-1}}$	Q_{i-1}

Tabulka 2.9: Pravdivostní tabulka JK obvodu

Jak vidíme na obrázku 2.10, JK obvod může být realizován z RS obvodu a dvou AND členů. Synchronní variantu získáme jednoduše, stačí jen vyměnit asynchronní RS za synchronní a připojit hodinový signál.

2.5.4 Klopný obvod T

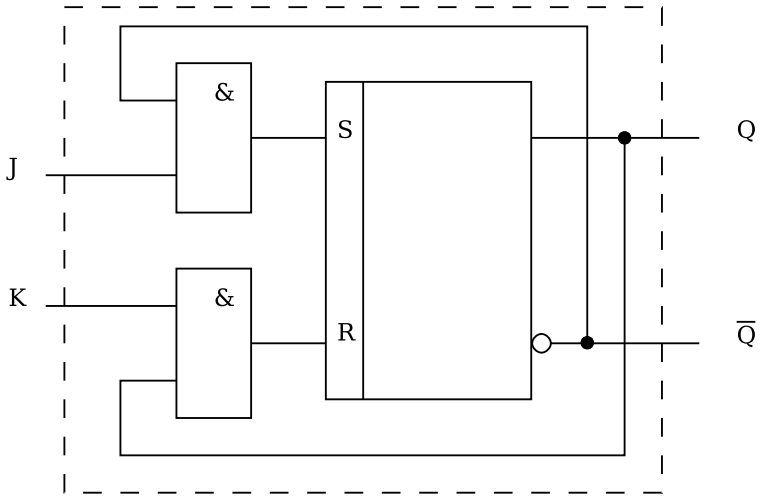
Obvod T (z anglického **t**oggle), někdy označovaný jako „přepínač paměti“, mění svůj výstup s každou změnou vstupního signálu (obvykle náběžná hrana). Tabulka 2.10 zachycuje chování obvodu.

T	Q_i
⌟‾	$\overline{Q_{i-1}}$
—	Q_{i-1}

Tabulka 2.10: Pravdivostní tabulka obvodu T

Asynchronní klopný obvod T lze konstruovat z asynchronního obvodu JK tím, že vstup T napojíme na oba vstupy – J i K. Tím docílíme požadovaného přepínacího chování.

„Překlápěcí“ chování obvodu T se využívá především v čítačích, které jsou tvořeny z kaskády složené z klopných obvodů typu T.



Obrázek 2.10: JK obvod

2.5.5 Klopný obvod D

Je běžně označovaný také jako „**d**ata“ nebo „**d**elay“ klopný obvod. Tento obvod reaguje nejčastěji na náběžnou hranu hodinového signálu na povolovacím vstupu C tak, že si uloží vstup D a dokud není signalizována změna (např. náběžná hrana na povolovacím vstupu), tak vysílá uloženou hodnotu jako výstup Q. Chování je vypsáno v tabulce 2.11.

D	C	Q_i
0	⌟⌜	0
1	⌟⌜	1
X	—	Q_{i-1}

Tabulka 2.11: Pravdivostní tabulka obvodu D, „X“ značí libovolnou hodnotu

2.5.6 Zdroje

Při psaní kapitoly bylo čerpáno z „Logic Circuit Design“ [48] a „Logické systémy“ [16].

2.6 Simulace

Tato část se zaměří na simulaci logických systémů. Text vychází z „Modeling and simulation of discrete-event systems“ [8], „Simulation“ [39] a [36].

Simulace je definována jako „technika napodobování chování určité situace pomocí podobné situace nebo zařízení s cílem získání dalších informací nebo jako prostředek školení (či pobavení)“. V definici „určitá situace“ koresponduje s logickým obvodem a „zařízení“ se simulačním programem. Pokud je cílem simulace získání nových informací, pak tuto činnost běžně označujeme jako *analytickou simulaci*. Druhý případ, ve kterém simulace slouží ke školení (nebo pobavení), se jmenuje *simulace virtuálního prostředí*.

Analytická simulace si klade za cíl poskytnout kvantitativní analýzu původního systému založenou na „přesných“ datech. Simulace v tomto případě je vykonávána co nejrychleji a tak,

aby zachytila co nejpřesněji sled událostí nastávající v původním systému.

Simulace virtuálního prostředí je vykonávána v reálném čase, tj. čase který odpovídá časovým úsekům původního (simulovaného) systému. Příkladem mohou být počítačové hry, např. simulace hokejového zápasu.

2.6.1 Důvody simulace

K simulaci se uchylujeme, protože z praktických důvodů není možné provádět pokusy přímo v reálném světě. Příčinou může být rozpočet, náročnost výroby testovaného objektu nebo objektů nutných k testu, případně riziko zničení unikátního testovaného objektu.

2.6.2 Typy simulací

Simulace systému se spojitým časem se nazývá *spojitá simulace*. Její provádění je v podstatě numerické vyčíslování modelu dynamického fyzikálního systému vyjádřeného množinou rovnic, typicky diferenciálních.

Simulace systému s diskrétními událostmi se nazývá *diskrétní simulace*. Je to počítačové vyčíslování modelu dynamického systému, kde fungování systému je reprezentováno jako chronologický sled událostí.

Dalším typem simulace je populární *Monte Carlo*, které ale neslouží k simulaci dynamických systémů. Tato třída algoritmů spoléhá na opakované náhodné vzorkování. Většinou se využívá k numerické integraci funkcí, které jsou neřešitelné analyticky.

2.6.3 Metoda provádění diskrétní simulace

Systémy s diskrétními událostmi (anglicky discrete-event system – DES) jsou systémy s diskrétními stavy a změnami pouze v časech událostí.

Za účelem simulace vnitřních stavů systému a jejich změn je potřeba mechanismus pro zpracovávání *budoucích událostí*. Budoucí událost znamená, že je její výskyt naplánován v nějakém bodě času v budoucnosti. Událost, která má nejnižší aktivační čas se nazývá *next event* (z angl. další událost). Mechanismus zpracovávající události sestává z tzv. *future event list* (FEL, také označovaná jako kalendář událostí) a záznamů událostí, které nesou jméno (obecněji např. ukazatel na objekt typu událost) a aktivační čas. Tyto karty jsou uspořádány ve FEL vzestupně podle aktivačního času.

Zpracování next event (také *algoritmus next event*) začíná vybráním next event, tj. akce s nejnižším aktivačním časem. Simulační čas se nastaví na aktivační čas právě vybrané události a následuje její spuštění. Během provádění akce může dojít ke změně stavových proměnných nebo tabulky událostí (FEL). Provádění se opakuje, končit může buďto dosažením koncového času, vyprázdněním tabulky událostí nebo dosažením určité koncové podmínky.

2.6.4 Simulace logického systému

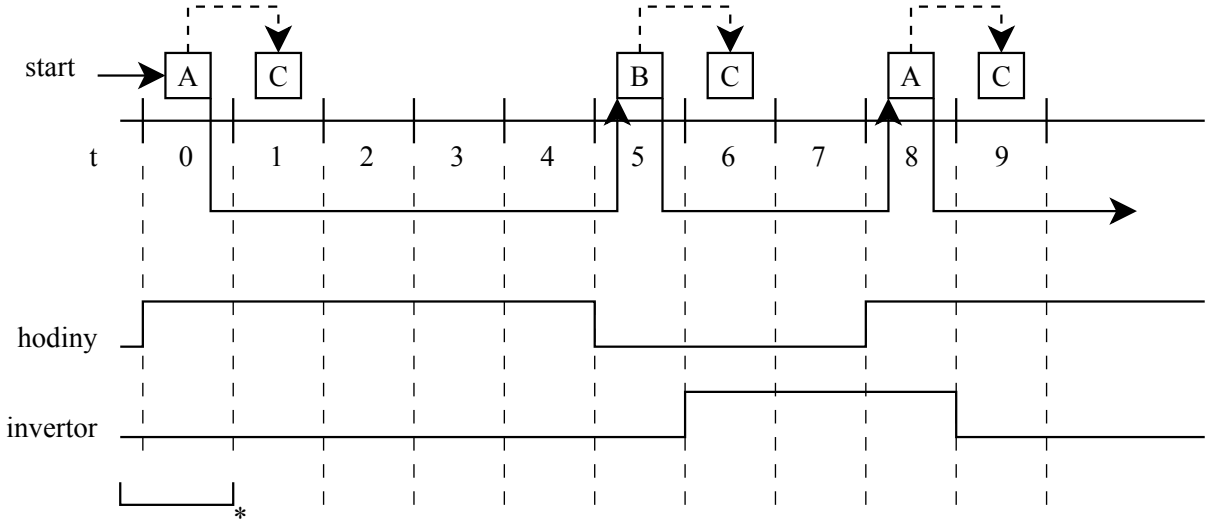
Jde o diskrétní simulaci, kde události jsou změny výstupů členů nebo obvodů. Na takovéto události jsou napojeny vstupy prvků, které mohou naplánovat změnu svých výstupů.

Mějme například generátor hodinového signálu. Před startem simulace si naplánuje, že v čase 0 nastane událost A, což je nastavení výstupu na úroveň 1 a naplánovaní události B v čase $t+5$. Stejně jako událost A, událost B nastaví výstup, tentokrát na 0, a naplánuje A na $t+3$.

Tento příklad není moc oslňující, ale mějme na paměti, že na změny výstupu (událostmi A a B) mohou reagovat další členy. Ku příkladu invertor I připojený k hodinám bude simulován

tak, že se invertor zaregistruje u výstupního signálu hodin. Díky tomu může při každé změně hodinového signálu přepočítat svůj výstup, tj. znegovat výstup z hodin. Pokud bychom chtěli simulovat zpoždění, pak by invertor naplánoval událost C například na $t + 1$, která by provedla změnu výstupu.

Kompletní ilustraci příkladu vidíme na obrázku 2.11. Oblast označená hvězdičkou symbolizuje nedefinovaný výstup invertoru. Nedefinovaná oblast je způsobená tím, že hodiny nastavují svůj výstup v čase 0, tudíž až v čase 1 může invertor přepočíst svůj výstup.



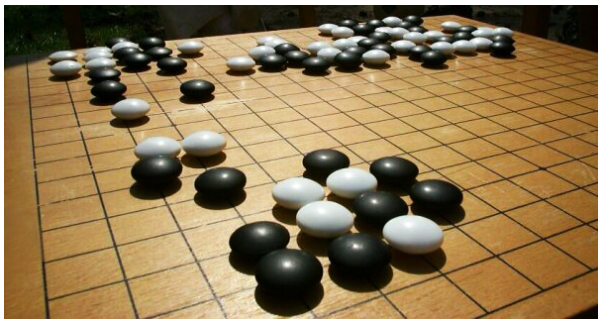
Obrázek 2.11: Příklad simulace logického systému

Kapitola 3

Počítačové hry a videohry

3.1 Hry

Člověk vymýšlel a hrál hry od nepaměti, mezi jedny z nejstarších patří čínské Go a africké Awari. Původně samozřejmě nešlo o hry provozované na počítači nebo konzoli, ale typicky o hry deskové. Koncept hry je založen na předstírání, schopnosti vžít se do herního světa, představovat si např. postavy, hrdiny, bojiště, vesmírné lodě, odlišné formy života atp.



Obrázek 3.1: Hra Go
autor fotografie: Donarreiskoffer

Obrázek 3.2: Hra Awari
autor fotografie: Wikiwizzy

Pod hrou rozumíme zábavnou činnost, často interaktivní, která vyžaduje aktivní účast. Liší se tak od jiných zdrojů zábavy, jakými jsou sledování televizního pořadu, filmu, čtení knihy nebo návštěvy kina či divadla, které jsou *pasivní* formy zábavy. To znamená, že se od člověka neočekává, že se bude jakkoliv podílet na této činnosti, že např. bude sdělovat hercům na jevišti, co jejich postava udělá v další scéně.

Naopak u her je vyžadována aktivita, oproti pasivním formám zábavy jsou pro člověka mnohem složitější. Přestože svět je virtuální, tak hráč není všemohoucí, je svázán herními pravidly. Tato pravidla přesně definují, které akce nebo tahy může nebo nemůže hráč provést. V počítačových hrách jsou pravidla často skryta, částečně proto, že při nevalidním vstupu hra hráče jednoduše ignoruje. Pokud lze danou akci ve hře provést, hráč očekává, že je tato akce povolena.

Mezi další formy zábavy patří hlavolamy (skládačky, hádanky, rébusy). Na rozdíl od her tu často chybí vystupování v roli hrdiny nebo umístění do virtuálního světa. Hlavolamy, označované také jako puzzle hry, mají hlavní pravidlo: existuje správné řešení, které se hráč snaží nalézt. Typicky jsou hrány v jednom člověku (nemusí platit vždy, bude zmíněno v další podkapitole) a vyžadují zvažování důsledků tahů.

3.2 Počítačové hry

Přestože se někdy počítačové hry a videohry rozdělují, v této práci budu rozumět pod počítačovou hrou i hru konzolovou, hru na mobilní telefon nebo tablet. Přece jen konzole je pouze speciální forma počítače. A i tento fakt přestává platit. Příkladem budiž firma Valve, která

plánuje „konzoli“ Steam Machine (Steam Box) s operačním systémem SteamOS založeným na Linuxu a hardwarem stejným, jako mají nynější PC. Operační systém bude dostupný pro kohokoliv zdarma a půjde jej nainstalovat na jakýkoliv počítač.

Typy, neboli žánry, počítačových her se poměrně dost odlišují od typů her nepočítačových. Na počítači totiž lze mimo jiné zajistit dodržování pravidel, časování a správné vyhodnocení výhry i přesto, že hráč neví či nechápe všechna pravidla hry.

3.2.1 Akční hry

Standardně obsahují výzvy fyzikální, jednodušší hádanky, závodění, širokou škálu úkolů točících se okolo konfliktů, které jsou často osobního rázu. Mohou obsahovat jednoduché ekonomické úkoly, většinou jde o sbíraní určitých objektů. Akční hry málokdy obsahují strategické nebo abstraktní úlohy. Mohou být 3D nebo 2D.

Existuje velké množství podtypů akčních her, jde o nejstarší herní žánr. Všechny mají ale jedno společné – nejdůležitější dovednost je reakční doba a koordinace ruky a oka. Akční hry bývají často jednodušší, protože si žádají rychlé reakce – složitější herní mechaniky by po hráči vyžadovaly nadlidské výkony.

3.2.2 Strategie

Strategické hry zpravidla obsahují strategické, taktické a logistické výzvy, někdy také doplněné o úkoly ekonomického zaměření.

3.2.3 RPG

„Role-playing games“ neboli „hry na hrdiny“ se soustředí na úlohy taktické, logistické a průzkum terénu. Také obsahují výzvy ekonomické, protože tento typ her obyčejně zahrnuje získávání kořisti, která může být směněna za lepší zbraně, zbroj a vybavení. Někdy obsahují hádanky a výzvy abstraktnější, málokdy pak fyzické. Velmi důležité aspekty, které sdílí snad všechny RPG hry, jsou silný příběh a možnost výběru směru rozvoje hráčem ovládaných postav. Tyto postavy se postupně zlepšují s nabytými zkušenostmi.

3.2.4 Puzzle hry

Úkoly v tomto žánru bývají výhradně logické, občas umocněné časovým omezením nebo obohacené o akční prvky. Puzzle hry jsou primárně o řešení hlavolamů (puzzle), někdy neobsahují ani příběh nebo vyšší cíl. Neznamená to však, že puzzle hra je pouze nahodilá směsice hlavolamů, obyčejně jde o variace na jedno téma. Více do hloubky jsou puzzle hry popsány v části 3.3.

3.2.5 Simulace

Tento žánr pokrývá sportovní hry a simulace vozidel. Zahrnují především výzvy fyzické a taktické, nikoliv pak průzkumné, ekonomické nebo konceptuální.

3.2.6 Hry soustředící se na stavbu a řízení

Jsou především o úkolech ekonomických a konceptuálních (tj. takových úkolech, ve kterých se hráč musí naučit či pochopit něco nového, aby je mohl splnit). Pouze velmi výjimečně obsahují konflikt nebo průzkum terénu a téměř nikdy nezahrnují fyzické výzvy.

3.2.7 Adventury

Název pochází z anglického „adventure“, což znamená dobrodružství. Jejich primární zaměření jsou průzkum a řešení hádanek. Občas obsahují i konceptuální úlohy, zřídkakdy pak jakékoliv fyzické úkoly.

3.3 Puzzle hry

Puzzle hry často obsahují tvary, barvy nebo symboly, které musí být uspořádány tak, aby vytvářely určitý vzorek. Někdy jsou tyto objekty napevno dané, jindy padají z nebe. V některých případech dokonce hráč neovládá objekty a postavy přímo, ale pouze nepřímo změnami v herním prostředí. Některé puzzle hry spoléhají na rychlé reflexy, zatímco jiné odměňují uvažování. Věc, co mají všechny tyto hry společné, je důraz na strategii, logické myšlení a důmyslnost.

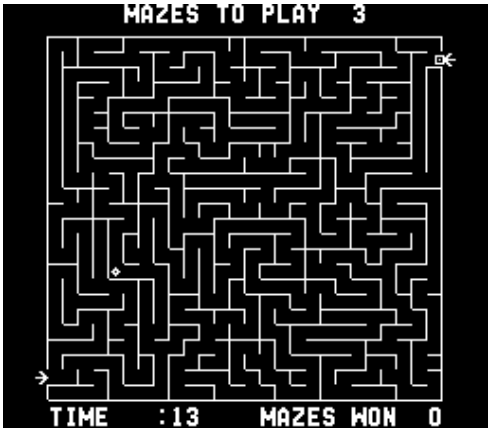
Podle „The History of Puzzle Games“ [18] byla první hra tohoto žánru legendární Tetris, který zná snad doslova každý.

I v období před vydáním hry Tetris obsahovaly hry některé prvky, které se následně staly nedílnou součástí puzzle her. Přestože hry tohoto období byly z velké části postaveny kolem zjednodušených cílů jako střílení mimozemšťanů, asteroidů nebo robotů, nemalá skupina těchto her si vyžadovala trochu více přemýšlení.

3.3.1 Období před vydáním hry Tetris

Amazing Maze

Mezi nejstarší „protopuzzle“ hry se řadí „Amazing Maze“ z roku 1976. Na první pohled připomíná více hru Pac-Man než Tetris. Když se ale podíváme blíže na herní mechaniky, tak zjistíme, že hra neobsahuje žádné nepřátele nebo zbraně a úkolem je dosáhnout cíle dříve, než druhý hráč (ať už lidský nebo řízený počítačem). Nejsou zde žádní duchové, vetřelci, asteroidy nebo vesmírné lodě, v závěru jde jenom o to, jak si váš mozek poradí s labyrintem.



Obrázek 3.3: Amazing Maze
autor obrázku: McLoaf

Obrázek 3.4: Basic Math
zdroj: atarimania.com

První výukové hry

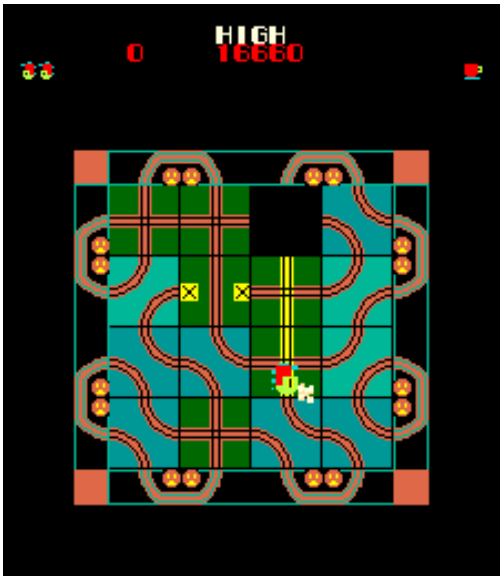
Vznikaly také výukové hry. Výrobci konzolí se snažili vytvořit dojem, že konzole slouží k výuce i k zábavě. Přestože tyto „hry“ byly často nezábavné (např. „Basic Math“) a většinou toho ani moc nenaučily, tak mají jeden důležitý rys viděný u puzzle her. Prosazují přemýšlení a strategii před akcí a dobrodružstvím. Také ukázali hráčům, že se hry nemusí odehrávat jen ve vesmíru nebo na bitevním poli.

Loco-Motion

Za zmínku stojí hra „Loco-Motion“ vydaná roku 1982. Motivem hry je vlak vyzvedávající pasažéry. Hráč ale nemá přímou kontrolu nad vlakem, tento element je typický pro puzzle hry. Cílem je postavit železniční trať ze čtverců s kolejemi (většinou zatáčky) tak, aby vlak dojel do cíle.

Q*Bert

Hra „Q*Bert“ (1982), přestože její koncept zní poměrně nesmyslně, byla velmi oblíbená. Hlavní protagonista je oranžový tvor s dlouhým nosem. Tato hráčem ovládaná postava skáče po pyramidě z barevných kostek a je pronásledována barevnými koulemi a fialovým hadem se jménem Coily. Pokaždé, když Q*Bert přistane na kostce, kostka změní barvu. Cílem každé úrovně je změnit barvu celé pyramidy. Množství nepřátel dělá ze hry Q*Bert křížence mezi akční a puzzle hrou, přesto tato hra přinesla prvek puzzle her – barvení. Princip barvení, nebo spíše snaha o dosažení stejných barev, je v puzzle žánru velmi používaný a stále aktuální.



Obrázek 3.5: Loco-Motion
autor obrázku: Lyverbe

Obrázek 3.6: Q*Bert
autor obrázku: McLoaf

Q*Bert's Qubes

„Q*Bert's Qubes“ vydaná v roce 1983 a 1984 pokračuje v užívání principu shody podle barvy. Velkou změnou je ale herní plocha – už nejde o pseudo-3D pyramidu, ale o mřížku 5×5 obsahující 3D kostky. Po každém skoku z kostky se kostka převrátí podle směru skoku hrdiny. Cílem je

dostat stejně natočené kostky v řadě o délce 5 (v libovolném směru) tak, aby konfigurace kostek odpovídala kostce zobrazené na obrazovce v levém horním rohu. Čím více shodných kostek s kostkou v levém horním rohu, tím více bodů na konci úrovně hráč dostane. Coily je vystřídán fialovou krysou, oproti předchozímu dílu ubyla většina nepřátel. Akční část – utíkání a uhýbání nepřátelům – v pokračování není příliš náročná, hráč se tedy může plně soustředit na puzzle část hry.

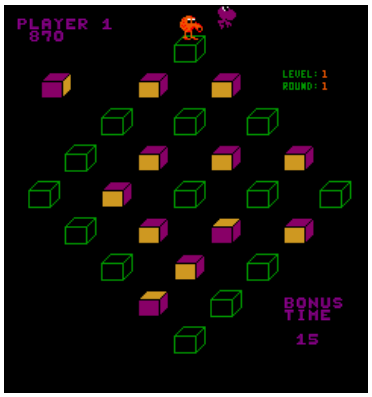
Hra nebyla příliš úspěšná, pravděpodobně proto, že její vydání korespondovalo s propadem video her v polovině osmdesátých let. Jde pravděpodobně o první opravdu návykovou puzzle hru.

Atari Video Cube

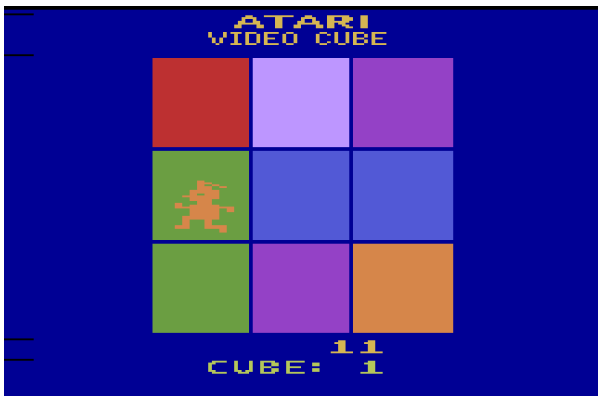
Lehčí verze Rubikovy kostky, tak by šla popsat hra „Atari Video Cube“ (1982). Na obrazovce je vždy vidět pouze jedna strana kostky, každá strana se skládá z devíti menších barevných čtverců, podobně jako Rubikova kostka. Každý z menších čtverců je vybarvený jednou z následující barev: červená, oranžová, zelená, modrá, fialová nebo bílá.

Cílem je sladit kostku tak, aby každá strana obsahovala pouze jednu barvu. Hráč ovládá postavu se jménem Hubie, který může sebrat barvu a vyměnit ji s barvou na jiném čtverci. Jediné omezující pravidlo je, že hráč nemůže chodit s postavou po čtvercích, které mají stejnou barvu jakou nese postava.

Hra „Atari Video Cube“ působí jako první opravdová puzzle hra – vyžaduje přemýšlet nad nelehkým problémem, požaduje nalezení strategie, zahrnuje barvy a tvary a hráč zápasí se svou myslí namísto nepřátel na obrazovce. Hráč je nucen přemýšlet ve 3D, přestože samotná hra je 2D, dále je potřeba si pamatovat, co je na ostatních stranách kostky. Pokud by byla zobrazena celá kostka, pak by se ztratilo ono puzzle kouzlo. Nebylo by třeba si nic pamatovat a popravdě ani moc přemýšlet – a přemýšlení je to, o čem celá tato hra je.



Obrázek 3.7: Q*Bert's Qubes
zdroj: arcade-museum.com



Obrázek 3.8: Atari Video Cube
zdroj: insert-disk.com

3.3.2 Tetris

V historii video her je jen pár titulů, které mohou být označeny jako revoluční. Mezi ně patří, kromě jiných, „Pac-Man“, „Super Mario Bros.“, „Street Fighter II“ a také „Tetris“. Základní principy hry „Tetris“ už byly použity dříve, ještě před vydáním v roce 1988. Přispěly k nim např. už zmíněné „Q*Bert's Qubes“ a „Atari Video Cube“. Přesto žádná puzzle hra nebyla tak

jednoduchá, elegantní a zároveň nezměrně oblíbená jako „Tetris“. Hra byla vytvořena rusem Alexejem Pajitnovem v roce 1985, poprvé se objevila v Americe roku 1987 v počítačové edici a na herních automatech o rok později.

Ale až roku 1989, když Nintendo představilo „Game Boy“, propukla největší vlna hraní „Tetrisu“. Hra byla totiž přibalována při koupi herního zařízení, navíc se skvěle hrála i na černobílé obrazovce. Byla jednoduchá a návyková.

„Tetris“ vycházel z tradiční hry „Pentomino“, kde se objekty otáčejí a skládají tak, aby dokonale (bez mezer) zaplnily řádky. Jméno „Tetris“ pochází z řeckého slova „tetra“ – čtyři. Každý herní dílek se skládá právě ze čtyř čtverců.

Zajímavá je otázка, zda se „Tetris“ proslavil díky hernímu systému „Game Boy“, nebo naopak. Každopádně, kromě nejpopulárnější puzzle hry na světe má „Tetris“ asi i další prvenství – hra s nejvíce pokračováními, variacemi, klony a napodobeninami. První verzi a původní Game Boy vydání si můžete prohlédnout na obrázcích 3.9 a 3.10.

Obrázek 3.9: Tetris – první verze, 1984
autor obr.: Jarkka Saariluoma

Obrázek 3.10: Tetris na Game Boy
autor obr.: CountingPine

Po masivním úspěchu „Tetrisu“ nastává velký příliv puzzle her, některé zajímavé počiny představím v další části.

3.3.3 Doba po hře Tetris

Block Out

„Block Out“ (1989) je v podstatě „Tetris“ ve 3D viděný ze shora. Nápad je to dobrý, bohužel provedení není příliš domyšlené. Hra trpí mnoha neduhy, ku příkladu je problém s orientací. Hráč moc dobře nevidí aktuální padající dílek a ani již položené kostky nejsou moc rozpoznatelné, je totiž k dispozici jen pohled připomínající studnu viděnou shora. Aktuální dílek lze otáčet, bohužel je vidět pouze drátový model dílku, který je většinou hodně matoucí. Zajímavý koncept zmařený nedotaženou realizací.

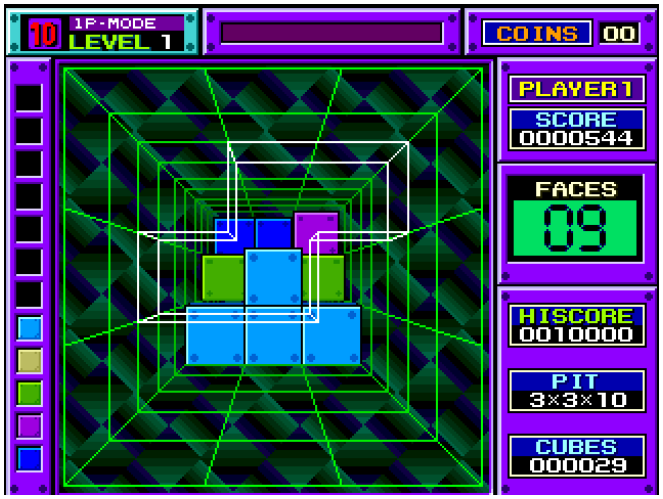
Boxxle

Remake známé hry „Sokoban“ (1982) na Game Boy, „Boxxle“ vyšel v roce 1989. V pohledu ze shora musí hráč ovládající skladníka natlačit bedny na vyznačená místa. Háček je v tom, že bedny nelze táhnout, pouze tlačit dopředu.

Klax

Ve hře „Klax“ (1989) pás posouvá dolů barevné dlaždice. Úkolem hráče je přeskládat je do tzv. „klaxů“, což jsou řady tří nebo více dlaždic stejné barvy. Hráč může mít na lopatce až pět dlaždic.

Každý vytvořený klax zmizí (podobně jako řada v „Tetrisu“). Hra je rozdělena do vln, každá vlna má svoje podmínky pro dokončení – např. tři klaxy, 10 000 bodů nebo přežít 50 dlaždic.



Obrázek 3.11: Block Out
zdroj: uvlist.net

Obrázek 3.12: Klax
zdroj: game-oldies.com

Na první pohled vypadá „Klax“ jednoduše, ale čím déle hrajete, tím více mechanik objevujete – např. výhody různých barev na lopatce nebo dosažení více klaxů jedním položením. Velmi rychle hra dojde do stavu, kdy dlaždice sviští na pásu tak rychle, že nemáte skoro čas přemýšlet. Tato kombinace puzzle a akční hry byla hráči shledávána velmi zábavnou.

Pipe Dream

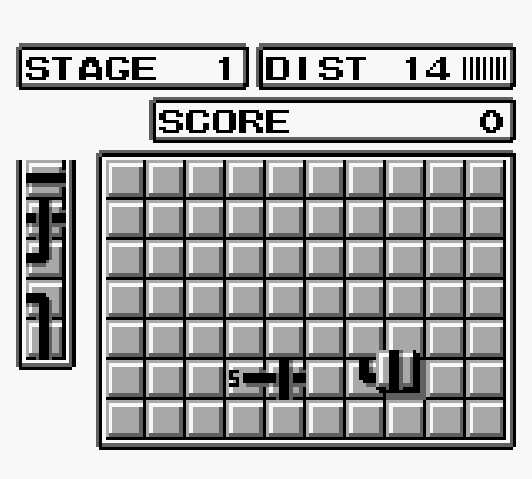
Skládání různých dílků potrubí do fungujícího celku o zadané délce – tak by šla popsat hra „Pipe Dream“ vydaná roku 1989 na Game Boy. Přestože hra není příliš chytlavá – je repetitivní (úrovně se moc neliší), tak přinesla inovativní myšlenku do světa puzzle her.

Columns

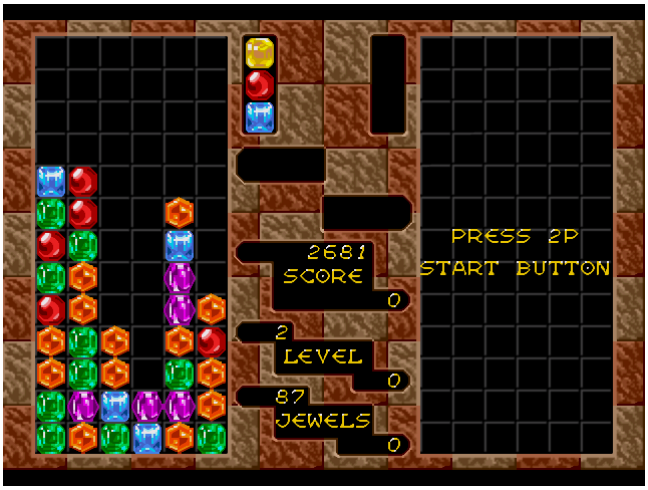
Vynikající puzzle hra na konzoli Sega, která vznikla jako reakce na úspěch hry „Tetris“. „Columns“ má jednoduchá pravidla i myšlenku a je extrémně návyková.

Z nebe padají barevné drahokamy spojené vždy po třech ve vertikálním směru. Tyto dílky nelze otáčet, ale je možné měnit pořadí různobarevných drahokamů. Úkolem je vytvářet řady tří nebo více stejně barevných drahokamů. Řada může být v libovolném směru – tj. horizontální, vertikální, nebo diagonální. Po dokončení řady řada zmizí a zbylé drahokamy se propadnou do volných míst. Nastavení obtížnosti určuje, z kolika barev se vybírají generované drahokamy. Stejně jako ve hře „Tetris“, pokud drahokamy přesáhnou horní okraj obrazovky, hra skončila.

Asi nejvíce uspokojivá část této hry je vytváření velkých řetězových reakcí spuštěných jediným drahokamem. Hra je propracovaná, obsahuje i několik herních módů. Hudba a vizuální stránka jsou vypilované a působí stylově (na rozdíl od nemalé části ostatních puzzle her).



Obrázek 3.13: Pipe Dream
zdroj: emuparadise.me



Obrázek 3.14: Columns
zdroj: gamefabrique.com

Přestože za vznik hra zcela jistě vděčí hře „Tetris“, „Columns“ je originální počin – využití barev, řetězových reakcí a módu, ve kterém hráč musí co nejrychleji zničit drahokam ležící na dně obrazovky. To vše dělá z této hry klasiku ve svém žánru.

Ishido: The Way of Stones

Cílem hry „Ishido“ je zbavit se všech kamenů na hrací ploše. Každý kámen je definovaný svou barvou a symbolem. Kameny lze položit pouze na sousední pole s kamenem, který má stejnou barvu nebo symbol. Vícenásobná shoda dvou nebo více kamenů musí splňovat podmínku, že i barva i symbol jsou stejné.

„Ishido“ neobsahuje žádné časové omezení a odměňuje náročné uvažování.

Obrázek 3.15: Ishido: The Way of Stones
zdroj: gamesdbase.com

Obrázek 3.16: Lemmings
zdroj: dr-hoiby.com

Lemmings

V reálném světě jsou lumíci malí hlodavci, kteří slepě následují druhé lumíky, což často vede k jejich smrti. V této hře se chovají velmi podobně, ale je na hráči, aby je zachránil před vyhynutím.

Každá úroveň hry má přesně daný počet lumíků, kteří propadnou do úrovně. Lumíci pak bezmyšlenkovitě jdou jedním směrem, úplně slepí vůči faktu, že za okamžik se utopí, spadnou nebo uhoří. Hráč musí takticky probouzet vrozené schopnosti vhodných lumíků tak, aby jich co nejvíce zachránil. Například pro zachránění skupiny lumíků před utopením může hráč jednoho (toho nejblíže vodě) instruovat, aby na sebe vzal roli nazvanou „blocker“, tj. nepustil ostatní lumíky dál. Ti se otočí a pokračují na druhou stranu, hrozba utopení je na chvíli zažehnána.

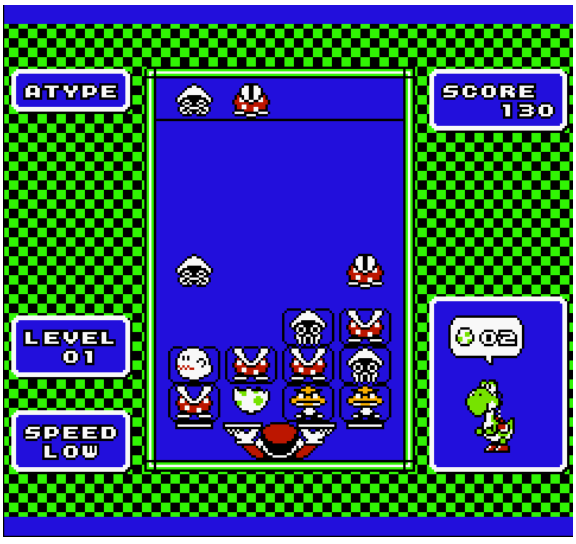
Cílem je zachránit určitý počet lumíků tím, že je hráč odvede do konce úrovně. Počet probuzení schopností v lumících je ale omezen, je přesně stanoven podle úrovně. Jak je běžné pro spoustu puzzle her, první úrovně jsou velmi jednoduché, zaučují hráče v používání různých schopností lumíků. Pozdější úrovně jsou ale mnohem obtížnější, mohou zabrat hodiny pokusů a občas je nutné mít i trochu štěstí pro úspěšné dokončení úrovně.

Hra „Lemmings“ je místy velmi náročná, ale díky jejímu vynalézavému hernímu principu, roztomilé grafice a návykovosti ji hráči hodnotili jako extrémně zábavnou.

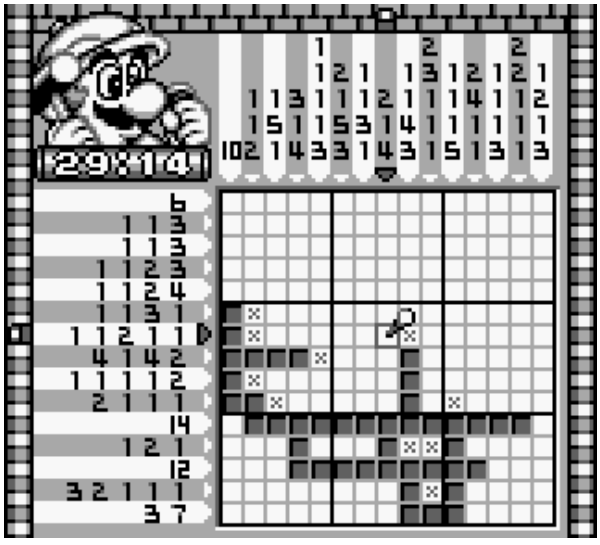
Mario & Yoshi

Hra „Mario & Yoshi“, také známá jako „Yoshi“ nebo „Yoshi's Egg“, byla uvedena na trh v roce 1991. Mario – světoznámá postava instalatéra – je v této hře doprovázen jeho přítelem/mazlíčkem Yoshim (zelený dinosaurus).

Z oblohy padají čtyři typy objektů – příšery Goomba (houby známé z jiných her s Mariem), duchové, chobotnice a skořápky. Nastavení obtížnosti určuje, po kolika tyto objekty padají (dva až čtyři zároveň). Herní objekty nelze přesouvat přímo, ale hráč může za pomoci postavy Mario vyměňovat podnosy, na kterých leží tyto objekty. Úkolem je dostat dva objekty stejného typu ve vertikálním směru.



Obrázek 3.17: Mario & Yoshi
zdroj: dailymotion.com



Obrázek 3.18: Mario's Picross
zdroj: mariowiki.com

Hezké zpestření jsou skořápky, existují dvě varianty – horní a dolní. Pokud se hráči podaří

je seskládat ve správném pořadí, tak se vylíhne Yoshi a pomůže vyčistit část obrazovky.

Mario's Picross

Hra „Mario's Picross" vydaná 1995 na Game Boy je mix křížovky, hry čísel a hledání min. Hraje se na matici různých velikostí – od 5×5 do 15×15. Cílem je odkrýt obrázek vyplňováním čtverců do matice. Na x-ové a y-ové ose u každého řádku a sloupce je číslo nebo sekvence čísel, která odpovídá počtu vyplněných čtverců v daném řádku či sloupci. Každá úroveň je omezena na 30 minut, za každé špatně vyplněné políčko ztrácí hráč část zbývajícího času.

„Mario's Picross" je perfektní puzzle hra – jednoduchá na naučení, návyková, povzbuzující přemýšlení, zábavná a obsahuje velké množství úrovní – přes 190. Je to ta pravá hra pro hloubavého člověka.

Bust-A-Move

„Bust-A-Move" (1994), také označovaná jako „Puzzle Bobble", je akční puzzle hra. Nahoře na obrazovce je v každé úrovni určité uskupení barevných bublin. Hráč ovládá šipku ve spodní části obrazovky, která je obsluhována dvěma dinosaury a vystřeluje barevné bubliny. Úkolem hráče je tvořit řetězy ze stejně obarvených bublin, pro úspěšné dokončení úrovně je třeba vyčistit všechny bubliny. Při vytvoření skupiny tří nebo více bublin stejné barvy bubliny prasknou a hráč dostane body. Vždy po určitém časovém úseku se strop pod tíhou bublin posune dolů, pokud bubliny dosáhnou hráčem ovládané šipky, hráč prohrává.

Obrázek 3.19: Bust-A-Move, zdroj: [30]

Hra má roztomilé grafické zpracování, je opravdu chytlavá a velmi zábavná. Jde snad o nejúspěšnější puzzle hru po „Tetrisu".

Portal a Portal 2

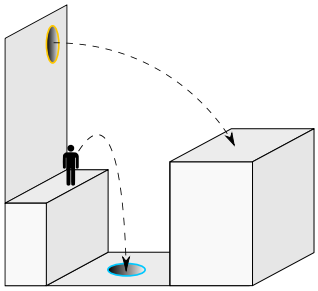
Ve 3D hře „Portal" (2007) dostává hráč jediný nástroj, který bude používat po celou hru – tzv. „portal gun". Kamera je z první osoby, hra se ovládá jako klasické FPS[1]. Přestože je to „gun", tak nejde o zbraň, ale opravdu o nástroj. „Portal gun" slouží k vytváření dvou portálů – modrého a oranžového. Hráč a herní objekty mohou vstoupit do jednoho portálu a vystoupit druhým, přičemž si zachovají svou hybnost. Pouze dva portály mohou být otevřeny v jeden okamžik.

[1] First Person Shooter – „střílečka" z první osoby.

Portály mohou být vytvořeny na bílých hladkých stěnách, které se vyskytují většinou ve velkých plochách.

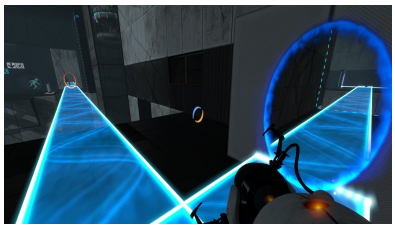
Sepsané na papíře to může vyznívat divně, možná složitě, ale herní mechanika je velmi jednoduchá – dva portály, hráč, obrovská tlačítka aktivovaná velkými kostkami a statičtí střílející roboti, kteří se musí překlopit.

I přes krátkou herní dobu a minimalistické ozvučení si „Portal“ získal zástupy fanoušků. Po dlouhém čase se puzzle hře podařilo proniknout mezi masy. V puzzle hrách se často setkáváme s žádným nebo zcela nevýrazným příběhem, to o „Portalu“ neplatí. Celou hrou hráče provází hlas zlého počítače zasypávající protagonistu vypilovanými monology (hlavní postava nemluví).



autoři: Dammit a Pbroks13

zdroj: games.cnews.ru



zdroj: zerot.wordpress.com

Obrázek 3.20: Ukázka použití portálu, hry Portal a Portal 2

„Portal 2“ (2011) klade větší důraz na příběh a herní dobu, přidává další mechaniky (především kapaliny, lasery a tzv. „Hard Light Bridge“, volně přeložené jako mosty z tvrdého světla), zvyšuje rozmanitost prostředí a jako jedna z mála puzzle her obsahuje i kooperativní mód dvou hráčů.

Za zmínku stojí studie pojednávající o vlivu her na kognitivní schopnosti. Ukazuje se, že při hraní „Portal 2“ dochází k většímu zlepšení těchto schopností, než při hraní hry speciálně vyvinuté pro tento účel – „Luminosity“ [40].

Některé informace o „Portal“ a „Portal 2“ jsem čerpal z [1] a [33].

3.3.4 Puzzle hry sloužící k výzkumu

Existuje několik netypických puzzle her, které využívají „výpočetní“ sílu mozků hráčů k řešení vědeckých problémů.

V několik let starém projektu „Foldit“ [15] se hráči snaží modelovat skládání proteinů. Velkým úspěchem bylo, když po pouhých třech týdnech[2] hráči rozluštili podstatu Mason-Pfizerova opičího viru, který způsobuje AIDS u opic. Problém, se kterým si vědci nevěděli rady dlouhých patnáct let [37].

Novějším počinem je „Nanocrafter“ [31], který se zabývá syntetickou biologií. Hráč manipuluje s části DNA, buduje rozličné funkční struktury (jako příklady aplikace poznatků jsou uváděny biologické obvody a nanoroboti).

3.4 Zdroje

Podklady pro tuto kapitolu byly čerpány z „Andrew Rollings and Ernest Adams on Game Design“ [38], „The History of Puzzle Games“ [18] a recenze [30].

[2] Tři týdny trvala akce, samotná odpověď byla nalezena po deseti dnech.

Kapitola 4

Multiplatformní aplikace

Před popisem platforem nejprve stručně nastíním jejich historii, pak se budu věnovat analýze současných platforem, mobilních i počítačových, a možnostem multiplatformního vývoje.

4.1 Mobilní platformy

4.1.1 Stručná historie mobilních platforem

V dobách „hloupých“ telefonů existovalo nemálo operačních systémů, většinou pocházejících přímo od výrobce daného zařízení – např. Symbian OS, Mobile Linux, Moblin, Windows Mobile [17]. Postupně mobilní zařízení nabírala další a další funkce, až do bodu, kdy psát speciální operační systém pro jednu řadu zařízení nebylo výhodné.

Také s rostoucím výkonem a s obrovskými ekosystémy[1] kolem největších platforem – nejdříve iOS od firmy Apple, následně Android od společnosti Google ale i Windows Phone (přestože stále silně minoritní) – se výrobci mobilních zařízení stále častěji uchylovali k používání produktů těchto gigantů namísto investic do vlastního operačního systému s nejistým osudem.

Výhody použití hotových operačních systémů jsou ku příkladu cena vývoje vlastního OS, nespočetně již existujících aplikací – každý OS má typicky svůj centralizovaný obchod či katalog s aplikacemi, zástupy vývojářů, již existující lidé věnující se zákaznické podpoře, uživateli otestované a přijaté GUI nebo fungující synchronizace.

Samozřejmě tu jsou i určité nevýhody, například zařízení musí splnit hardwarové požadavky OS, univerzálnější OS systémy jsou o něco náročnější na systémové prostředky než OS psaný přímo pro dané zařízení, neunikátní uživatelské rozhraní. Tyto nevýhody nejsou ale tak hrozné, většina z nich je řešitelná, nebo z větší části vyřešena. Například nejnovější Android není moc hardwarově náročný (na aktuálních zařízeních ve srovnaní se staršími verzemi), uživatelské rozhraní si výrobci často (k nelibosti pokročilejších uživatelů) modifikují. Přestože aplikace by mohly běžet rychleji na OS šitém na míru danému přístroji, výkon mobilních zařízení pokročil natolik, že se jednoduše nevyplatí drahé optimalizace a relativní pomalost aplikací se vyřeší hrubou silou – výkonnějším hardwarem.

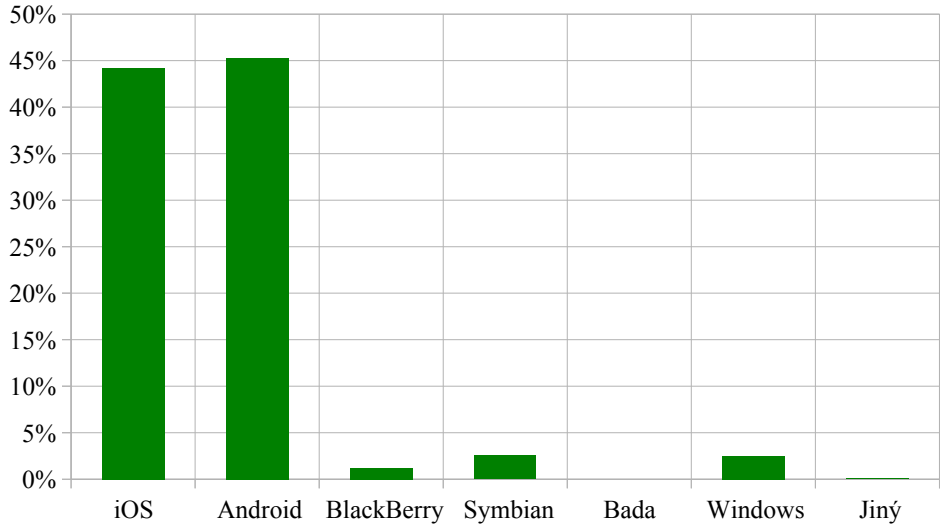
4.1.2 Současný stav na poli mobilních platforem

Na obrázku 4.1 vidíme, že nyní (rok 2014) iOS spolu s Androidem kralují mobilnímu trhu.

iOS

iOS je uzavřený mobilní operační systém, za kterým stojí společnost Apple. Vývoj pro iOS platformu probíhá tradičně v jazyce Objective C. Tento jazyk doprovází iOS a OS X od počátku, byl používán v operačním systému NeXTSTEP ze kterého pochází operační systémy firmy Apple [27].

[1] Ekosystémem se myslí vývojová prostředí, firmy a především aplikace pro daný operační systém/platformu.



Obrázek 4.1: Zastoupení mobilních OS v červenci 2014, zdroj dat: netapplications.com

Nedávno Apple vypustil jazyk Swift, který je navržen tak, aby byl schopen spolupracovat s kódem v Objective C, přináší moderní vlastnosti – closure, generické programování, automatickou správu paměti, rozhraní (pojmenovaná jako „protocol“), ale i funkcionální prvky [20].

Android

Na rozdíl od iOS je Android plně otevřený. Je založený na Linuxu a jeho licence spadá do „Open Source“[2] kategorie specifikované skupinou „Open Source Initiative“, tzn. že je volně dostupný zdrojový kód, kdokoliv může provádět změny a ty distribuovat.

Google, společnost stojící za Androidem, se rozhodl, že platformou pro aplikace na Android se stane Java. Toto rozhodnutí může působit trochu rozporuplně, volit Javu, která je stále často vnímána jako hardwarově náročná, na slabá mobilní zařízení. Hlavní důvod ale byla jednoduchost a přímočarost Javy a velký počet vývojářů dobře ovládající tento jazyk [17]. Google vyvinul vlastní JVM[3] – Dalvik VM, který silně optimalizoval pro v té době (vydání první verze Androidu v roce 2008) nepříliš výkoná mobilní zařízení. V nynější nejnovější verzi Androidu 5.0 Lollipop se upouští od Dalvik VM, který je nahrazen ART (jedna z klíčových vlastností ART je ahead-of-time kompilace umožňující rychlejší spouštění aplikací) [3].

4.1.3 Specifické rysy vývoje pro mobilní platformy

Oproti vývoji aplikací na osobní počítače (desktop) se při vývoji pro mobilní platformu a konzole musejí používat tzv. emulátory. Jde o software, který napodobuje cílové zařízení, v našem případě určitý chytrý telefon s nějakou verzí operačního systému.

Práce s emulátorem není vždy jednoduchá, aplikace se po přeložení musí nahrát do emulovaného přístroje a nainstalovat, stejně jako by se to dělo na opravdovém mobilním telefonu či tabletu. Většinou bývají tyto akce automatizované ve vývojovém prostředí, programátor musí jen počkat.

[2] Konkrétně jde o „Apache Software License 2.0“ – velmi volná licence, části jsou pod „GPLv2“ (např. jádro).

[3] Java Virtual Machine – provádí samotné vykonávání programu napsaného v Javě nebo jiném JVM kompatibilním jazyku.

Dalším negativním rysem je výkon emulovaného přístroje, který nemusí nutně vadit u aplikací komunikujících s uživatelem pouze pomocí formulářů, zato u her bývá tíživý problém s nedostatečným výkonem opravdu běžný. Vývojář je pak nucen vše zkoušet na opravdovém přístroji, což opět znepříjemňuje a komplikuje vývoj (ladění přímo na chytrém telefonu nebývá moc pohodlné).

4.2 Počítačové operační systémy

4.2.1 Vývoj operačních systémů na počítačích

V tzv. nultém období (1940–1955), v úplných počátcích výpočetní techniky, se žádný operační systém nepoužíval. Veškeré programovaní spočívalo v zadávání příkazů na úrovni strojového kódu pomocí propojovacího panelu (plugboard). V té době vznikali předchůdci plnohodnotných operačních systémů – *knihovny*, které lidé psali za účelem úspory času a práce.

V první fázi, která je ohraničena roky 1955 a 1970, se zefektivňuje užívání počítačů. Uživatelé už nutně nemusí být neustále přítomni u stroje a vznikají první jednoduché operační systémy. Slouží především k dávkovému spouštění uživateli zadaných úloh – zobrazuje stav, v případě chyby uloží obsah paměti, spustí další úlohu. Později OS už řeší i řadu obtížnějších úkolů, např. sdílení systému mnoha uživateli, více spuštěných úloh (multiprogramming), ochrana paměti mezi úlohami, vyrovnávací paměť pro načítání další úlohy (souběžně s prováděním aktuální úlohy), prioritizace podle časové náročnosti úlohy. Operační systémy této doby byly velmi komplikované a neudržovatelné – byly totiž psány v jazyku symbolických adres[4].

Etapa druhá trvající od 1970 do 1980 zavádí převratný koncept – souběžné používání počítače více uživateli. Každý uživatel má vlastní terminál a z něj může používat počítač. Dochází ke zrodu konceptu souborového systému. V této etapě vzniká operační systém CTSS (MIT), který podnítil vznik legendárního MULTICS (MIT, Bell Labs a General Electric). Zavádí mnoho konceptů, ze kterých vycházejí operační systémy dodnes – privilegovaný režim (protected rings), hierarchické souborové systémy, zařízení vystupující jako soubory. Vzniká UNIX – komerční přenositelný operační systém psaný v jazyce C. Odtud pochází nápady jako OS napsaný ve vyšším programovacím jazyce, přenositelnost, roury (pipes), připojitelné souborové systémy (mount).

Ve třetím období nastupuje na scénu operační systém CP/M. IBM potřebovalo software pro jejich osobní počítače, bohužel CP/M nebyl dodělaný. Bill Gates (Microsoft) zachraňuje situaci příslibem, že rychle vytvoří OS. Skupuje 86-DOS a tak vzniká MS-DOS, který je schopen spouštět programy pro CP/M. Operační systém nyní slouží jako knihovna podprogramů a k vykonávání příkazů.

Čtvrtá fáze (1990–) se vyznačuje počítačovými sítěmi – konektivita je primární požadavek. Síťové aplikace ženou počítačový průmysl (web, e-mail). Vznik poskytovatelů připojení, informace se stává komoditou, reklama je běžnou součástí.

Toto krátké shrnutí vychází z [42], tam také může čtenář nalézt mnohem detailnější informace o vývoji operačních systémů.

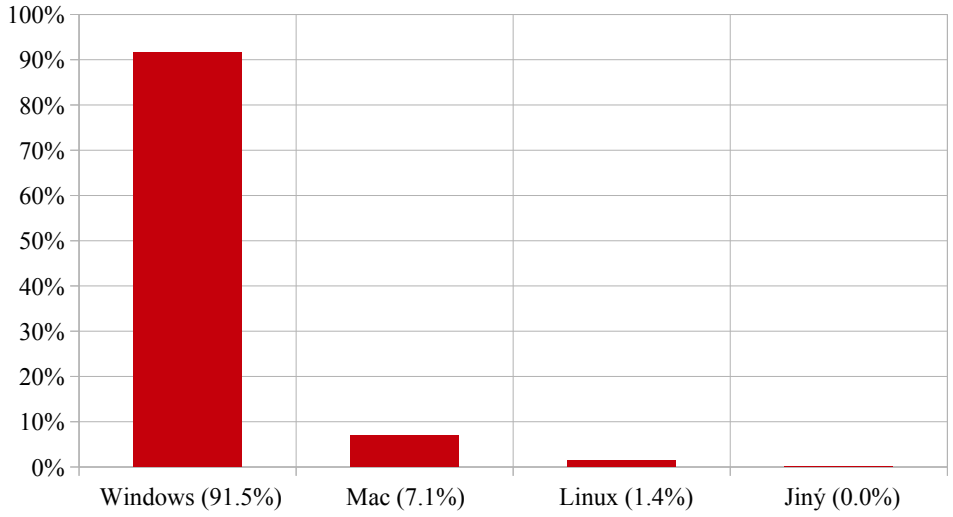
4.2.2 Aktuální stav operačních systémů na osobních počítačích

Největšími hráči na poli osobních počítačů (ne všech počítačů – na serverech je situace diametrálně odlišná) jsou operační systém od firmy Microsoft – Windows, OS X postavený na

[4] Označení „jazyk symbolických adres“ nebo také „jazyk symbolických instrukcí“ se v praxi nepoužívá, běžně se setkáváme s kratším pojmenováním „assembler“.

UNIXu [34] a Linux[5] – svobodná implementace UNIXu.

Z obrázku 4.2 je patrné, že systémy od firmy Microsoft dominují trhu s desktopovými operačními systémy. Přesto vidíme, podle mě sílící, trend ve vydávání multiplatformních aplikací.



Obrázek 4.2: Zastoupení OS na PC v říjnu 2014, zdroj dat: netapplications.com

Určitě stojí za zmínku, že přestože zastoupení Linuxu je něco přes 1 %, tak několik statistik ukazuje, že v průměru na jeden desktop s Linuxem připadá podstatně více hráčů, než je tomu u systémů Windows a Mac [25][7].

4.3 Multiplatformní vývoj

Pokud chceme vyvíjet aplikaci, nemůžeme si dovolit ignorovat Windows. Stejně tak si ale myslím, že je výhodné cílit i na další platformy – Mac i Linux. Mají sice podstatně menší zastoupení, ale většinou také mnohem menší konkurenci a pokud je aplikace portována na Mac, tak port na Linux nebývá velký problém, protože tyto systémy jsou celkem podobné.

Co se týká mobilního trhu, tak je nutnost podporovat Android a iOS, Windows není priorita, pouze bonus, pokud ho framework/knihovna/platforma podporuje.

Rozhodnutí o podporovaných platformách musí být ale učiněno v počáteční fázi, kdy se teprve rozhoduje o tom, jaký se použije programovací jazyk, herní engine, knihovny a frameworky. Pokud se vytváří port aplikace na další platformu až zpětně, může to vést k drastickému prodražení, hlavně v případě používání nemultiplatformních řešení.

Z předchozího textu by se mohlo zdát, že vyvíjet aplikaci multiplatformně je vždy lepší, než mít nativní aplikaci pro každou platformu zvlášť. Nemusí to být ale nutně pravda, alespoň u „neherních“ aplikací. Každá mobilní platforma má totiž určitý styl ovládacích prvků v uživatelském rozhraní, jiné doporučené postupy a často i pro danou platformu unikátní ovládací prvky. V případě použití multiplatformní knihovny ale nezohledňujeme prvky specifické pro konkrétní platformu[6] a aplikace tak může působit negativním dojmem, že nezapadá do zbytku systému (vzhledem, ovládáním, reakcemi ovládacích prvků) [24].

[5] Někdy je možné jej také vidět pod označením „GNU/Linux“, které se ale moc nerozšířilo. Více informací lze získat zhlédnutím filmu „Revolution OS“ [29].

[6] Ať již kvůli technickým omezením knihovny, nebo protože nechceme mít spoustu kódu pro každou platformu zvlášť – proto jsme si přece zvolili multiplatformní knihovnu.

V následující části představím několik nejznámějších multiplatformních knihoven.

4.3.1 Xamarin

Placená platforma těžící z jazyka C# a prostředí .NET [52]. Výsledné aplikace lze provozovat na všech hlavních mobilních platformách (Android, iOS a Windows Phone).

Nelze je ale beze změn provozovat na osobních počítačích – musí se implementovat speciální GUI jen pro desktop[7]. Je tedy obtížnější ladění a vše se musí provádět v emulátoru mobilního zařízení.

Xamarin přímo vývoj her nepodporuje, pro 2D hry lze ale použít např. CocosSharp knihovnu, která se postará o vše související s grafikou, zvukem, ovládáním a scénami.

Existují ale i důvody pro nepoužití Xamarinu, např. objemnější výsledné aplikace, omezené sdílení UI kódu mezi cílenými platformami, menší komunita a občasná nestabilita API [51].

4.3.2 Unity

Velká platforma cílící na vývoj her, 2D i 3D, využívající platformu .NET s hlavním jazykem C# [47]. Podpora koncových platforem je ohromující, z mobilních to jsou všechny hlavní – tj. iOS, Android a Windows Phone, z desktopových Windows, Mac i Linux a z konzolí PS4, XBOX360 a Wii. Užívání Unity je placené (existuje verze zdarma, která je značně omezena).

Je třeba ale zdůraznit, že Unity je kompletní balík pro vývoj her, obsahuje nástroje pro skriptování, tvoření scén ale i kupříkladu ladění výkonu (profiler).

4.3.3 jMonkeyEngine

Plnohodnotný 3D Open Source herní engine s jazykem Java jako hlavním [21]. Má užší záběr, co se podporovaných platforem týče, zato ale nabízí opravdu hodně. Hlavní mobilní (Android a iOS) i desktopové platformy (Windows, Mac, Linux) jsou podporovány, aplikaci lze přeložit i jako tzv. applet a provozovat přímo na webových stránkách.

Přestože nedosahuje záběru komerčního Unity, i tak oplývá nemálo pokročilými funkcemi, mezi ně patří např. pokročilé techniky vytváření stínů (PSSM, SSAO), podpora shaderů, vlastních materiálů, filtry (glow, bloom), částicové efekty nebo simulace fyziky.

Již podle volby licence se dá očekávat, že použití jMonkeyEngine není zpoplatněno.

4.3.4 libGDX

Open Source framework pro vývoj především mobilních 2D[8] her [53]. Hlavním jazykem je Java, z vlastní zkušenosti ale mohu říct, že práce ve Scale je také pohodlná.[9] Framework je dostupný zdarma, podporuje všechny hlavní mobilní platformy (Android, iOS, BlackBerry) i desktopové (Windows, Linux, Mac). Při dodržení určitých postupů a doporučení lze výslednou aplikaci přeložit do webové podoby HTML + JavaScript + WebGL.

Přestože jej tvůrce adresuje jako framework, pokrývá všechny potřeby pro vytvoření 2D mobilní hry, nebál bych se označení herní engine.

[7] Xamarin.Forms nepodporují WPF – GUI framework od Microsoftu.

[8] Existuje částečná podpora 3D, stále ale v rané vývojové fázi.

[9] Vyjma serializace do JSONu, tam jsem narazil na problémy s kolekcemi Scaly, což se ale dalo celkem očekávat.

4.3.5 Qt

Knihovna zaměřená na tvorbu aplikací s GUI [35]. Hlavní jazyk je C++, z toho vyplývá vyšší náročnost na programátora v podobě kódu starajícího se o správu paměti (na platformě .NET i JVM je tento požadavek minimální). Qt je zdarma pro projekty s Open Source licencí GPL a LGPL, jinak je nutné zaplatit. Podporuje 2D i 3D, jde ale stále o knihovnu, nikoliv o plnohodnotný herní engine [54].

Podporuje hlavní mobilní platformy – Android, iOS i Windows Phone. Stejně tak dobře je na tom podpora desktopových platforem – Windows, Mac i Linux, ve starších verzích také Solaris a AIX.

4.3.6 Cocos2d-x

Cocos2d-x je 2D Open Source herní engine s MIT licencí [9]. Primárním jazykem, který se používá pro vývoj, je C++. Zajímavostí je, že podporuje i jazyky JavaScript a Lua (přestože výkon asi nebude srovnatelný s nativním C++).

Podporované platformy nepřekvapí, z mobilních to jsou iOS, Android a Windows, z desktopových pak Windows, Mac a Linux.

4.3.7 HTML5

Jak je zmíněno v [14], místo nutnosti skvěle zvládat C#, Windows API a Visual Studio pro Windows Phone, Objective-C, Xcode a iOS API pro iOS a jazyk Java, Android API a Eclipse pro tvoření aplikací pro Android, je efektivnější zvládnout jedno multiplatformní API a jeden jazyk. Čas, který by byl vynaložen na psaní portů na různé platformy, může být využit k implementaci dalších vlastností, k rozšiřování aplikace.

Jako volba pro tento účel se přímo nabízí HTML5 + CSS3 + JavaScript. Hotová aplikace může být nasazena jako klasická aplikace pro danou platformu (prostý balíček k nainstalování), bez nutnosti provozovat ji ve webovém prohlížeči. Takové aplikaci pak říkáme „hybridní webová aplikace“.

Je ovšem důležité se vyhnout částem kódu specifických pro určitou platformu, případně je dostatečně minimalizovat. Nejproblematičtější bývá přístup k hardwaru – tj. např. fotoaparát, GPS nebo akcelerometr. Nic ale není ztraceno, existuje řada knihoven, které abstrahují tyto API specifické pro jednu platformu. Jednou z nich je zdarma dostupná Open Source knihovna Apache Cordova (případně její distribuce PhoneGap, za kterou stojí Adobe) [5].

Výsledné aplikace nemusí být jen 2D, lze využít JavaScript API WebGL pro vykreslování 3D obsahu. Také se doporučuje použít nějakou knihovnu či framework, protože psaní pouze v čistém JS není zdaleka tak produktivní jako použití knihovny hotové a léty ozkoušené, a to jak programátory, tak uživateli. Vhodnými kandidáty jsou ku příkladu jQuery Mobile, Zepto.js nebo Sencha Touch.

Kapitola 5

Vybrané aplikace pro podporu výuky

V této kapitole se pokusím popsat a případně srovnat několik různých aplikací, které se snaží uživatele seznámit se základy či některými aspekty logických systémů.

Člověk hledající výukové aplikace se zaměřením na logické obvody narazí na mnoho, často velmi jednoduchých, aplikací, které jsou spíše neinteraktivní zjednodušená elektronická podoba papírové učebnice. Několik z nich je představeno v oddílu 5.1. Osobně zastávám názor, že jde o nedostatečné využití média.

Existuje velké množství simulátorů logických obvodů, mnoho z nich zdarma i on-line. Některé z nich předvedu v sekci 5.2. Většina aplikací dále uvedená je dostupná zdarma, pokud tomu tak není, tak tento fakt zmíním.

Rád bych se ale zaměřil na herní počiny, především puzzle hry. Důvod je, myslím si, celkem zřejmý – výstupem této práce bude jednoduchá puzzle hra snažící se hráče zlehka uvést do tajů hradel, klopných obvodů i celých logických systémů.

5.1 Elektronické učebnice

5.1.1 Logic gates

„Logic gates“ – aplikace sloužící k výuce hradel (AND, NAND, OR, NOR, XOR, NXOR, NOT), klopných obvodů (D, JK, T, RS) a jednodušších logických obvodů. Každá položka nejdříve zobrazí pravdivostní tabulku, pak následuje interaktivní vyobrazení obvodu či hradla. Tato aplikace, na rozdíl od mnoha jiných v této kategorii, představuje poměrně široký výběr klopných obvodů.

Musím ale zdůraznit, že aplikace jako taková je velmi omezená – nelze skládat vlastní obvody, pouze měnit vstupy u před-vytvořených obvodů. Dalším negativem je nepříliš dotažené grafické zpracování – znázornění obvodů, velmi výrazné barvy, často se překrývající prvky 5.1a.

5.1.2 Logical Gates

Aplikace ukazuje pravdivostní tabulky hradel NOT, AND, OR, NAND, NOR, XOR a XNOR. Na rozdíl od předcházející neobsahuje interaktivní ukázku hradel a obvodů. Nabízí „kalkulačku“ vyčíslující vždy právě jednu logickou operaci.

Co ale oproti všem zde uvedeným aplikacím zvládá, je předvedení zapojení nejznámějších hradel jen za pomoci NOR nebo NAND.

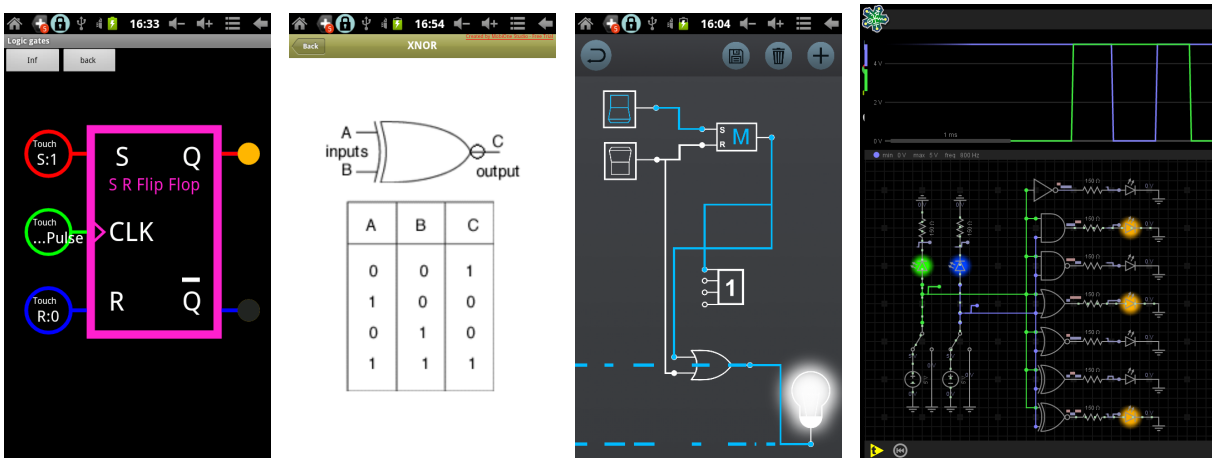
5.2 Simulátory

5.2.1 The LOGIC LAB

„The LOGIC LAB“ je webová aplikace využívající platformu Flash. Obsahuje nejpoužívanější hradla – AND, OR, NOT, XOR – spolu s jejich negovanými variantami. Dokonce je přítomno i několik klopných obvodů – T, RS a JK. Aplikace je dostupná zdarma [44], ukázku obvodu vidíme na obrázku 5.2a.

5.2.2 Logic Simulator Pro

Jde o stroze vyhlížející simulátor nejen logických obvodů cílený na mobilní platformu. Obsahuje hodiny, generátory signálu, spínač, žárovku, jedno-číselný LCD displej, hradla AND, OR, NOT, identitu, NAND, NOR, XOR, XNOR a jeden klopný obvod – RS.



(a) Logic gates (b) Logical Gates (c) Logic Simulator Pro (d) EveryCircuit Free

Obrázek 5.1: Aplikace pro podporu výuky

Občas nastávaly problémy s křížením kabelů (lze spatřit v obrázku 5.1c). Propojování kabelů se provádí tažením z výstupu do vstupu. Nebylo jakkoliv znázorněné, dokud nebyl spoj vytvořen, aplikace tedy působí, že nereaguje na uživatele.

5.2.3 EveryCircuit Free

Program je velmi propracovaný interaktivní simulátor elektrických obvodů. Nezaměřuje se na logické obvody, obsahuje pouze nejznámější hradla – AND, OR, NOT, NAND, NOR, XOR, XNOR. Žádné klopné obvody nejsou přítomny.

Nevýhodou verze zdarma je velmi omezená velikost pracovní plochy, omezená nabídka ukázkových obvodů a nepřístupné pokročilejší komponenty. Dalším negativem je nepříliš populární vyžadování připojení k Internetu, tzv. „always-online“, přestože aplikace neobsahuje nic, co by tento požadavek ospravedlňovalo.

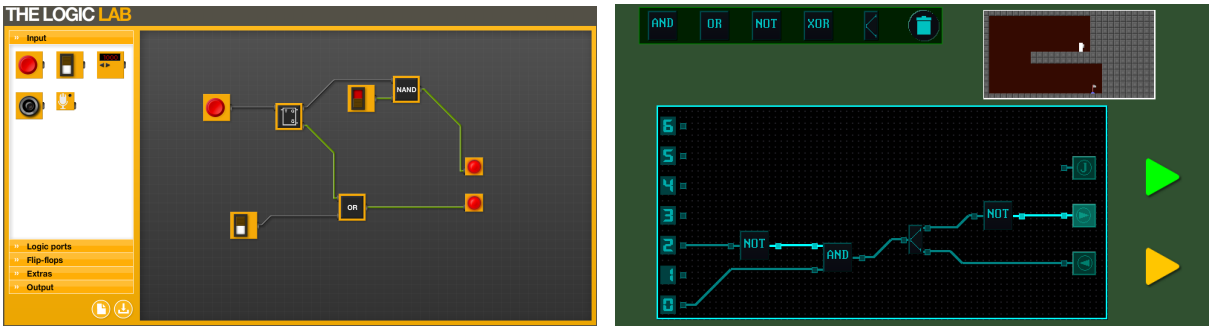
5.3 Hry

5.3.1 Pocket Robots Test Chamber

„Pocket Robots Test Chamber“ je velmi jednoduše vypadající puzzle hra, ve které je úkolem hráče provést robota 2D bludištěm (podle typu úrovně pohled ze strany nebo ze shora).

Splnění úkolu provádí hráč nepřímo konstrukcí logického obvodu, který po spuštění simulace ovládá robota. Vstupy obvodu (na levé straně obrazovky) mohou být připojeny na různé zdroje signálu podle typu úrovně. Např. s přibývajícím časem se postupně zapínají vstupy 0–6, nebo jsou vyvedeny senzory barvy pozadí. Výstup obvodu je připojen k řízení robota, které sestává z několika málo příkazů – jeď doleva, doprava a zapni jetpack [12].

Hra je chytlavá a po několika úvodních úrovních nabírá na obtížnosti. Zkoušel jsem verzi pro PC, ovládání je typické pro mobilní platformy. Kromě opravdu jednoduchého grafického provedení a občasnému nehezkému přeskládání kabelů nemohu hře moc vytknout.



(a) The LOGIC LAB (b) Pocket Robots Test Chamber

Obrázek 5.2: Aplikace pro podporu výuky

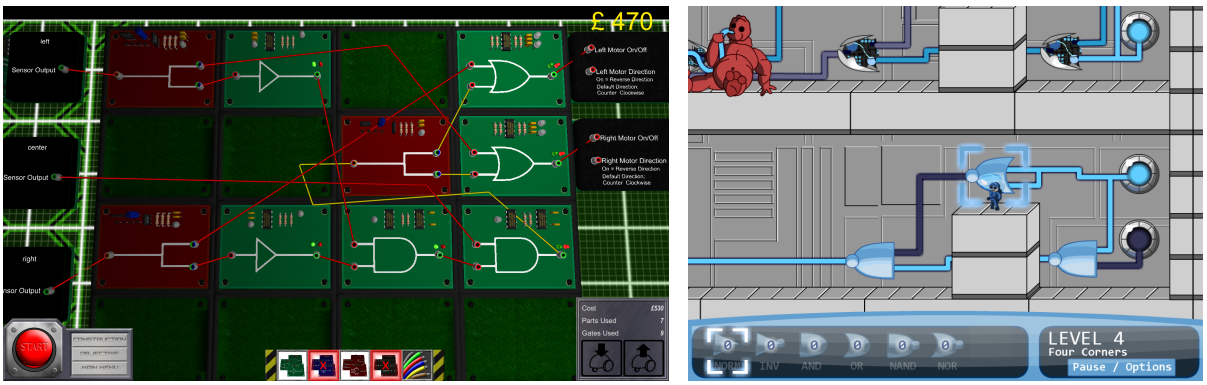
5.3.2 LogicBots

Úkolem hráče v této placené hře je postavit robota a vytvořit ovládací logický obvod tak, aby robot splnil předložený problém – např. následovat čáru na zemi do cíle [50].

Hra dokáže hráče vtáhnout a zabavit na dlouhý čas. Předpokládám, že je toho dosaženo tím, že hráči je dána obrovská volnost při stavbě robota. V plné verzi je i tzv. „sandbox“, kde jsou jednotlivé díly dostupné v neomezeném množství. Hudba výstižně dokresluje uvolněnou atmosféru vhodnou pro stavbu a sledování robota.

Hra se nezaměřuje čistě na logické obvody, obsahuje základní hradla (AND, OR, NOT, XOR), generátory signálu, časovače, ale i tranzistory, kondenzátory a další prvky.

Ovládání není intuitivní, často se v různých módech liší (např. prostřední tlačítko myši mívá různý význam, dalším problémem je chybějící možnost přesunutí či zkopírování hradla). Přestože graficky hra působí jednodušeji, FPS tomu neodpovídá – nedostatečně optimalizované vykreslování. Nutno podotknout, že hra je stále ve vývoji. Je tedy pravděpodobné, že část problémů bude opravena do vydání titulu.



(a) LogicBots (b) Gates of Logic

Obrázek 5.3: Hry pro podporu výuky

5.3.3 Gates of Logic

Plošinová hra stojící na platformně Flash. Užívá jak prvky typické pro akční hry, např. herní postava může zemřít při kolizi s bodáky nebo virem (nepřítel), tak i prvky specifické pro puzzle hry – přesouvání hradel za účelem vyřešení problému, kterým může být vypnutí silového pole, zlikvidování viru nebo dokončení úrovně.

Hra je po grafické i estetické stránce solidní, hudba i zvukové efekty vhodně dotvářejí atmosféru. Ovládání občas působí zpomaleně.

„Gates of Logic“ nabízí následující hradla: identita, invertor, AND, OR, NAND, NOR a XOR. Herní logika rozlišuje, na rozdíl od typických dvou, tři stavy – zapnuto, vypnuto a nedefinováno (ve hře vykresleno jako nepravidelné blikání vodiče) [41].

Kapitola 6

Specifikace požadavků a návrh

V této kapitole popíši body, které by aplikace měla splňovat. Následně stručně představím zvolenou architekturu a ukáži návrh několika nejdůležitějších tříd.

6.1 Požadavky na hru

Aplikace by neměla působit nepřátelsky, komplikovaně. Především první úrovně musí jasně vysvětlovat herní mechaniky. Každé hradlo či klopný obvod je nutné představit – např. krátkým textovým popisem, animací vyjadřující funkci, tabulkou (může obsahovat i obrázky). Důraz by měl být kladen na názornost a výstižnost, ku příkladu vodiče by měly mít jasně odlišené stavy různou grafickou reprezentací.

Přestože půjde o puzzle hru, počáteční úrovně nesmí počítat s jakoukoliv předchozí znalostí logických systémů, obtížnost těchto úrovní musí být relativně nízká a narůstat velmi pomalu.

Podobně jako je nežádoucí příliš rychlý příval informací o logických systémech (např. velký počet nových hradel v jedné úrovni), je stejně nežádoucí po hráči vyžadovat řešení příliš komplikovaných problémů (přinejmenším v úrovních představující nové prvky a v počátečních úrovních obecně). Za příliš komplikovaný problém považuji např. pokládání většího množství vodičů. Je nutno se zamyslet nad tím, zda vůbec někdy nechat hráče umisťovat vodiče.

Stojí za zvážení, zda se pouštět do systému odemykání úrovní. Tato persistence je totiž problematická, pokud by aplikace běžela na školních zařízeních, která jsou používána desítkami uživatelů. Na základních a středních školách se často nevidí, že by každý student měl vlastní účet.

Ovládání hry musí být pohodlné na dotykových zařízeních, je tedy nutné to zohlednit v GUI. Příkladem může být situace, kdy hráč přidává prvek obvodu – objekty by měly jít jednoduše přetáhnout z oblasti inventáře na herní plochu. Stejně uzpůsobené musí být i ovládací prvky, např. tlačítka a jakékoliv aktivní prvky nesmí být příliš malé, aby je bylo možné spolehlivě trefit nejen myší, ale i prstem na dotykové obrazovce.

Hra bude obsahovat úrovně seřazené podle obtížnosti. Každá součástka bude představena při prvním užití (tj. pokud nebyla použita v žádné úrovni předcházející aktuální). V následujících úrovních se již bude počítat s tím, že hráč zná tuto součástku. Přesto bude možné, např. po kliknutí na prvek na herní ploše a vybrání ikony otazníku, vyvolat krátkou nápovědu obsahující popis vybrané komponenty.

6.2 Prostředky

Cílová platforma hry je především Android, nechtěl jsem se ale vzdát i možnosti provozovat aplikaci na osobních počítačích.

6.2.1 Scala

Protože byl zvolen Android, tak se jako implementační jazyk nabízí Java. Již nějakou dobu jsem ale experimentoval s jazykem Scala, který je s Javou kompatibilní[1]. Scala obsahuje rysy jak objektového, tak i funkcionálního programování. Jde o silně staticky typovaný jazyk, který klade důraz na stručnost a výstižnost zdrojového kódu [32]. Nakonec jsem zvolil právě jazyk Scala, v následujícím odstavci osvětlím mé rozhodnutí.

Vyjmenuji několik důležitých vlastností: oproti Javě jsou i funkce objekty, statické a singleton třídy nahradil *objekt*, nemalou část typů může odvodit přímo překladač (není tedy nutné je explicitně uvádět), odpadá nutnost používat get a set metody, neměnné (immutable) datové struktury, širší možnosti generického programování, case třídy, podpora tzv. „for-comprehension“ a „pattern matching“. Jak vidno, tak vlastnostmi se může rovnat i např. velmi oblíbenému jazyku C#, který v nemálo případech i předčí.

Za zmínku stojí také fakt, že Scala samotná nenutí k jinému stylu programování, pouze obsahuje prostředky, které funkcionální styl umožňují.

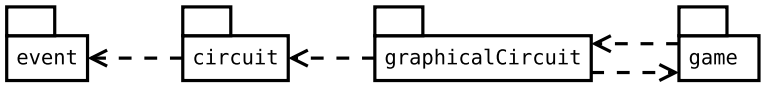
6.2.2 libGDX

Jako hlavní knihovna byla zvolena *libGDX* (představena v kapitole 4.3.4), která podporuje většinu mobilních i desktopových platforem. libGDX je na tvorbu 2D her ideální, obsahuje podporu pro animace, zvukové efekty, základní GUI prvky, vstup z klávesnice, myši i dotykových zařízení. Zároveň je k dispozici zdarma pod Open Source licencí a existuje poměrně velký počet informačních zdrojů.

6.3 Návrh aplikace

6.3.1 Architektura

Vrstva simulační a grafická budou odděleny, simulace je zcela nezávislá na grafice, viz obr. 6.1.

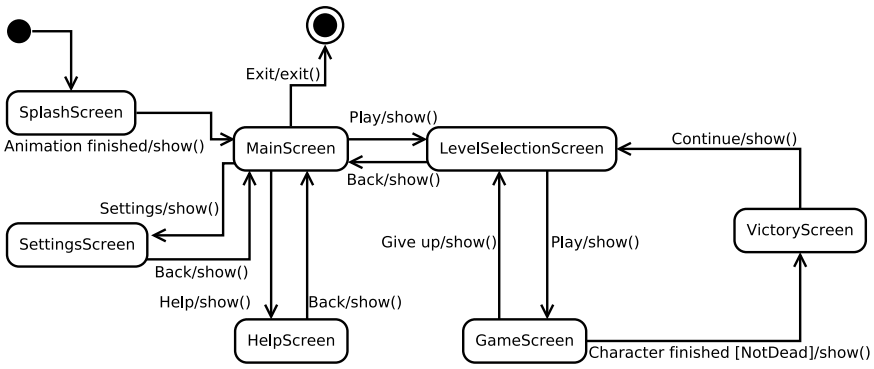


Obrázek 6.1: Zobrazení vztahů mezi vybranými balíky

Třídy v simulačním balíku `circuit` reprezentují chování daného prvku či vodiče v obvodu. Obsahují propojovací logiku a vhodné vyjádření pro rychlou simulaci. Každý prvek disponuje schopností serializace (vodiče se neukládají zvlášť, propoje jsou ukládány spolu s prvky).

Balík `graphicalCircuit` se skládá ze tříd, které reprezentují překážky, senzory, hradla, klopné obvody, zdroje signálu, vodiče a celý obvod v grafické podobě. V tomto balíku se klade důraz na propojení se třídami z balíku `circuit`, dále na umístění, zobrazení a manipulaci s částmi obvodu na herní ploše.

Detailní návrh důležitých tříd v podobě diagramů si můžete prohlédnout v příloze C. Při jejich tvorbě jsem používal značení popsané v příloze B.



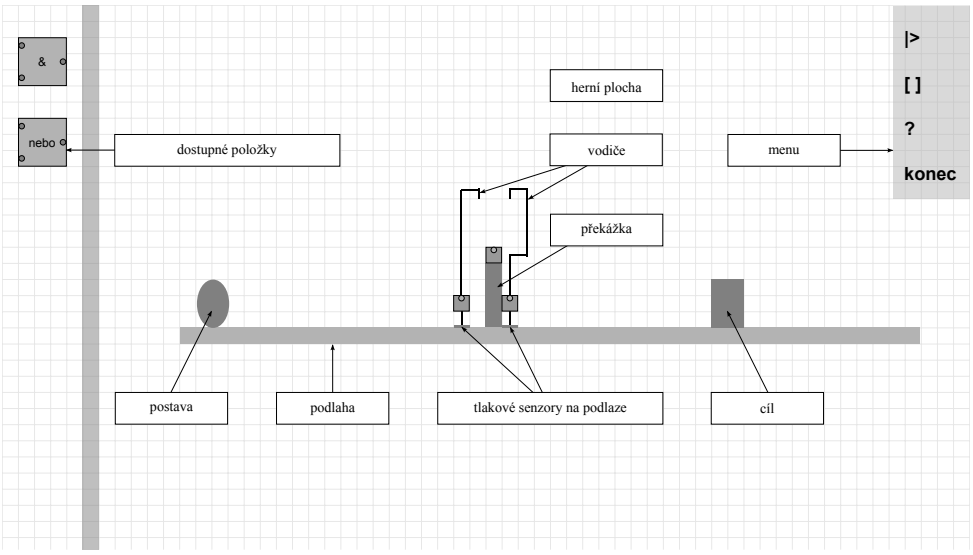
Obrázek 6.2: Návrh obrazovek

6.3.2 Obrazovky

Obrázek 6.2 demonstruje návrh obrazovek. Domnívám se, že vše důležité je zřejmé z diagramu. Návrh je totiž v podstatě shodný s jakoukoliv jednodušší hrou.

6.3.3 Herní obrazovka

Koncept rozložení prvků na herní obrazovce vidíme na obrázku 6.3. Na levé straně se nachází inventář s prvky, které jsou hráči v aktuální úrovni dostupné. Vpravo nahoře leží herní menu, které slouží k ovládání simulace (spuštění, zastavení), zobrazení nápovědy nebo popisu úrovně a opuštění úrovně. Na většině obrazovky se rozprostírá herní plocha.



Obrázek 6.3: Návrh rozložení ovládacích prvků na obrazovce

Ta obsahuje dvě vrstvy – popředí nese především postavu, terén a překážky. Vrstva více v pozadí je osazována komponentami obvodu – hlavně vodiči, hradly a klopnými obvody. Herní plocha je rozdělena mřížkou na jednotlivá políčka. Vodič se šířkou právě jednoho políčka patří

[1] Existují jistá omezení a především čitelnost kódu je dostačující pouze při užívání Java tříd ve Scale, ne naopak.

k nejmenším objektům, které lze umístit na pracovní plochu. Hradla a klopné obvody jsou podstatně větší, často zaujímají 3×3 políčka (v obrázku hradlo OR a AND).

Postava se po spuštění simulace začne přesouvat směrem doprava. Při své cestě může spínat různé senzory, na obrázku vidíme tlakové senzory na podlaze. Úkolem hráče je zajistit bezpečný přesun postavy ze startovní pozice do cílové.

6.3.4 Obrazovka výběru úrovně

Návrhy obrazovky k výběru úrovně vidíme na obrázku 6.4. Návrh v části (a) poskytuje informace o několika úrovních zároveň. Při větším počtu úrovní lze posouvat obrazovkou tažením a zobrazit si tak další řadu. Menší nevýhodou je duplikace některých prvků, např. tlačítka spouštějícího úroveň. Výhodu má v přehlednosti, ve větším množství relevantních informací a ve snadnějším ovládání na dotykovém zařízení.

Obrázek 6.4: Dva návrhy obrazovky výběru úrovně

Druhý návrh, část (b), následuje schéma více známé z počítačových uživatelských rozhraní. Vlevo je výběr podle jména úrovně, vpravo se zobrazují informace o aktuálně vybrané úrovni. Tento přístup vyžaduje více uživatelského vstupu pro výběr úrovně – je nutné najít v seznamu jméno úrovně, vybrat jej a pak potvrdit tlačítkem. Pokud uživatel hledá úroveň, ale nezná její označení, pak musí postupně vybírat jednotlivé úrovně, aby si zobrazil popis, což je dost nešikovné.

Protože se od výsledné aplikace očekává, že bude dobře použitelná na dotykových zařízeních, byl vybrán návrh (a).

Kapitola 7

Implementace

7.1 Aplikované principy

V průběhu implementace jsem se snažil dodržovat několik zvolených principů, v této části je vyjmenuji a krátce popíši. Vycházel jsem především z článku „The Principles of Good Programming“ [11].

7.1.1 DRY

DRY – jeden z nejdůležitějších principů programování a návrhu vůbec. DRY je zkratka anglické fráze „Don't repeat yourself!“, která se překládá jako „Neopakuj se!“. V programování máme k dispozici hned několik konstrukcí, které přímo zachycují tuto radu – smyčky, funkce, třídy, atd. Jakmile programátor napíše stejný nebo velmi podobný kód po několikáté (kupř. předpis výpočtu, několik po sobě jdoucích stejných příkazů, řešení stejného problému), tak nastává čas na zavedení nové abstrakce.

Za ukázku aplikace tohoto principu můžeme považovat „package object dip“, který sdružuje různá rozšíření běžných typů (označováno jako návrhový vzor „Pimp my Library“), např. třída „SeqPimps“ umožňuje filtrování sekvence podle typu, vytvoření náhodně zamíchané sekvence nebo vrácení náhodného prvku. Tyto kusy kódu se opakovaly na různých místech, byly proto přesunuty a sjednoceny na jednom místě, kde jsou lehce přístupné odkudkoliv.

7.1.2 KISS

Zkratka pochází z anglického „Keep it simple, stupid!“. Princip uvádí, že máme dělat co nejjednodušší systémy, třídy či jiné celky.

Funkcionální programování tento princip přirozeně splňuje, protože preferuje čisté krátké funkce („pure“, bez vedlejších efektů). Stejně tak upřednostňování jednoduchých neměnných (tzv. „immutable“) tříd následuje princip KISS.

Scala umožňuje ale kombinaci objektového a funkcionálního přístupu, navíc knihovny pocházející z Java prostředí bývají objektově orientované. Když se na problematiku podíváme obecněji, tak při důrazu na výkon a stabilitu vykreslování bývají mnohem výhodnější „mutable“ třídy a objekty[1].

Při vývoji jsem se v některých případech nevyhnul komplexním třídám. Problém jsem řešil tak, že jsem velkou komplexní třídu rozdělil na více krátkých konstruktů trait podle zaměření kódu – např. vytvoření, serializace, vykreslování, reakce na posun v simulačním čase, reakce na uživatele, atp.

[1] Mutable přístup je nutno užít minimálně pro operace volané při vykreslování každého snímku. Alokace paměti a následné automatické uvolnění garbage collectorem jsou časově náročné operace.

7.1.3 SRP

SRP označuje „Single responsibility principle“, což se překládá jako „Princip jedné odpovědnosti“. Rada nám říká, že každá část kódu (třída, metoda či funkce) by měla provádět pouze jeden jasně specifikovaný úkol.

Ne vždy je jednoduché vyhodnotit, zda daná třída provádí právě jeden úkol, především v situacích, kdy jsou potřeba splnit určité krátké dílčí úkoly. Je tedy vždy značně subjektivní, kdy se rozhodnout, že je nutné aplikovat SRP a rozdělit kód na více částí.

Asi největší dopad užití tohoto principu bylo rozdělení popisu obvodu na dvě části – simulační (stará se pouze o popis chování) a část grafickou (je zodpovědná primárně za zobrazování obvodu a interakci s uživatelem).

Jak bylo zmíněno výše, v jazyce Scala může člověk využít vícenásobné dědění pomocí konstruktů trait. Je tedy možné rozdělit různé pod-úkoly do samostatných konstruktů trait, ze kterých bude následně dědit jedna třída, a tak aplikovat princip SRP.

7.1.4 Použitelnost a uživatelská spokojenost

Jak bylo nastíněno v sekci 6.1, aplikace by měla být relativně jednoduchá na ovládání a intuitivní. V této podsekci vyložím, která pravidla, doporučení a principy jsem aplikoval při návrhu a vývoji aplikace.

Spojení „uživatelská spokojenost“[2] je více používané v anglické variantě – „user experience“, častěji viděno pod zkratkou „UX“. Jde o obor zkoumající chování osob, jejich přístup a emoce při používání určitého systému nebo produktu.

„Použitelnost“ je vlastností předmětu, která popisuje jednoduchost a snadnou naučitelnost zacházení s tímto předmětem.

ISO 9241

V určitých aspektech jsem vycházel ze standardu ISO 9241. Kladení důrazu na iterativnost vývoje uživatelského rozhraní odráží tento standard při vývoji herní aplikace nejvíce. Příkladem mohou být následující vylepšení:

- Na porty prvků byly přidány malé šipky, které znázorňují směr toku informace. Vylepšení výrazně usnadňuje orientaci především v případě prvků se stejným počtem vstupů i výstupů.
- Předělání grafického objektu reprezentujícího úroveň v obrazovce výběru úrovně z pouhého tlačítka na obdélníky, které jsou lépe graficky ztvárněné, přehlednější a především zvýšily plochu, na které může uživatel použít gesto uchopení a posouvat obsah obrazovky.
- Umístění do horní části obrazovky tzv. „creative inventory“, tj. nabídky všech prvků dostupných ve hře při tvoření nového obvodu, se ukázalo jako nepříliš dobrá volba. Buď byla nabídka příliš malá a na dotykových zařízeních byl výběr a posun nepohodlný, nebo zabírala příliš velkou část obrazovky. Nedostatek byl vyřešen přesunem této nabídky na stejné místo, kde leží i nabídka výběru prvků v běžném režimu (tj. řešení obvodu). Mezi oběma typy nabídky se přepíná tlačítkem umístěným nahoře. Dále je nabídka se všemi prvky podbarvena lehce odlišně (jinak by nemuselo být zřejmé, jaká nabídka je aktuálně zvolena).

[2] Existují různé překlady tohoto spojení, narazil např. na „uživatelský prožitek“, „uživatelský zážitek“, „zkušenost uživatele s produktem“.

- Předpoklad, že si uživatel zapamatuje, které prvky přidal do obvodu a jsou tzv. odemčené (a tedy upravitelné), byl mylný. Stávalo se, že se hráč snažil přesouvat i prvky uzamčené. Proto jsem se rozhodl, že uzamčené prvky barevně odliším, po úpravě jsou mírně zabarveny do modra, což by mělo zlepšit použitelnost.
- Původně, když byly herní prvky uzamčené k modifikaci uživatelem, tak neumožňovaly zobrazení nápovědy. Bylo to řešení jednoduché na implementaci. Později bylo toto chování přepracováno tak, aby lépe odráželo očekávání uživatelů – tj. aby bylo možné zobrazit nápovědu u libovolného prvku, tj. i u prvků, které jsou uzamčeny na pracovní ploše.

C.R.A.P.

Jako další pomůcka při návrhu a zdokonalování uživatelského rozhraní bylo aplikování pravidel „C.R.A.P.“[46] (případně „CRAP“; někdy, méně často, také označováno jako „CARP“).

Písmeno C reprezentuje kontrast (z angl. „contrast“). Princip říká, že důležité, výchozí či více používané prvky mají mít vyšší kontrast něž prvky méně důležité nebo skoro nepoužívané.

Znak R zastupuje slovo „opakování“ (z angl. „repetition“), můžeme si pod ním představit konzistenci uživatelského rozhraní. Když uživatel narazí na prvek, který vypadá a chová se podobně jako prvek, který uživatel už dříve použil, tak je pro uživatele velmi jednoduché odvodit, jak s tímto prvkem zacházet. Takové uživatelské rozhraní (a potažmo aplikaci) pak vnímá jako přehledné, snadné a kvalitní.

„Zarovnání“ vystupuje v akronymu pod A (z angl. „alignment“). Jde o radu, která zřejmě nejvíce ovlivní vzhled rozhraní. Formulář, okno či dialog, který následuje tento princip, působí přehledněji a jednodušeji na vyplnění (tj. použití), než formulář obsahující stejné položky, které ale nejsou zarovnané (běžně na mřížku).

Poslední je písmeno P, které označuje „blízkost“ (z angl. „proximity“). Princip říká, že podobné nebo související prvky mají ležet blízko sebe. Člověk si totiž podvědomě vytváří spojení mezi objekty na základě vzdálenosti (ne výlučně, ale vzdálenost hraje důležitou roli). Praktická ukázka tohoto principu může být menu s výrobky často situované v levé části webové stránky. Pokud menu rozdělíme na více částí, např. elektronické výrobky rozdělíme na audio zařízení, video zařízení a počítače, dosáhneme podstatně lepší použitelnosti – uživatel se rychle zorientuje a nalezne požadovaný odkaz.

Je poměrně zajímavé, že tzv. zdánlivá použitelnost, tj. jak na nás na první pohled rozhraní působí (esteticky, grafickým uspořádáním), hraje velkou roli při hodnocení celkové použitelnosti [23].

7.2 Použité vývojové nástroje

Před, v průběhu a po implementaci aplikace jsem použil následující nástroje.

7.2.1 IntelliJ IDEA

Produkt z dílny české firmy JetBrains, integrované vývojové prostředí s velmi dobrou podporou jazyka Scala, ale i dalších méně běžných, nicméně funkcemi velmi zajímavých, JVM jazyků jakými jsou např. Kotlin či Groovy [19]. V edici „community“ je IntelliJ IDEA dostupná zdarma a osobně ji preferuji před alternativním Scala IDE založeném na Eclipse.

Důvodem této preference je především mnohem užitečnější našeptávání – nejen podle prefixu, ale i podle „Camel“ nebo jen části identifikátoru. IntelliJ IDEA často bere v potaz typ výrazu či proměnné a nabízí vhodnější volbu výše v seznamu nabízených možností.

Přestože z hlediska funkcí a rozsahu podpory jazyka Scala je IntelliJ IDEA pravděpodobně nejvhodnější IDE na trhu pro vývoj v tomto jazyce, narazil jsem na několik menších problémů, které podrobněji přiblížím v sekci 7.4.

7.2.2 Git

Git je známý Open Source verzovací systém. Je distribuovaný, takže na rozdíl od např. SVN není třeba serverová část. Disponuje i poměrně pokročilými funkcemi, jakými je kupř. podpora více větví.

Při vývoji jsem verzoval jak kód, tak i tento text spolu s dalšími pomocnými soubory – různé poznámky, obrázky, atp. Repozitář jsem udržoval i na vzdáleném serveru pro případ, kdy by došlo k poškození dat na pracovním PC.

7.2.3 Inkscape

Vektorový grafický editor vydávaný pod Open Source licencí. Jako nativní formát používá SVG, grafiku lze jednoduše vyexportovat i do formátu PDF, který se používá pro obrázky v tomto celém textu.

Vyjma GUI prvků byla veškerá grafika pro hru vytvořena v tomto programu. Párkrát jsem zažil pád programu, jde o nějakou chybu spojenou s klonovaným objektem typu group a aplikací transformace, nešlo ale o nic neřešitelného. Byl jsem příjemně překvapen kvalitou tohoto zdarma dostupného produktu.

7.2.4 Atom

Na menší úpravy textových souborů, např. poznámek, jsem využíval Open Source editor Atom. Podporuje velké množství formátů, umí základní našeptávání, komunita vytvořila nepřeberné množství modulů, pomocí kterých lze přidávat do editoru novou funkcionalitu, případně měnit stávající.

V minulosti jsem krátce používal Sublime Text, ale protože jde o placený produkt (s časově omezenou testovací verzí), přešel jsem na Atom. Ten je o něco pomalejší, především doba prvního načtení aplikace, ale vývoj rychle pokračuje a v každé nové verzi je vidět velké množství změn.

7.2.5 Dia

Open Source editor diagramů Dia mi pomohl při návrhu a v první fázi implementace. Přibližně polovina diagramů v tomto textu pochází z programu Dia – dobrou ukázkou je návrh balíku `graphicalCircuit` v příloze C.

Nepodporuje pouze diagramy tříd, ale mnoho dalších. Příkladem budiž diagram návrhu obrazovek na obrázku 6.2 a digram balíků – obr. 6.1.

7.2.6 Dia2Scala

„Dia2Scala“ – nástroj určený k vygenerování kódu z diagramu tříd ve formátu editoru Dia (XML), který vznikl jako vedlejší produkt této práce. Velmi usnadnil počáteční fázi implementace. Více si o něm můžete přečíst v příloze D.

7.2.7 Další software

Při vývoji jsem použil ještě několik dalších aplikací, velmi stručně je popíši v této podsekci. Uvádím je v této společné podsekci především proto, že jsem jejich užíváním nestrávil příliš mnoho času, šlo např. o jednorázové převody z jednoho formátu do jiného.

Audacity Open Source editor zvukových souborů.

foobar2000 Přehrávač hudby, umí i převádět mezi různými formáty, freeware.

Hiero Utilita k exportu fontů z vektorového formátu (např. ttf) do bitmapového formátu, který lze použít k rychlému vykreslování.

ImageMagick Sada konzolových nástrojů pro manipulaci s obrazovými daty a formáty.

MSYS GNU nástroje pro Windows.

LibreOffice Balík kancelářských nástrojů pod Open Source licencí. Byl použit na vytvoření PDF formuláře – dotazníku a k vyhodnocení získaných dat.

Adobe Acrobat Reader Proprietární PDF prohlížeč s dobrou podporou formulářů.

SumatraPDF Open Source (GPL3) PDF prohlížeč, u kterého jsem využil především funkce sledování změn a automatického překreslení PDF dokumentu.

TeXnicCenter Svobodný editor souborů TEX.

XnView Proprietární prohlížeč obrázků, který je zdarma dostupný pro osobní použití a pro vzdělávací a neziskové účely.

7.3 Knihovny

Tato sekce přiblíží knihovny, které byly použity k vytvoření hry.

7.3.1 libGDX

Jde o multiplatformní Open Source framework pro tvorbu (především) 2D her napsaný v jazyce Java. Některé obecné informace o libGDX lze nalézt v předchozích částech práce – podsekce 4.3.4 a 6.2.2.

Při tvorbě aplikace jsem využil atlasu textur (jeho cílem je urychlit vykreslování, obecně slouží k seskupování textur, které se vykreslují po sobě – např. prvky uživatelského rozhraní).

Nejvíce jsem interagoval se „scene2d“ částí. Jde o abstrakci implementující graf scény a jeho jednotlivé prvky – např. ovládací prvky, okna, atp. Konkrétně kupř. objekt robota dědí od scene2d třídy „Actor“, tedy umí se vykreslit na daných souřadnicích a umí se aktualizovat.

Další poměrně důležitá část libGDX je fyzikální knihovna „Box2D“. Někdo by mohl namítnout, že nebylo nutné ji využívat v tomto konkrétním případě, ve hře se totiž nachází pouze obdélníková tělesa. Přesto mi pomohla s několika menšími úkoly (výpočet kolizí, gravitace, pohyb robota). Myslím si, že to byla správná volba z hlediska potencionálních rozšíření – zmiňme třeba jednoduché přidání neobdélníkových těles, která mohou sloužit k simulaci plošin či schodů.

7.3.2 Spray JSON

Odlehčená knihovna sloužící k serializaci (a deserializaci) do formátu JSON (JavaScript Object Notation) v jazyce Scala. Podporuje práci s „case class“ a transparentně pracuje se všemi základními datovými typy a veškerými kolekcemi.

Knihovna libGDX obsahuje třídy pracující s formátem JSON, bohužel přímo je podporována pouze Java. Pokud chce člověk serializovat Scala kolekce či hojně používané „case class“, tak tvrdě narazí. Z tohoto důvodu byla použita knihovna Spray JSON pro práci s formátem JSON.

V samotné aplikaci se JSON používá pro ukládání úrovní a nastavení programu (kupř. jazyk a hlasitost).

7.3.3 ScalaTest

Známá knihovna ScalaTest poskytuje nástroje pro psaní automatických testů. Je použita především pro testování implementace obvodu a diskrétní simulace.

7.3.4 jOOR

Její jméno pochází ze spojení „Java Object Oriented Reflection“. Je to malá, ale velmi užitečná knihovna, která obaluje funkce související s reflexí. Tyto funkce jsou značně neintuitivní, operace které poskytují nelze zapsat výstižně. Popsané problémy adresuje knihovna jOOR.

7.4 Problémy při vývoji

Velmi stručně vyjmenuji problémy, které se vyskytly v průběhu vývoje aplikace. Šlo především o malé nedokonalosti a lehce řešitelné záležitosti.

7.4.1 IntelliJ IDEA

Toto IDE, přestože velmi dobré, nepodporuje některé nejpokročilejší vlastnosti jazyka Scala. Mluvím o tzv. makrech, jsou především používána knihovnami a i pokročilým uživatelům jazyka činí nemalé potíže.

Makra slouží jako další vrstva při kompilaci, umožňují upravovat kompilátorem vygenerovaný mezikód. Pomocí nich lze dosáhnout až neuvěřitelných výsledků, např. knihovna Shapeless rozšiřuje generické možnosti jazyka (polymorphic function values, heterogenní listy, abstrakce nad aritami a mnoho dalšího)[43].

Konkrétní případ potíží je špatná detekce chyb v souborech, kde se využívá knihovna Spray JSON. Řešením bylo třídy rozdělit do více konstruktů trait a tím omezit šíření hlášení falešných chyb pouze na metody, které pracují přímo s knihovnou.

7.4.2 libGDX

Framework libGDX obsahuje většinu věcí, které průměrná mobilní hra potřebuje. Ale jako v každém softwaru i v tomto jsou nějaké menší problémy.

Při psaní kódu pro polo-průhledný obdélník, který znázorňuje přesouvaný prvek obvodu, jsem narazil na problém s vykreslováním geometrických tvarů s průhledností pomocí „ShapeRenderer“. Na desktopu vše fungovalo podle očekávání na poprvé, bohužel na tabletu s Androidem se buďto neaplikovala průhlednost, nebo (při pokusu o manuální zapnutí „blending“ vlastnosti v OpenGL) se nevykreslovalo nic. Problém byl nakonec vyřešen použitím Scene2D

modulu – třída `Image` a jednobarevný obrázek o velikosti jednoho pixelu roztažený a obarvený tak, aby odpovídal polo-průhlednému šedému obdélníku.

Vykreslování znaků fontu trpí zaokrouhlovacími chybami. Bylo nutné „sáhnout“ dovnitř objektu popisujícího font a zmenšit problematické znaky o velmi malé číslo (v kódu $0,001$f).

Posledním zádrhelem s libGDX byl popis vzhledu GUI elementů. Příklady uváděly nevalidní (ne-striktní) JSON, chyběly uvozovky u klíčů i řetězců. To způsobovalo problémy při používání ostatních nástrojů, např. IDE, které nevalidní JSON (zcela pochopitelně) neumělo naformátovat.

7.4.3 ProGuard

Z důvodu omezení DalvikVM bylo nutné využívat program ProGuard. Slouží k optimalizacím generovaného kódu, toho dosahuje především vypouštěním nepoužitých metod a tříd a přejmenováváním často používaných identifikátorů.

Naneštěstí přímo nepodporuje ani Android ani základní knihovnu jazyka Scala. Bylo tedy nutné přidat poměrně velké množství výjimek a ty v průběhu vývoje rozšiřovat, protože se často stávalo, že ProGuard zahodil např. popis rozhraní či atributy třídy, ke kterým se přistupovalo pouze reflexí.

Konkrétním příkladem budiž odstraňování metody `hashCode` z rozhraní `immutable.Map`.

7.5 Popis aplikace

7.5.1 Herní obsah a dostupnost informací

Prototyp obsahuje deset úrovní. Každá úroveň představuje hráči nějaký nový koncept, většinou jde o hradlo či klopný obvod. Po vstupu do úrovně je postupně zobrazován text, který popisuje a vysvětluje, co je v dané úrovni nového a co se musí provést. Tento text může hráč kdykoliv znovu vyvolat, nehrozí tedy riziko neúmyslného zavření a následné frustrace, která je způsobená nedostatkem informací o fungování neznámého nového prvku.

Nápověda využívá dlaždice na herní ploše typu „monitor“. Při odkazu v textu na monitor se nad něj přesune kamera a monitor se rozsvítí. Hráč tedy zřetelně vidí, o které konkrétní části obvodu se píše v průvodním textu.

Je také pamatováno na možnost, že hráč zapomene, jak funguje určitý dříve představený prvek a není tedy k dispozici popis v průvodním textu aktuální úrovně. Situace je řešena tak, že u každého prvku je možné jednoduše vyvolat nápovědu se stručným popisem jeho chování.

7.5.2 Grafika

Skoro všechna grafika je vlastní tvorbou (vyjma předlohy GUI prvků, loga a fontu). Jak bylo uvedeno výše v podsekci 7.2.3, obrázky byly tvořeny v editoru Inkscape.

Kromě textur prvků na hrací ploše a robota, byly nakresleny také jednoduché ikony pro různá tlačítka (např. nový soubor, otevření souboru, opuštění úrovně, nastavení).

Každé hradlo a klopný obvod nesou unikátní symbol. Pro snadnější orientaci jsou vstupy a výstupy označeny malými šipkami. Pokud má prvek více rozdílných vstupů, např. RS obvod, pak se nachází jednoznakové označení u všech vstupů. Dále prototyp obsahuje dva typu terénu, startovní a cílovou zónu, tlakový senzor a elektrickou překážku.

Hra umožňuje režim s nízkým rozlišením textur, který je vhodný pro hardwarově slabší tablety a mobilní zařízení. Na desktopu se doporučuje použít textury s vyšším rozlišením. Volba lepších textur v nastavení je automaticky vybrána při prvním spuštění na počítači.

7.5.3 Hudba a zvuk

Byla použita volně dostupná hudba pod licencí „Creative Commons Attribution“. Hra obsahuje několik zvukových efektů, které jsou také pod licencemi umožňující použití zdarma i v komerčních produktech. Všichni autoři zvukových (a ostatních) děl jsou samozřejmě uvedeni přímo v aplikaci v obrazovce „O hře“.

7.5.4 Jazyk

Program využívá pro drtivou většinu řetězců modul „i18n“ z knihovny libGDX. Uživatelské rozhraní je přeloženo jak do anglického, tak do českého jazyka. V aktuálním stavu úrovně obsahují pouze český překlad. Formát úrovní ale podporuje více jazyků, takže v budoucnu bude možné vydat i anglickou mutaci hry.

7.5.5 Implementovaná hradla a klopné obvody

Výuková hra obsahuje hradla NOT, AND, OR a XOR. Z klopných obvodů jsou přítomny RS, JK, toggle a data. Dále jsou dostupné prvky zdroj signálu true a false, časovač (dvě verze, jedna s volitelným nastavením prodlevy) a identita (dva typy s různým zpožděním).

7.5.6 Nové úrovně

Tvoření nových úrovní je jednoduché, stačí v obrazovce výběru úrovně aktivovat tlačítko s obrázkem prázdného listu papíru. Nově vytvořená úroveň je v tzv. creative módu, ve kterém je možné přidávat libovolné prvky na hrací plochu a do oblasti inventáře. Všechny vestavěné úrovně byly tvořeny touto cestou. Můžeme tedy říci, že program obsahuje plnohodnotný editor úrovní.

7.5.7 Podporované platformy

Ze zadání práce vyplývá, že mezi podporovanými platformami nesmí chybět Android. Hra byla otestována a shledána funkční na Androidu verze 2.3.4 a 4.4.4.

Herní aplikace je také testovaná na stolním počítači s Windows 7 (Java Platform verze 8.31 64b) a Lubuntu 14.04.2 LTS (OpenJDK RE IcedTea 2.5.5 64b, Java verze 1.7.0_79).

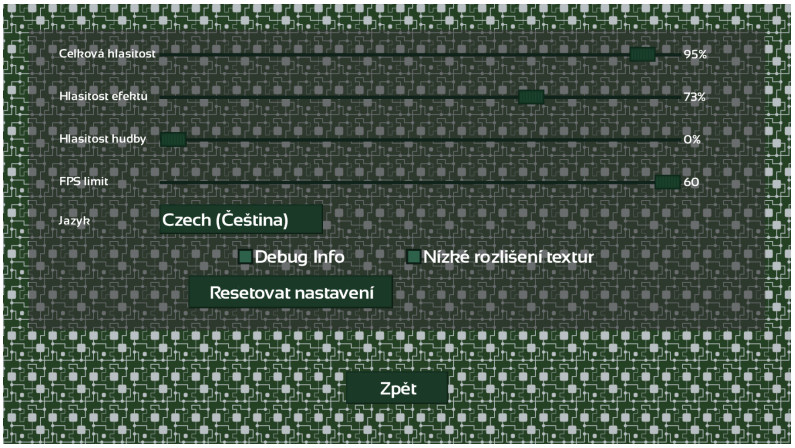
Jak bylo zmíněno dříve v podsekci 4.3.4, libGDX podporuje i iOS a několik dalších platforem. Neměl by tedy být větší problém v budoucnu vydat i verze pro další platformy.

7.5.8 Možnosti nastavení

V obrazovce nastavení je možné měnit nezávisle hlasitost hudby, zvukových efektů a celkovou hlasitost. Jak bylo uvedeno výše, tak lze povolit či zakázat režim s vyšším rozlišením textur. Lze také nastavit maximální možný počet snímků za vteřinu (běžně označováno jako FPS limit). To může snížit spotřebu zařízení, případně i stabilizovat prodlevy mezi vykreslovanými snímky (např. na méně výkonných mobilních zařízeních).

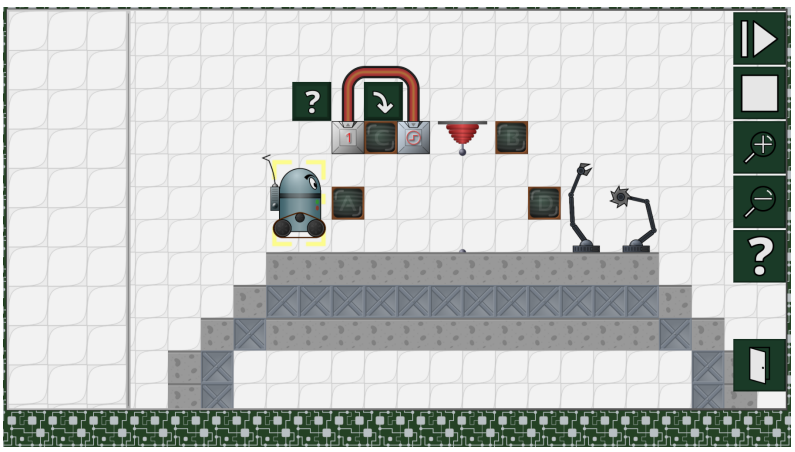
7.5.9 Ukázky

Následují snímky ze hry. Obrázek 7.1 demonstruje obrazovku nastavení popsanou v 7.5.8. Další obrázek 7.2 ukazuje výběr úrovně, vpravo dole vidíme tlačítka pro otevření uživatelem vytvořené úrovně a vytvoření nové uživatelské úrovně.



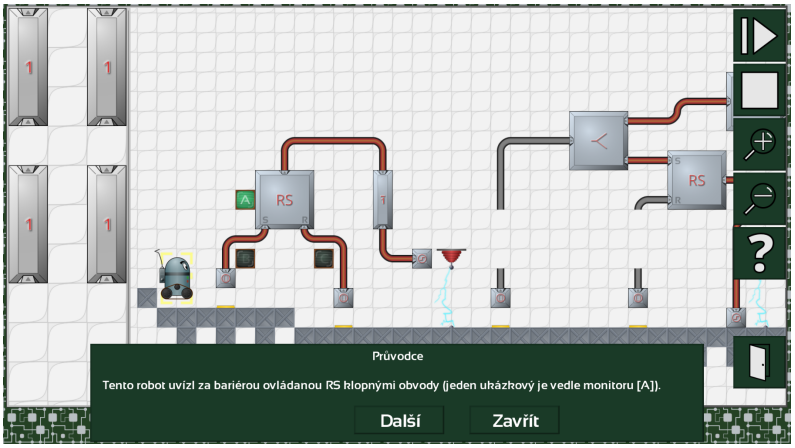
Obrázek 7.1: Obrazovka nastavení

Obrázek 7.2: Výběr úrovně

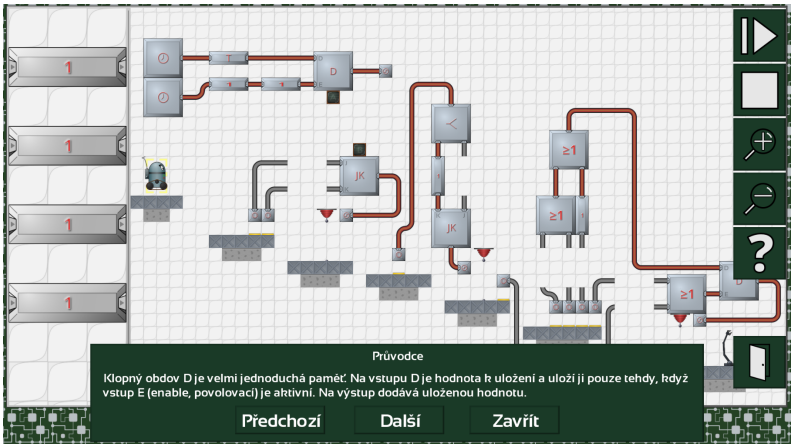


Obrázek 7.3: Úroveň – základní koncepty hry

Zbylé snímky (7.3, 7.4 a 7.5) zachycují konkrétní úrovně. Vlevo se nachází inventář, uprostřed herní plocha, napravo menu a dole průvodní text.



Obrázek 7.4: Úroveň – RS klopný obvod, dlaždice typu monitor



Obrázek 7.5: Úroveň – JK a D obvody

7.6 Testování

Sekce popisuje přístup k testování při práci na implementaci aplikace.

7.6.1 Simulační vrstva a serializace dat

Třídy, které reprezentují obvod (statickou i dynamickou část), a implementace kalendáře událostí jsou z velké části pokryty jednotkovými testy. Stejně tak jsou testovány i serializace a deserializace prvku, obvodu, grafického obvodu a úrovně.

7.6.2 Testování funkčnosti

Ze začátku vývoje byly pro testování používány především jednotkové testy. Ve fázi, kdy se započalo s implementací grafického uživatelského rozhraní, jsem začal testovat inkrementálně, vždy po přidání nějakého celku. Díky tomu, že knihovna libGDX podporuje kromě Androidu také desktop, jsem byl schopen velmi jednoduše zkoušet každou změnu uživatelského rozhraní.

Vždy jednou za čas, většinou po dokončení určité větší části, jsem aplikaci vyzkoušel v emulátoru s OS Android 4.4 a také na reálném zařízení – postarším tabletu s OS Android verze 2.3.

Kapitola 8

Zhodnocení aplikace

Kapitola ukáže, jak na aplikaci reagovali běžní uživatelé. Také vyjmenuje, přiblíží a zdůvodní možná a zvažovaná rozšíření.

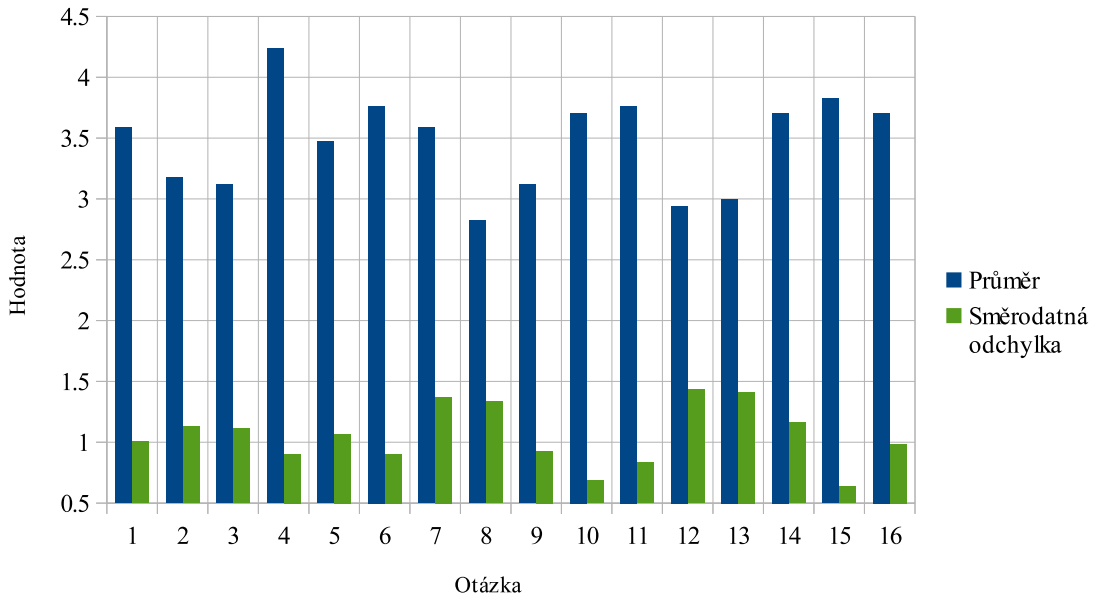
8.1 Hodnocení uživateli

Hodnocení mladými uživateli probíhalo formou dotazníku. Celkový počet odevzdaných formulářů dosáhl čísla 17. První část se skládala z běžně používaného dotazníku pro hodnocení uživatelské přívětivosti [6], druhá část se dotazovala na otázky specifické pro aplikaci (týkají se např. nápovědy a grafiky). Použité otázky můžete nalézt v příloze E. Pokud se zajímáte o detailnější pohled na data získaná z dotazníků, nalistujte si přílohu F.

Na všechny otázky se odpovídalo číslem na stupnici 1–5, kde 1 znamená „naprosto nesouhlasím“ a 5 „zcela souhlasím“. Data prezentovaná dále jsou přepočtena na skóre (tj. hodnota negativních otázek je invertována). Čím větší je hodnota skóre, tím pozitivnější je hodnocení.

8.1.1 Přehled

Graf na obr. 8.1 ukazuje průměrné hodnocení a směrodatnou odchylku hodnocení podle otázky.



Obrázek 8.1: Průměr a směrodatná odchylka hodnocení otázek

Nejhůře hodnocená je otázka číslo 8, která říká, že hráč shledává ovládání nešikovným. Průměrná hodnota je 2,8 (z intervalu 1–5), což je poměrně vysoké skóre. 47 % hodnotilo negativně

(na stupnici zvolili 1 nebo 2), 18 % neutrálně (3) a 35 % pozitivně (4 nebo 5, „pozitivně“ ve významu, že nesouhlasí s výrokem o špatném ovládání).

Nejlépe dotazovaní hodnotili otázku 4 – nesouhlas s tím, že by potřebovali odbornou pomoc při používání aplikace. Jak vidíme, tak nejvíce se účastníci shodli na hodnocení otázek 15 a 10. Otázka 15 řeší, zda si uživatel myslí, že by aplikace mohla někoho povzbudit k zájmu o technický obor. Desátý výrok říká, že se uživatel nepotřeboval mnoho naučit k úspěšnému ovládání hry.

8.1.2 Grafika a estetika

Na otázku, zda shledávají grafické a estetické zpracování hry dostatečné, odpovědělo kladně 65 %, neutrálně 12 % a negativně 23 %. Otázka má skóre o velikosti 3,7. Z výsledků můžeme vyvodit, že tři čtvrtiny uživatelů je spokojeno se stávajícím (poměrně jednoduchým) grafickým zpracováním a jedna čtvrtina se domnívá, že by grafická stránka měla být předělána nebo vylepšena.

8.1.3 Efektivita aplikace

S vyjádřením, že si uživatel myslí, že aplikace by mohla někoho povzbudit k zájmu o technický obor, se vyjádřilo kladně (ohodnotilo číslem 4 nebo 5) 71 % dotazovaných. Neutrální postoj, tj. volbou ohodnocení 3, zastává 29 % zúčastněných. Nikdo nevybral horší hodnocení než 3. Celkové skóre otázky je 3,8. Řekl bych, že uživatelé zastávají kladný názor na efektivitu dopadu hry.

8.1.4 Nasazení na školách

S nasazením na základní škole jako doplňkového nástroje pro volbu zaměření budoucího studia souhlasí 59 %, neutrálních je 29 % a proti hlasovalo 12 %. Hodnocení otázky je ve výsledku přibližně 3,7 bodů. Nadpoloviční většina dotázaných tedy souhlasí s nápadem použití aplikace na základních školách.

8.1.5 Nápověda

K průvodnímu textu v dolní části obrazovky se kladně vyjádřilo 53 %, neutrálně 47 %. Přestože je hodnocení poměrně vysoké, 3,76, některé texty k úrovním byly přepracovány, aby lépe vysvětlili zadané úkoly (kupř. hodně hráčů mělo problém s natáčením prvku a správnou orientací portů).

Je zajímavé, že 29 % uživatelů nepoužilo nikdy nápovědu pro konkrétní prvek. Ti, co ji použili, se na hodnocení neshodli. 42 % hodnotí kladně, 42 % negativně a 16 % ani dobře ani špatně. Je zřejmé, že texty nápovědy pro konkrétní prvek se budou muset zrevidovat.

8.1.6 SUS

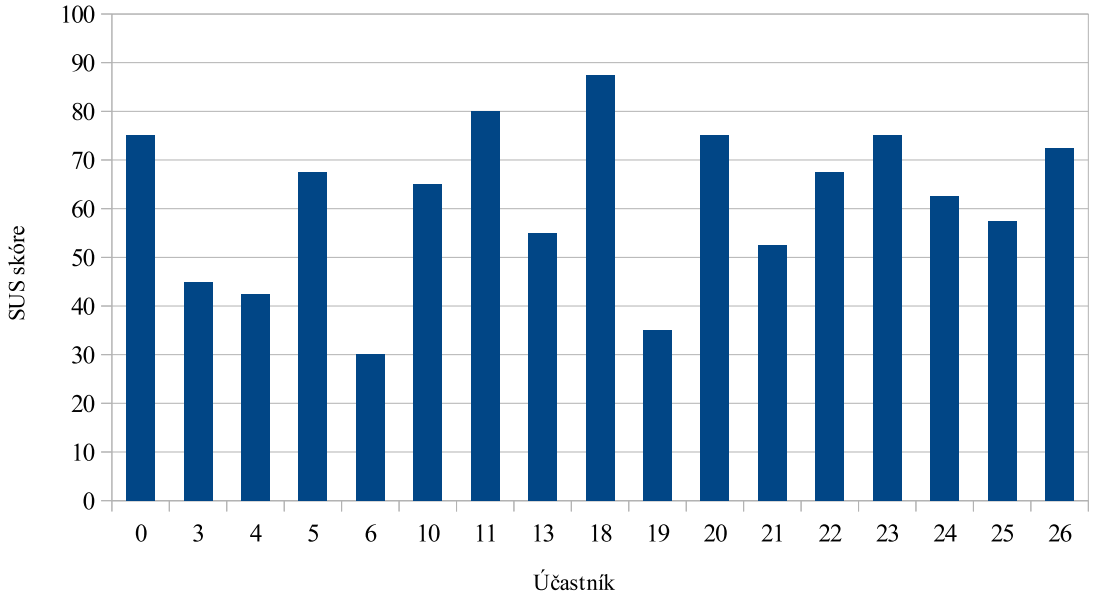
Dostáváme se k hodnocení SUS, které nabývá hodnot 0–100 (více je lépe). Čtenář si může prohlédnout na obrázku 8.2 vypočtené SUS skóre podle jednotlivých uživatelů[1].

Průměrné skóre je 61,5 a medián dosahuje hodnoty 65. Povšimněme si, že rozdíly mezi hodnoceními jednotlivých uživatelů jsou velmi velké (směrodatná odchylka je 16,3).

8.1.7 Shrnutí uživatelského hodnocení

Z předcházejícího textu můžeme říci, že aplikace byla uživateli hodnocena vesměs pozitivně.

[1] Spodní osa nese identifikační číslo. Je z ní patrné, že velké množství registrovaných neodevzdalo dotazník.



Obrázek 8.2: Skóre uživatelské přívětivosti (SUS)

Z dotazníků vyplývá, že některé stránky by mohly být ještě vylepšeny, především grafická podoba a nápověda pro jednotlivé prvky. Naopak hlavní zaměření aplikace – motivovat, představit, zaujmout – hodnotí kladně. Nápověda v úrovních je vnímána jako dostačující až dobrá, s případným představením žákům základních škol souhlasí a názor na motivační vliv aplikace mají také pozitivní.

SUS skóre je spíše průměrné, ale ne vyloženě špatné. Nabízí se otázka, zda aplikace tohoto typu může vůbec dosahovat nejvyšších příček. Jako problém vidím fakt, že pouze u relativně malé části uživatelů hra uspěje. Pouze u lidí, kteří mají určité předpoklady pro logické myšlení. A to není sám o sobě problém, s tímto úmyslem byla přece vytvořena. Bohužel to ale znamená, že v jakémkoliv měření uživatelské spokojenosti (UX) bude dosahovat horších hodnocení, protože uživatelé budou zaměňovat pojmy „naučit se ovládat aplikaci“ a „zvládnout látku vysvětlovanou aplikací“. Např. tato tvrzení z SUS jsou problematická kvůli záměně z roviny ovládání do roviny pochopení: „Shledávám hru zbytečně složitou.“ a „Při hraní této hry jsem si byl jistý tím, co dělám.“. Musíme tedy brát výsledky měření výukové aplikace z jakéhokoliv obecného testu uživatelské spokojenosti s rezervou.

8.2 Zvažovaná rozšíření

Prvním je vylepšení formátu, ve kterém se ukládají obvody. Používá se JSON a poměrně dost místa na disku zabírají pojmenování vlastností. Vylepšení spočívá v použití kompresního algoritmu. Experimentoval jsem s algoritmem gzip a výsledný soubor byl často i desetkrát menší.

Další možné rozšíření jsou různá vylepšení vodičů – např. podpora více barev, křížení, pohodlnější úpravy.

Implementace dalších typů terénu a překážek, především nesmrtících, by zvýšila rozmanitost výběru pro tvorbu úrovní a zcela jistě tak zvýšila pocit unikátnosti nových místností.

Je plánované částečné nebo úplné přepracování grafiky. Také přidání více zvukových efektů

a rozšíření výběru hudby na pozadí by hru posunulo dále.

Chování robota by mohlo být upraveno tak, aby se při nárazu na překážku otočil a pokračoval dál v jízdě na opačnou stranu. To, ve spojení s novými typy nesmrtících překážek, by mohlo otevřít zajímavé možnosti pro rozložení úrovní.

Ke zlepšení uživatelského komfortu by mohlo přispět i automatické natočení prvku do vhodné pozice po přetažení na hrací plochu.

Problémy nevhodného natočení, tj. pokusu o připojení výstupního portu ke kabelu, který je už připojen k jinému výstupu, (případně vstup připojovat ke vstupu jiného prvku) by mohly být řešeny lépe. Jednou z možností je zobrazení dialogu, případně méně rušivé řešení – zvýraznit port, který nelze ke kabelu připojit, např. červenou kružnicí či jej zabarvit do červena.

Stávající verze hry má kompletní jazykovou podporu pouze pro češtinu. Kód aplikace podporuje více jazyků, je ale třeba provést překlad. Především překlad textů úrovní a nápovědy pro jednotlivé prvky bude poměrně náročný, především kvůli svému rozsahu. Jazykem, pro který by se měla přidat podpora, by zcela jistě měla být angličtina.

Někteří hráči se vyjádřili, že by uvítali inventář na spodní straně obrazovky, jiní by jej preferovali napravo. Řešením by mohlo být přidání dalších profilů rozložení ovládacích prvků na herní obrazovce a možnost změny buďto přímo z herní obrazovky, nebo např. z obrazovky nastavení.

Kapitola 9

Závěr

Práce se zabývá motivační výukovou puzzle hrou, která má představit pozvolnou a názornou formou úplné základy logických systémů. Cílem bylo vytvořit takovou hru, která by případně mohla povzbudit zájem o techniku (ať už pouze jako koníček nebo jako budoucí studijní obor).

Byly popsány základy logických systémů, nejvíce používaná hradla a klopné obvody a způsob simulace logických systémů. Stručně byly představeny hry, počítačové hry a jejich žánry, speciální důraz byl kladen na puzzle hry. Různé techniky a přístupy byly předvedeny na puzzle hrách, které se svým úspěchem navždy zapsaly do herní historie. Byly prostudovány různé mobilní a počítačové platformy včetně jejich stručné historie. Následuje popis známých multiplatformních knihoven.

Analýza několika vybraných aplikací, které se zabývají hradly, klopnými obvody či logickými systémy, přinesla zjištění, že na trhu se nachází pouze malé množství kvalitnějších aplikací, které se na tuto oblast zaměřují. Většina aplikací, které se zabývají výukou logických systémů nebo jejich částí, nejsou příliš lákavé a často vůbec nevyužívají potenciálu média.

Následně jsou specifikovány požadavky na výslednou hru, zvoleny vhodné implementační prostředky a je proveden návrh. Protože cílová platforma je Android, byla zvolena jako hlavní programovací jazyk Scala. Jde o jazyk nad JVM kompatibilní s jazykem Java, oproti ní je ale mnohem výstižnější a stručnější, podporuje velké množství novinek, především z funkcionálního světa. Jako hlavní knihovna byla zvolena libGDX, která je multiplatformní (především to je podpora pro Android, iOS, Windows, Mac i Linux), má velkou komunitu a poskytuje vše potřebné pro vytvoření 2D hry.

Další část práce se zabývá implementací – popisem užitých principů a přístupů, představením vývojových nástrojů a knihoven, stručným nastíněním problémů a nakonec popisem aplikace. Vytvořil jsem originální grafiku pro všechny herní objekty (např. robot, hradla, terén, klopné obvody, vodiče) a pro část prvků grafického uživatelského rozhraní (kupř. ikonky na tlačítkách či textura pozadí). Výsledná hra uvádí hráče do světa logických systémů tak, jak bylo specifikováno v kapitole 6.

Aplikace byla průběžně testována, pro nižší vrstvy existují jednotkové testy. Hra byla vyzkoušena a shledána funkční na tabletu s operačním systémem Android 2.3.4, dále byla úspěšně provozována na Androidu 4.4.4, stolním počítači s Windows 7 s JRE 8.31 64b a také na PC s Lubuntu 14.04 s 64b OpenJDK odpovídající verzi JRE 1.7.0_79.

Uživatelé hru hodnotili vesměs pozitivně. Z dotazníků vyplývá, že by dotázaní souhlasili s použitím aplikace ve školách jako doplňujícího nástroje při zvažování budoucího zaměření studia. Jsou také názoru, že hra může podnítit zájem o techniku a nasměrovat tak talentované jedince na správnou cestu.

Bylo uvedeno mnoho možných rozšíření. Některá vycházejí z přání hráčů, např. automatické natočení prvků, nastavitelné rozložení ovládacích elementů na herní obrazovce nebo vylepšení grafiky. Ostatní rozšíření jsou především zvětšením záběru a vypilováním stávajících vlastností, kupř. úspornější formát souborů s úrovněmi, větší rozmanitost hudby a zvukových efektů nebo nové typy překážek, vodičů a terénu.

Literatura

[1] Adams, D.: Portal Review. [online], Naposledy navštíveno 26. 11. 2014.
URL http://www.ign.com/articles/2007/10/09/portal-review

[2] Alexander, A.: Scala idiom: Prefer immutable code (immutable data structures). [online], Naposledy navštíveno 24. 12. 2014.
URL http://alvinalexander.com/scala/scala-idiom-immutable-code-functional-programming-immutability

[3] Bořánek, R.: Android 5.0 Lollipop: šifrování a konec Dalviku. [online], Naposledy navštíveno 29. 11. 2014.
URL http://www.root.cz/clanky/android-5-0-lollipop-sifrovani-a-konec-dalviku/

[4] Brabec, S.: TeX pro každého – stručný úvod do typografie. [online], Naposledy navštíveno 8. 1. 2015.
URL http://www.root.cz/clanky/strucny-uvod-do-typografie/

[5] Brian: PhoneGap, Cordova, and what's in a name? [online], Naposledy navštíveno 5. 12. 2014.
URL http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/

[6] Brooke, J.: SUS - A quick and dirty usability scale. 1986.

[7] Burgess, R.: Latest Steam numbers show Linux, Mac gamers almost equal. [online], Naposledy navštíveno 2. 12. 2014.
URL http://www.techspot.com/news/51834-latest-steam-numbers-show-linux-mac-gamers-almost-equal.html

[8] Choi, K.; Kang, D.: *Modeling and simulation of discrete-event systems*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2013, ISBN 978-1-118-38699-6.

[9] Cocos2d-x. [online], Naposledy navštíveno 30. 11. 2014, wiki.
URL http://www.cocos2d-x.org/wiki/Cocos2d-x

[10] Data o studentech, poprvé zapsaných a absolventech vysokých škol, MŠMT ČR. [online], Naposledy navštíveno 24. 3. 2015.
URL http://www.msmt.cz/vzdelavani/skolstvi-v-cr/statistika-skolstvi/data-o-studentech-poprve-zapsanych-a-absolventech-vysokych

[11] Diggins, C.: The Principles of Good Programming. [online], Naposledy navštíveno 23. 4. 2015.
URL http://www.artima.com/weblogs/viewpost.jsp?thread=331531

[12] eˆcha: Pocket Robots Test Chamber. [online], Naposledy navštíveno 7. 12. 2014.
URL http://echa.ru/gm/prtc/

[13] Eysselt, M.: Logická a funkční schémata, výňatek z oborové normy ONT345553. 2002.

[14] Fischer, P.: Building Cross-Platform Apps with HTML5. [online], Naposledy navštíveno 30. 11. 2014.
URL https://software.intel.com/en-us/html5/articles/building-cross-platform-apps-with-html5

[15] Foldit. [online], Naposledy navštíveno 25. 11. 2014.
URL http://fold.it/portal/

[16] Frištacký, N.; Kolesár, M.; Kolenička, J.; aj.: *Logické systémy*. SNTL Praha, ALFA Bratislava, druhé vydání, 1990.

[17] Hashimi, S. Y.; Komatineni, S.; MacLean, D.: *Pro Android 2*. Apress, 2010, ISBN 13-978-1-4302-2659-8.

[18] Huynh, L.: The History of Puzzle Games. [online], Naposledy navštíveno 15. 11. 2014.
URL http://web.archive.org/web/20100204081152/http://www.gamespot.com/features/vgs/universal/puzzle_hs/

[19] IntelliJ IDEA – The Most Intelligent Java IDE. [online], Naposledy navštíveno 31. 3. 2015.
URL https://www.jetbrains.com/idea/

[20] Introducing Swift. [online], Naposledy navštíveno 29. 11. 2014.
URL https://developer.apple.com/swift/

[21] jMonkeyEngine: jMonkeyEngine 3.0 – Java OpenGL Game Engine. [online], Naposledy navštíveno 4. 12. 2014.
URL http://jmonkeyengine.org

[22] Kolektiv autorů: *Pravidla českého pravopisu*. Academia, 2005, ISBN 80-200-1327-X.

[23] Kurosu, M.; Kashimura, K.: Apparent Usability vs. Inherent Usability Experimental analysis on the determinants of the apparent usability. In *Conference Companion on Human Factors in Computing Systems*, 1995, s. 292–293.

[24] Lehtimäki, J.: Multi-platform Frameworks Destroy Android UX. [online], Naposledy navštíveno 23. 11. 2014.
URL http://www.androiduipatterns.com/2012/03/multi-platform-frameworks-destroy.html

[25] liamdawe: Linux Game Sales Statistics From Multiple Developers. [online], Naposledy navštíveno 2. 12. 2014.
URL http://www.gamingonlinux.com/articles/linux-game-sales-statistics-from-multiple-developers.2963

[26] Šmarda, Z.; Růžičková, I.: Vybrané partie z matematiky. 2004, skripta VUT Brno.

[27] Mark, D.; Nutting, J.; LaMarche, J.: *Beginning iPhone 4 Development: Exploring the iOS SDK*. Apress, 2011, ISBN 978-1-4302-3024-3.

[28] Maso, B.: Příspěvek na fóru ve vláknu „Recommendation for UML class diagram modeling“. [online], Naposledy navštíveno 18. 12. 2014.
URL http://scala-programming-language.1934581.n4.nabble.com/Recommendation-for-UML-class-diagram-modeling-tp1936798p1936799.html

[29] Moore, J.: Revolution OS. [DVD], 2001, film.

[30] Mouse_Master: BUST 'A' MOVE. [online], Naposledy navštíveno 22. 11. 2014.
URL
http://www.neo-geo.com/reviews/neo-reviews/bust-a-move/bust-a-move.html

[31] Nanocrafter. [online], Naposledy navštíveno 25. 11. 2014.
URL http://nanocrafter.org/

[32] Odersky, M.: What is Scala? [online], Naposledy navštíveno 13. 12. 2014.
URL http://www.scala-lang.org/what-is-scala.html

[33] Onyett, C.: Portal 2 Review. [online], Naposledy navštíveno 26. 11. 2014.
URL http://www.ign.com/articles/2011/04/19/portal-2-review

[34] OS X for UNIX Users. July 2011, technology brief.
URL https:
//ssl.apple.com/media/us/osx/2012/docs/OSX_for_UNIX_Users_TB_July2011.pdf

[35] Oyj, D.: Qt | Cross-platform application & UI development framework. [online], Naposledy navštíveno 4. 12. 2014.
URL http://www.qt.io/

[36] Peringer, P.: Modelování a simulace. 2011, slajdy pro předmět IMS, FIT VUT v Brně.

[37] Praetorius, D.: Gamers Decode AIDS Protein That Stumped Researchers For 15 Years In Just 3 Weeks (VIDEO). [online], Naposledy navštíveno 25. 11. 2014.
URL http://www.huffingtonpost.com/2011/09/19/aids-protein-decoded-gamers_n_970113.html

[38] Rollings, A.; Adams, E.: *Andrew Rollings and Ernest Adams on game design*. Indianapolis: New Riders, první vydání, 2003, ISBN 15-927-3001-9.

[39] Ross, S.: *Simulation*. Academic Press, 2002, ISBN 0-12-598053-1.

[40] Rusu, L.: Portal 2 Improves Cognitive Skills more than Lumosity, Study Shows. [online], Naposledy navštíveno 25. 11. 2014.
URL http://www.zmescience.com/medicine/mind-and-brain/portal-2-cognitive-function-01102014/

[41] scriptwelder: Gates of Logic. [online], Naposledy navštíveno 8. 12. 2014.
URL http://www.kongregate.com/games/scriptwelder/gates-of-logic

[42] Seltzer, M.: A Brief History of Operating Systems. 2014, slajdy do předmětu CS 161: Operating Systems.

[43] shapeless: generic programming for Scala. [online], Naposledy navštíveno 31. 3. 2015.
URL https://github.com/milessabin/shapeless

[44] Temmerman, K.: The LOGIC LAB. [online], Naposledy navštíveno 22. 11. 2014.
URL http://www.neuroproductions.be/logic-lab/

[45] Tichý, M.: Elektronika. [online], Naposledy navštíveno 16. 10. 2014.
URL
http://physics.mff.cuni.cz/kfpp/skripta/elektronika/kap9/jednocpoc.html

[46] Travis, D.: A CRAP way to improve usability. [online], Naposledy navštíveno 31. 3. 2015.
URL
http://www.userfocus.co.uk/articles/A_CRAP_way_to_improve_usability.html

[47] Unity Technologies: Unity - Multiplatform. [online], Naposledy navštíveno 4. 12. 2014.
URL http://unity3d.com/unity/multiplatform

[48] Vingron, S.: *Logic Circuit Design*. Springer Heidelberg Dordrecht London New York, 2012, ISBN 978-3642276569.

[49] VonC: Odpověď na otázku „any UML tools for Scala“. [online], Naposledy navštíveno 18. 12. 2014.
URL http://stackoverflow.com/questions/7815983/any-uml-tools-for-scala/7817723#7817723

[50] Ward, K.: LogicBots. [online], Naposledy navštíveno 8. 12. 2014.
URL http://www.incandescentgames.co.uk/logicbots.html

[51] Whitney, L.: Why I Don't Recommend Xamarin for Mobile Development. [online], Naposledy navštíveno 3. 12. 2014.
URL http://www.whitneyland.com/2013/05/why-i-dont-recommend-xamarin-for-mobile-development.html

[52] Xamarin Inc.: Xamarin Platform. [online], Naposledy navštíveno 3. 12. 2014.
URL http://xamarin.com/platform

[53] Zechner, M.: libgdx - Goals and Features. [online], Naposledy navštíveno 30. 11. 2014.
URL http://libgdx.badlogicgames.com/features.html

[54] zester: Looking for tutorial on making 3d games with Qt5.0.x ? [online], Naposledy navštíveno 4. 12. 2014.
URL http://qt-project.org/forums/viewthread/27947/#126036

Příloha A

Obsah přiloženého média

Na přiloženém médiu se nachází následující adresáře a soubory:

/thesis/ text práce ve formátu PDF a LaTeX

/src/ zdrojové kódy aplikace a audiovizuální data nezbytná pro běh aplikace

/bin/ přeložená aplikace pro platformu PC a Android

/readme.txt návod na instalaci a spuštění aplikace

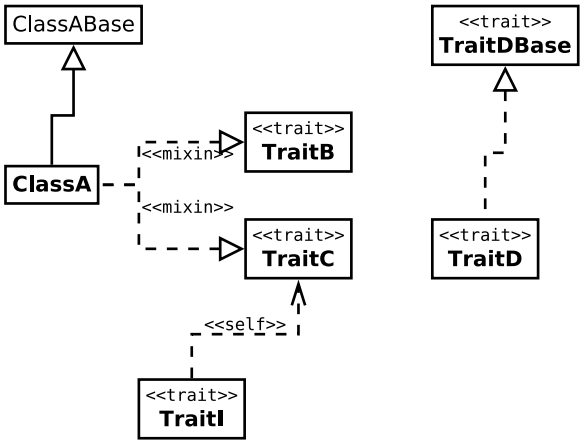
Příloha B

UML a Scala

Jazyk UML je standardem při návrhu, velká část společností jej používá, je to způsob, jak vyjádřit návrh aplikace v grafické podobě pomocí diagramů. Bohužel v sobě nese určité předpoklady a z nich plynoucí omezení. Je například vhodný pro jazyk Java, protože dědění, rozhraní, vlastnosti, metody a další elementy přímo korespondují se zápisem v Javě. Naopak pro jazyk Scala není UML tak vhodné, navíc čím více funkcionálních vlastností ze Scaly používáme, tím více zjišťujeme, kolik toho v UML chybí, nebo dokonce je nepopsatelné [28].

Jak modelovat část vlastností z jazyka Scala je vyřešeno v [49]. Vycházel jsem především z tohoto zdroje, níže se nachází detailnější popis s ukázkami diagramů a odpovídajícího zdrojového kódu.

B.1 Konstrukt trait a dědění



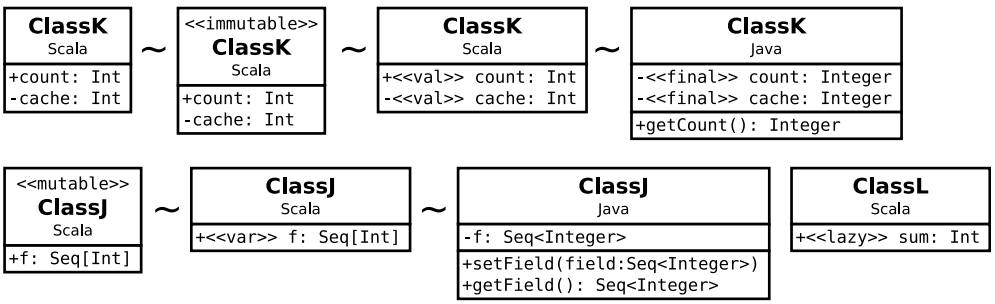
Obrázek B.1: Dědění

Trait se zakresluje jako třída se stereotypem „trait“. Dědění mezi trait konstrukty je značeno stejně jako implementace rozhraní – obr. B.1 vpravo nahoře. V levé části diagramu vidíme tzv. mixin dědění, tj. dědění od jednoho či více konstruktů trait, znázorňuje se podobně jako implementace rozhraní doplněná o stereotyp „mixin“. Ve stejné části vidíme také vztah závislosti se stereotypem „self“, ten říká, že trait `TraitI` má tzv. self type typu `TraitC`.

```
trait TraitB; trait TraitC; trait TraitDBase

abstract class ClassABase
class ClassA extends ClassABase with TraitB with TraitC
trait TraidD extends TraitDBase
trait TraitI { self: TraitC => }
```

Zdrojový kód B.1: Dědičnost

B.2 Var, val, mutable a immutable



Obrázek B.2: Var, val, mutable a immutable vlastnosti

Je důležité přijmout fakt, že Scala preferuje tzv. immutable datové struktury [2]. To jsou struktury, které se po vytvoření nemění. S tímto souvisí užívání `var` a `val`. První je typická proměnná, lze do ní přiřadit opakovaně novou hodnotu. Druhé klíčové slovo popisuje proměnnou, do které nelze znovu přiřadit hodnotu, nelze ji po vytvoření změnit[1].

Není překvapující, že ve Scale je preferováno užívání `val` před `var`, upřednostňuje se funkcionální přístup. Proto v diagramu tříd používám konvenci, že pokud není explicitně řečeno, že jde o mutable (proměnlivou) třídu či proměnnou, tak výchozí stav je immutable (neměnná) třída či proměnná.

Ukázku diagramu tříd s novými stereotypy a ekvivalenty ze světa Javy vidíme na obrázku B.2. Účel stereotypů atributu `val` a `var` je zřejmý, stereotypy tříd `immutable` a `mutable` mají podobnou roli s tím rozdílem, že ovlivňují všechny atributy dané třídy. Přestože je stereotyp třídy `immutable` považovaný za výchozí, lze jej i explicitně u třídy uvést.

Datové třídy jsou typicky implementovány ve Scale jako tzv. case class. Na rozdíl od Javy se často používají anemické třídy (neimplementují žádnou logiku, pouze drží data).

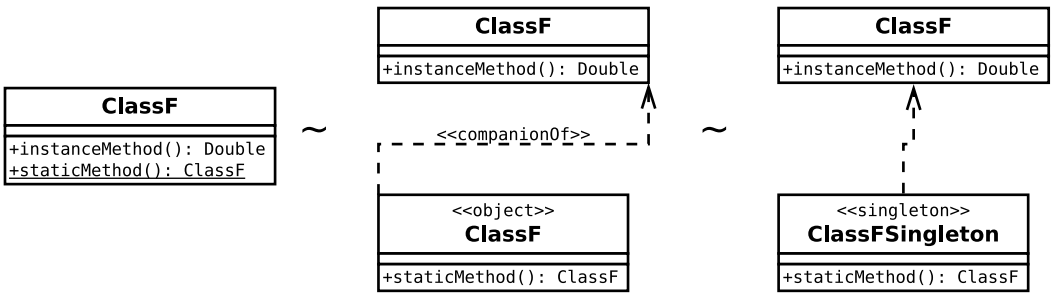
Scala také podporuje tzv. lazy proměnné známé z funkcionálních jazyků, příkladem budiž Haskell. Jde o proměnné typu `val`, `lazy` značí, že výpočet hodnoty probíhá až při prvním použití proměnné při běhu programu. Zápis vidíme v digramu u atributu `sum` třídy `ClassL`.

```
class ClassK(val count: Int, cache: Int)

class ClassJ(var field: Option[Int])

class ClassL { lazy val sum = (1 to 1000000).sum }
```

Zdrojový kód B.2: Ukázky immutable, mutable a lazy vlastností

[1] To ale neznamená, že samotná hodnota není instance mutable třídy.

B.3 Objekty



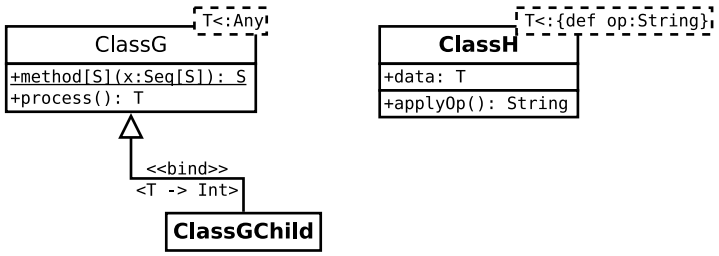
Obrázek B.3: Objekt

Obrázek B.3 uvádí obdobu statické metody v jazyku Java. Ve Scale existuje přímo typ `object`, který je ve své podstatě singleton na úrovni jazyka. Pokud existuje i stejně pojmenovaná třída, pak objekt adresujeme jako companion objekt. Všechny tři zápisy (oddělené tildou) jsou ekvivalentní – první je úsporný, proto jej často používám, druhý je upovídanější a zdůrazňuje konstrukce Scaly ve zdrojovém kódu a poslední je Java verze.

```
class ClassF {
       import ClassF._

       def instanceMethod: Double = 0
}

object ClassF {
       def staticMethod: ClassF = null
}
```

Zdrojový kód B.3: Scala objekt

B.4 Pokročilé typy



Obrázek B.4: Generická třídy a strukturální typ

Pokročilejší datové typy vidíme na obrázku B.4. Vlevo je příklad užití generického typu s horním omezením na `Any`. Metoda `method` předvádí užití generického typu, který je omezen pouze na danou metodu. Ve spodní části obrázku třída `ClassGChild` dědí od `ClassG` a generický typ specifikuje na `Int`.

Napravo vidíme užití tzv. strukturálního typu. Typ je popsán pouze pomocí podoby rozhraní (struktury), ne pomocí jména konkrétního typu. V jiných jazycích se můžeme setkat se stejnou funkcionalitou ale jiným jménem, např. duck-typing. Náš příklad říká, že typ `T` musí mít metodu `op`, která vrací řetězec. Ve zdrojovém kódu níže vidíme, že lze použít i anonymní třídu.

```
abstract class ClassG[T <: Any] {
  def process: T
}

object ClassG {
  def method[S](x: Seq[S]): S = x.head
}

class ClassGChild extends ClassG[Int] {
  def process = 2
}

class ClassH[T <: { def op: String }](val data: T) {
  def applyOp: String = data.op
}

println(ClassG.method(Seq("a", "b")))
println(new ClassGChild().process)
println(new ClassH(new { def op = "Whoo" }).applyOp)
```

Zdrojový kód B.4: Pokročilé datové typy, generické programování

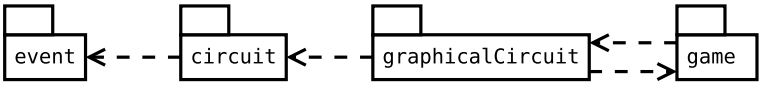
Příloha C

Návrh tříd

Dále uvedené diagramy tříd používají UML rozšířené o stereotypy, které umožňují vyjádřit konstrukce používané v jazyce Scala. Detailní popis této notace může čtenář nalézt v příloze B.

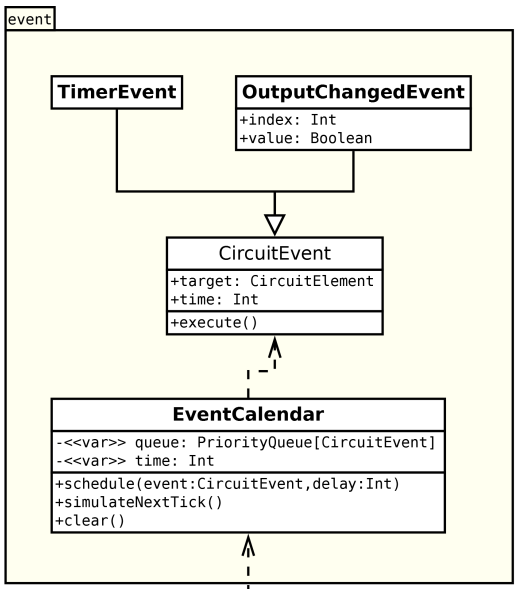
V této části jsou představeny některé důležité třídy. Jde především o třídy související s herní postavou, simulací a zobrazením obvodu. V návrhu je záměrně vynecháno propojení s knihovnou použitou pro vykreslování – počítá se s tím, že tato knihovna bude splňovat principy objektově orientovaného programování, nebude tedy problém přidat reference na textury a jiné objekty do tříd v balíku `graphicalCircuit`.

Obrázek C.1 připomíná čtenáři vazby mezi balíky, které byly poprvé zmíněny v části 6.3.1.



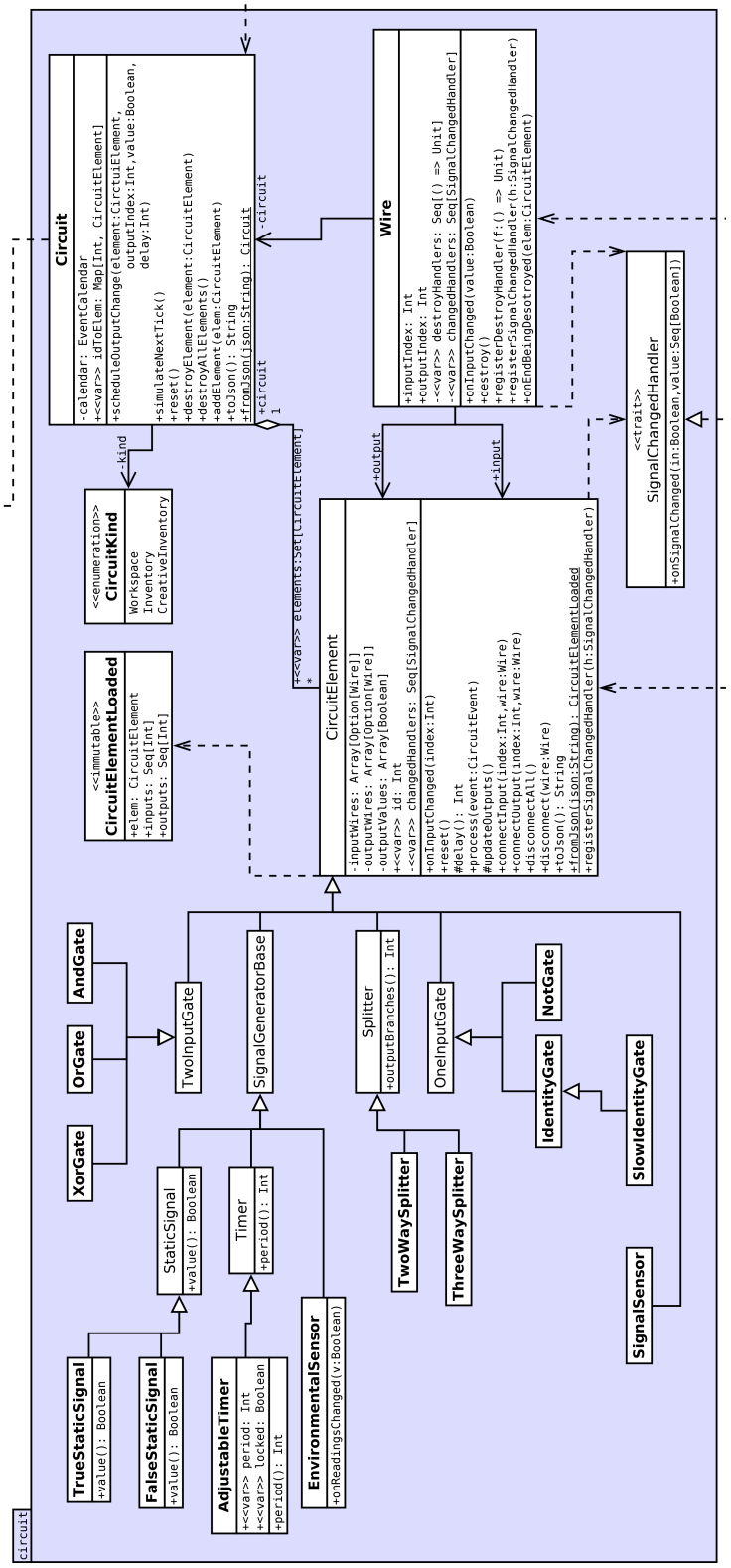
Obrázek C.1: Vztahy mezi vybranými balíky

C.1 Balík `event`



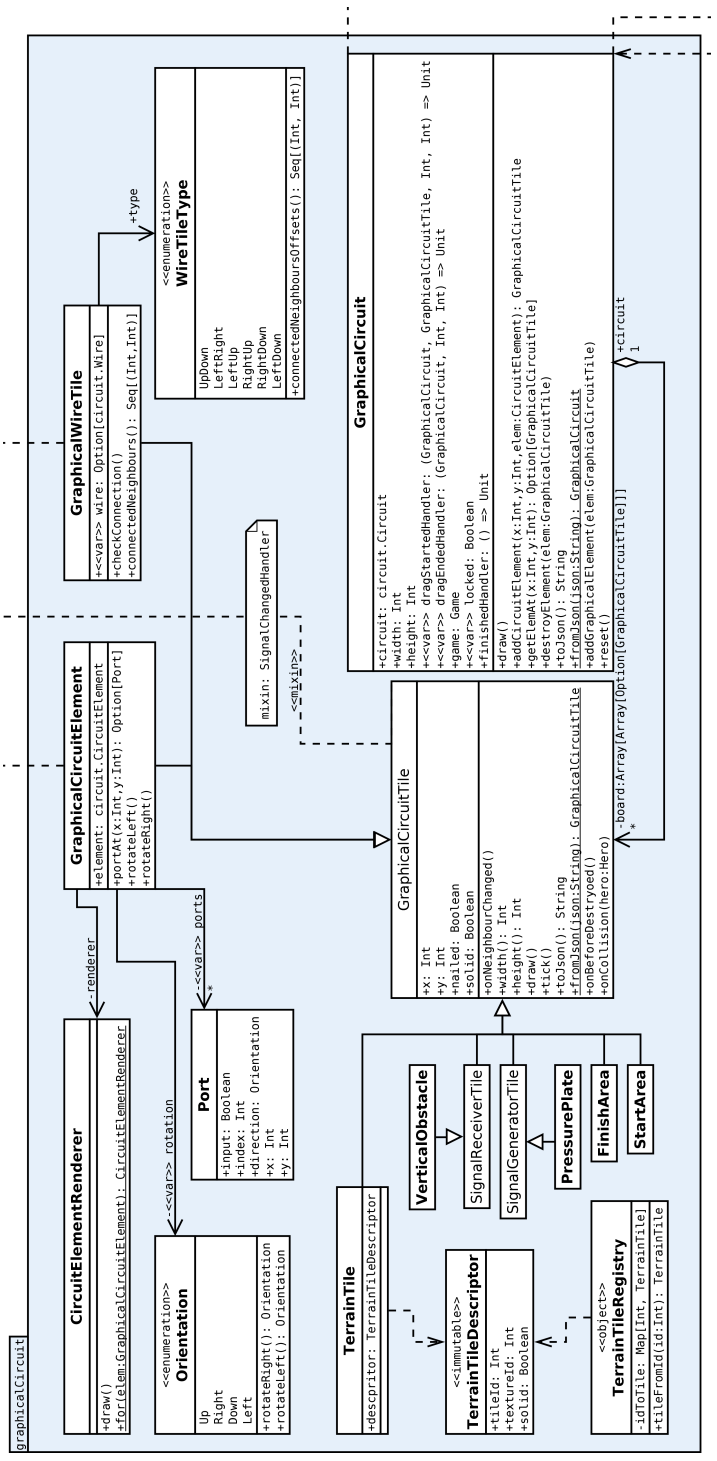
Obrázek C.2: Diagram tříd balíku `event`

C.2 Balík `circuit`



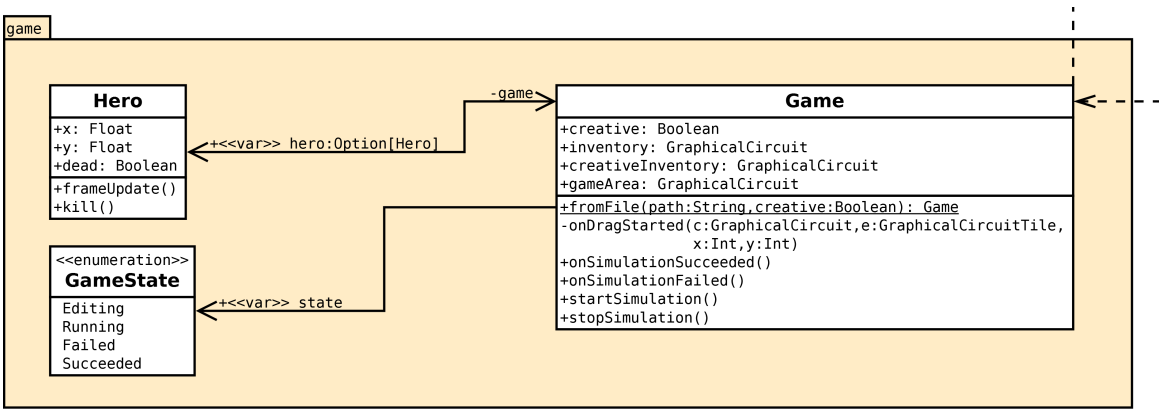
Obrázek C.3: Diagram tříd balíku `circuit`

C.3 Balík `graphicalCircuit`



Obrázek C.4: Diagram tříd balíku `graphicalCircuit`

C.4 Balík game



Obrázek C.5: Diagram tříd balíku game

Příloha D

Nástroj Dia2Scala

Tento program vznikl v průběhu psaní první poloviny této práce. Jeho úkolem je konvertovat soubor ve formátu programu Dia, tj. diagramu tříd, do zdrojových souborů programovacího jazyka Scala. Vytvoří tak startovní bod pro proces implementace, ušetří ruční přepisování všech navržených tříd, atributů a metod.

D.1 Popis generátoru

Nástroj podporuje dědění jak mezi třídami, tak i třídou od několika konstruktů trait (detailněji popsáno v části B.1). Zvládá všechny možnosti zápisu mutable a immutable třídy a atributu tak, jak bylo vysvětleno v sekci B.2. Stejně tak jsou podporovány různé notace konstruktu objekt, který byl objasněn v části B.3. V neposlední řadě připraví i objekty typu „Enumeration“, což je výčtový typ (velmi podobný kupř. „enum“ z jazyka Java). Program rozumí i balíkům a lze zapnout automatické seskupování tříd podle relace dědění.

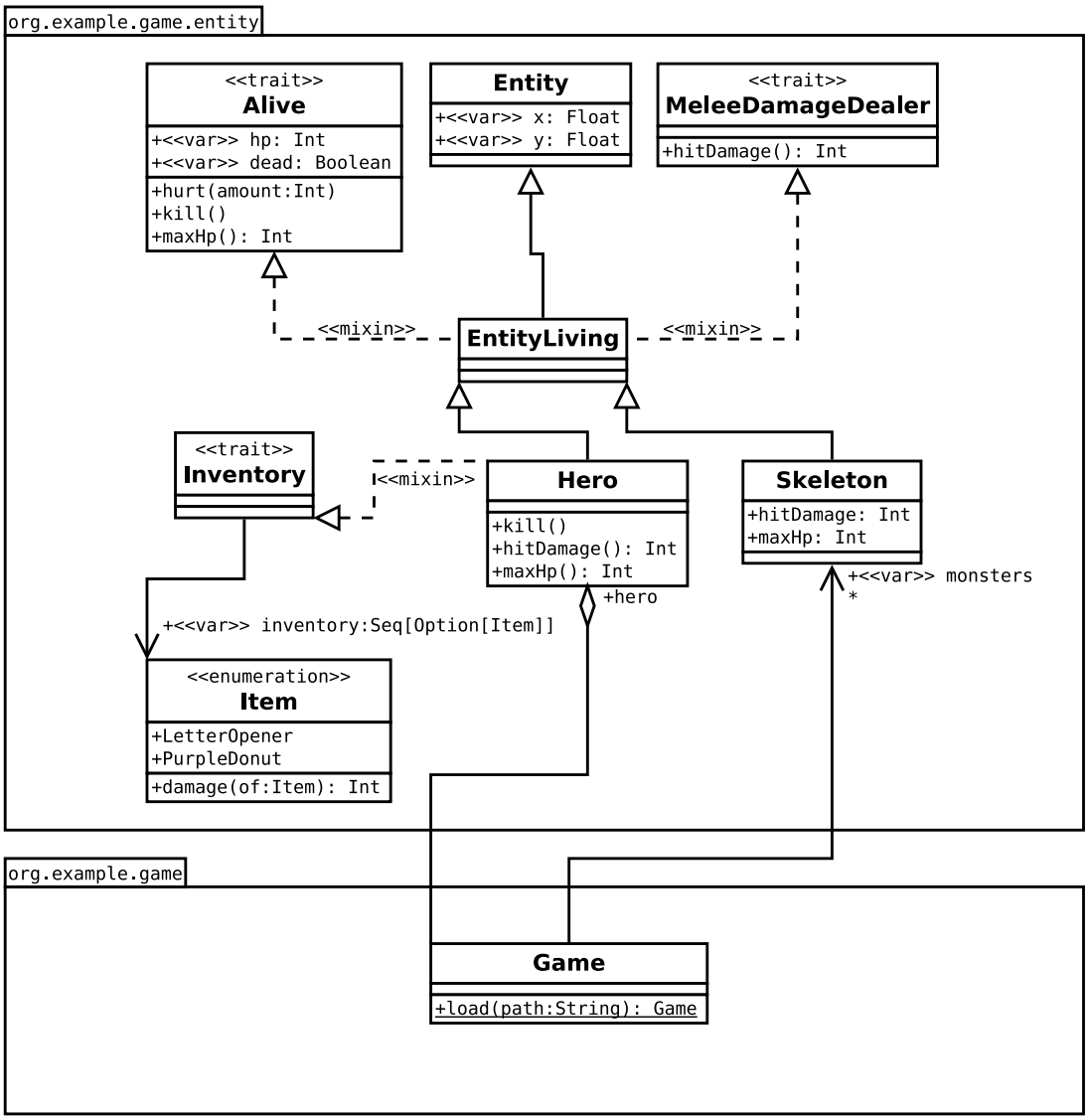
Scala je velmi obsáhlý jazyk, zdaleka ne všechny její přednosti byly použity pro vývoj výsledné herní aplikace. Řada pokročilejších vlastností, jakými jsou např. uživatelem definované generické třídy (viz. sekce B.4), proto není v nástroji Dia2Scala implementována.

I tak byl vývoj náročnější, než jsem původně očekával. Správná interpretace typů Scaly byla klíčová. Nástroj generuje příkazy „import“, uživatelský typ (třída) může být odkazována jak přímo jménem (leží ve stejném balíku), tak ale i nepřímo vazbou (asociace z UML). Při generování kódu musí nástroj správně zacházet se jménem třídy – použít relativní jméno třídy v jiném balíku, než ve kterém je třída definována, povede k chybě při kompilaci. Druhá možnost je méně kritická, pro programátora však velmi nepohodlná – používat značně dlouhá plně kvalifikovaná jména tříd i tam, kde nejsou nutně nezbytná.

Celý projekt je uvolněn pod Open Source licencí GPL3. Zdrojové kódy nástroje Dia2Scala jsou vystaveny na službě GitHub, kdokoliv tedy může zasílat opravy nebo celý projekt tzv. „forknout“ a nadále jej vylepšovat a vydávat i bez mého explicitního svolení.

D.2 Ukázka

Nástroj je spuštěný s parametry `-f example01.dia -d`, které nastavují vstupní soubor a zapnutí shlukování tříd podle relace dědění. Vstupní soubor v grafické podobě vidíme na obrázku D.1. Následují jednotlivé vygenerované zdrojové soubory, popisek u každého z nich odpovídá jménu s relativní cestou (ve výchozím nastavení složka „out“ v aktuálním pracovním adresáři).



Obrázek D.1: Vstupní soubor příkladu nástroje Dia2Scala

```scala
package org.example.game.entity

trait MeleeDamageDealer {

  def hitDamage(): Int = ???

}
```

Zdrojový kód D.1: `org/example/game/entity/MeleeDamageDealer.scala`

```scala
package org.example.game.entity

trait Alive {
  var hp: Int = _
  var dead: Boolean = _

  def hurt(amount: Int){ ??? }

  def kill(){ ??? }

  def maxHp(): Int = ???

}
```

Zdrojový kód D.2: `org/example/game/entity/Alive.scala`

```scala
package org.example.game.entity

class Entity {
  var x: Float = _
  var y: Float = _

}


class EntityLiving extends Entity with Alive with MeleeDamageDealer {

}


class Hero extends EntityLiving with Inventory {

  override def kill(){ ??? }

  override def hitDamage(): Int = ???

  override def maxHp(): Int = ???

}


class Skeleton extends EntityLiving {
  override val hitDamage: Int = ???
  override val maxHp: Int = ???

}
```

Zdrojový kód D.3: `org/example/game/entity/Entity.scala`

```
package org.example.game.entity

import org.example.game.entity.Item._

trait Inventory {
  var inventory: Seq[Option[Item]] = _

}
```

Zdrojový kód D.4: `org/example/game/entity/Inventory.scala`

```
package org.example.game.entity

object Item extends Enumeration {
  type Item = Value
  val LetterOpener, PurpleDonut = Value

  def damage(of: Item): Int = ???

}
```

Zdrojový kód D.5: `org/example/game/entity/Item.scala`

```
package org.example.game

import org.example.game.Game._
import org.example.game.entity.Hero
import org.example.game.entity.Skeleton

class Game {
  val hero: Hero = ???
  var monsters: Seq[Skeleton] = _

}


object Game {

  def load(path: String): Game = ???

}
```

Zdrojový kód D.6: `org/example/game/Game.scala`

Příloha E

Dotazník

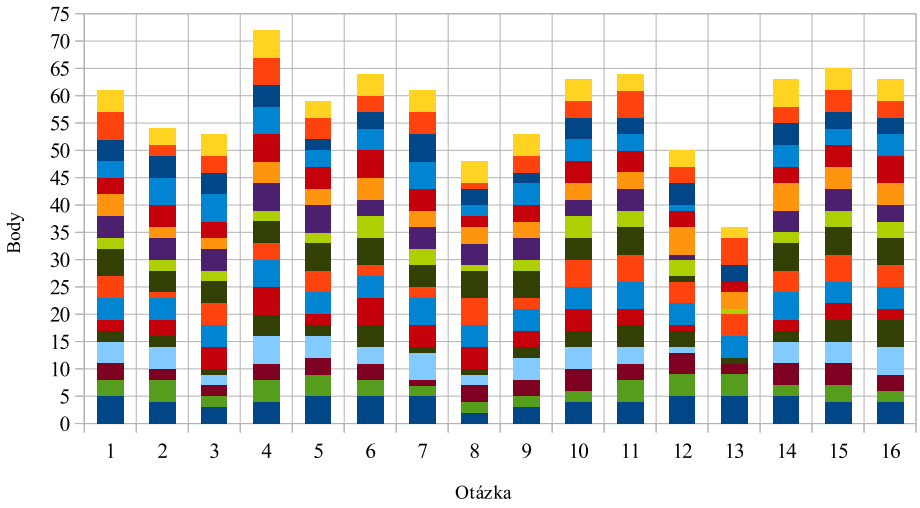
Níže uvádím otázky z dotazníku pro hodnocení uživateli. Položky 1–10 jsou překladem z hojně užívaného testu použitelnosti „System Usability Scale“ [6].

1. Nevadilo by mi si tuto hru ještě zahrát.
2. Shledávám hru zbytečně složitou.
3. Myslel jsem si, že hra byla jednoduchá z hlediska použití (ovládání).
4. Myslím, že bych potřeboval pomoc technicky založené osoby, abych zvládl ovládání této hry.
5. Zjistil jsem, že různé funkce jsou do hry dobře začleněné.
6. Myslel jsem si, že hra nebyla příliš konzistentní.
7. Řekl bych, že většina lidí by se naučila hrát tuto hru velmi rychle.
8. Uvědomil jsem si, že ovládání hry bylo nešikovné.
9. Při hraní této hry jsem si byl jistý tím, co dělám.
10. Potřeboval jsem se naučit spoustu věcí, než jsem mohl začít hru hrát.
11. Průvodní text (v dolní části obrazovky) mi přišel nápomocný.
12. Využíval jsem možnost nápovědy pro konkrétní prvek.
13. Nápověda pro konkrétní prvek mi pomohla. (Přeskočte, pokud jste nevyužili tuto nápovědu.)
14. Grafická a estetická stránka, na poměry zdarma dostupné výukové hry, mi přišla dostatečná.
15. Dovedu si představit, že by hra mohla někoho povzbudit k zájmu o technický obor (studium či koníček).
16. Myslím, že by hra mohla být představena žákům na základní škole jako pomůcka při zvažování oboru budoucího studia.

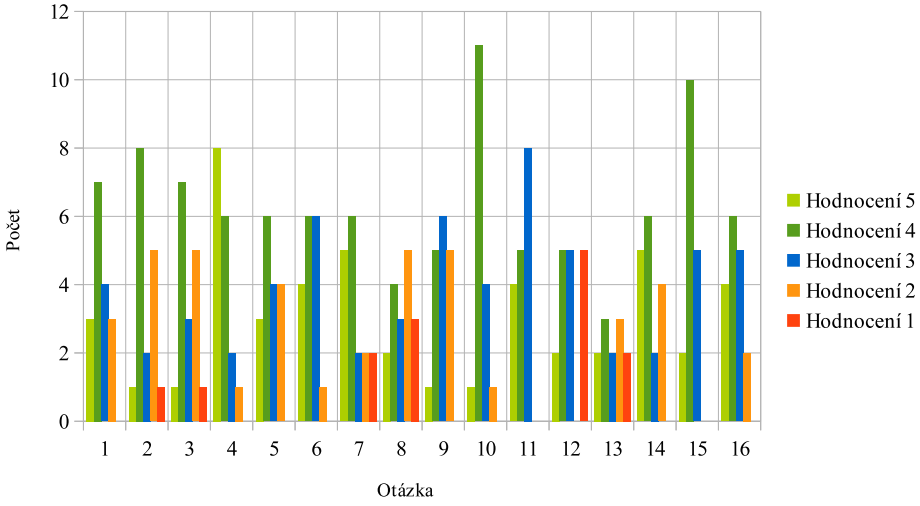
Příloha F

Detailní vyhodnocení dotazníků

Zde uvádím několik diagramů, které se podrobněji zabývají především strukturou hodnocení.



Obrázek F.1: Hodnocení otázek jednotlivými uživateli



Obrázek F.2: Počty jednotlivých ohodnocení podle otázky