

# Kutil

Tomáš Křen

2010

## Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
1.1	Zadání bakalářské práce . . . . .	2
1.2	Uvedení do problému . . . . .	3
1.3	<i>KData</i> jako seznam . . . . .	4
1.3.1	předmět <i>krabice</i> . . . . .	5
1.3.2	Předmět <i>symbol</i> . . . . .	5
1.3.3	Předmět <i>číslo</i> . . . . .	5
1.3.4	Důležité <i>elementární funkce</i> pro práci s <i>krabicemi</i> , <i>čísly</i> a <i>symbols</i> . . . . .	5
1.4	<i>KData</i> jako výraz Lambda kalkulu . . . . .	7
1.4.1	Definice výrazu Lambda kalkulu . . . . .	7
1.4.2	Definice volných proměnných . . . . .	8
1.4.3	Definice substituce . . . . .	8
1.4.4	Beta konverze . . . . .	8
1.4.5	Důležité <i>elementární funkce</i> pro práci s výrazy Lambda kalkulu . . . . .	8
1.5	Kontext . . . . .	9
1.5.1	Přiřazení <i>kData</i> -hodnoty k <i>symbolu</i> . . . . .	9
1.5.2	Přiřazení <i>uživatelské funkce</i> k <i>symbolu</i> . . . . .	10
1.5.3	Rozsah platnosti kontextu . . . . .	10
<b>2</b>	<b>Uživatelské rozhraní</b>	<b>10</b>
2.1	Ovládací okno . . . . .	11
2.1.1	Ovládání . . . . .	11
2.1.2	Konzole . . . . .	12
2.1.3	Palety předmětů . . . . .	12
2.2	Okno reality a ovládání pomocí kláves . . . . .	14
2.3	XML formát uložených vynálezů . . . . .	14

<b>3</b>	<b>Zavedení implementačních termínů</b>	<b>15</b>
3.1	Tvar . . . . .	15
3.2	KData . . . . .	15
3.3	Částice . . . . .	15
3.4	Fyzikální simulace . . . . .	16
3.4.1	Posluchač kolizí . . . . .	16
3.5	Plán . . . . .	16
3.6	Věc . . . . .	16
3.7	Plán funkce . . . . .	16
3.8	Kontext . . . . .	17
3.9	Dimenze . . . . .	17
3.10	Teleport . . . . .	17
3.11	Funkce . . . . .	17
3.12	Realita . . . . .	18
<b>4</b>	<b>Implementace v jazyku Java</b>	<b>19</b>
<b>5</b>	<b>Programy s podobným zaměřením</b>	<b>19</b>
5.1	The Incredible Machine . . . . .	19
5.2	Další podobné hry . . . . .	20
<b>6</b>	<b>Využívané zdroje</b>	<b>20</b>
6.1	phys2d . . . . .	20
6.2	XmlWriter . . . . .	20
6.3	Metoda aktivního vykreslování . . . . .	20
<b>7</b>	<b>OS, jazyk, vývojové prostředí</b>	<b>20</b>
<b>8</b>	<b>Co zbývá dodělat a možná rozšíření programu</b>	<b>21</b>
8.1	Nejvyšší priorita . . . . .	21
8.2	Střední priorita . . . . .	21
8.3	Nižší priorita . . . . .	21

# 1 Úvod

## 1.1 Zadání bakalářské práce

Cílem práce je vytvořit program fungující jako nástroj pro programování v dvojrozměrném fyzikálním prostředí. Z hlediska uživatelského rozhraní bude podobný hře The Incredible Machine. Program bude umožňovat uživateli interaktivně vytvářet stroje sestavené z předmětů různých typů. Interakci předmětů mezi sebou navzájem bude zajišťovat fyzikální simulátor. Důležitou vlastností programu bude, že bude možno stroje vkládat do "krabičky se vstupem a výstupem", které bude možno použít jako součástku nějakého dalšího stroje. Cílem práce je také pokusit se nalézt a převést i další primitiva programovacího světa

do světa fyzikálního. Program by měl sloužit jako hra přibližující dětem svět programování skrze intuitivnost fyzikálního světa.

## 1.2 Uvedení do problému

Kutil je program, který si klade za cíl umožnit modelovat virtuální světy, jejichž stavebními bloky jsou z jedné části předměty fyzického světa a z druhé části syntax jazyka. Jinými slovy Kutil je volný přechod mezi trojicí: programovací jazyk, vývojové prostředí a fyzikální simulátor. Virtuální svět vymodelovaný v rámci Kutila budeme nazývat *realita*.

*Realita* v Kutilovi se podobá skupině navzájem propojených místností. Každá místnost je dvojrozměrný prostor, ve kterém jsou umístěny předměty. Některé pohyblivé, ostatní statické. Na pohyblivé předměty působí odshora dolu gravitace. Tyto místnosti budeme nazývat *dimenze*. Tedy *realita* se skládá z několika *dimenzí*, z *reality* však uživatel v jeden okamžik vidí jedinou *dimenzi*, této dimenzi budeme říkat *aktuální dimenze*.

Každý předmět má navíc ke svým fyzikálním vlastnostem (jako jsou tvar, váha, pozice, rychlost atd.) ještě jednu důležitou vlastnost, kterou je jeho textová složka, kterou nazýváme *kData*. *KData* můžeme reprezentovat jedním textovým řetězcem, ale tomu jak je interpretovat se meze nekladou.

Velice důležitým druhem předmětu jsou statické předměty nazývané *elementární funkce*. Základní formou *elementární funkce* je *unární operace*: jedná se o krabíčku, do které může z vrchu spadnout libovolný pohyblivý předmět (tím, že do ní spadne, zmizí). Pokud se tak stane, tak v závislosti na definici této *unární operace* a na tom co do ní spadlo<sup>1</sup> za okamžik ze spodku *unární operace* vypadne jiný pohyblivý předmět. To co spadlo do *unární operace* je jejím vstupem, to co vypadlo je výstupem.

Obdobně fungují i další typy *elementárních funkcí*:

- *binární operace* - krabíčka tvaru písmene V, kterážto má dva vstupy a jeden výstup. Provádí akci až poté co do ní přišli oba dva vstupy.
- *roz dvojka* - krabíčka tvaru písmene A, kterážto má jeden vstup a dva výstupy
- *křížovatka* - krabíčka tvaru písmene X, kterážto má dva vstupy a dva výstupy. Provádí akci až poté co do ní přišli oba dva vstupy.

Dalším důležitým druhem předmětu je statický předmět nazývaný *funkce* (nebo výstižněji *uživatelská funkce*). Je velice podobný *unární operaci*, s tím rozdílem, že při pádu předmětu do *funkce* vznikne nová *dimenze*. Ta je postavena podle *plánu funkce* této *funkce* (více o *plánu funkce* viz 3.7). Nahoře uprostřed nové *dimenze* se objeví to, co spadlo do *funkce*. Následně se v této *dimenzi* něco děje (v závislosti na tom co v ní je za předměty, tedy v závislosti na *plánu funkce*) až do té doby než něco dalšího spadne na spodek *dimenze*. V tuto chvíli se přepne okno znovu na zobrazování původní *dimenze* a vnitřní *dimenze* je ukončena.

<sup>1</sup>Častokrát jediná vlastnost bráná v potaz *elementárními funkcemi* je právě *kData*.

To co spadlo na spodek *dimenze* (stane se to výstupem této *funkce*) vypadne z krabičky ven.

Obdobně jako *funkce* funguje předmět *rekurze*. Při vstupu předmětu do *rekurze* se jako *plán funkce* použije *plán funkce* pro *funkci*, ve které je předmět *rekurze* použit.

Dalším podstatným typem pohyblivého předmětu jsou *panáčky*, jedná se o předměty, jejichž chování může uživatel ovlivnit za běhu simulace. Jedná se tedy o abstrakci „vstupního proudu“.

## Více o *kData*

*KData* jsou textovou složkou každého předmětu. Ačkoli o *kDatech* mluvíme jako o textové složce, vnitřně nejsou *kData* reprezentována jako textový řetězec, naproti tomu jsou vnitřně mnohem blíže seznamu. Dále je také implementován přístup, který na *kData* pohlíží jako na výrazy Lambda kalkulu.

### 1.3 *KData* jako seznam

Přestože bylo řečeno, že tomu, jak interpretovat *kData* se meze nekladou, existuje jedna interpretace, kterážto je využívána většinou *elementárních funkcí*. Tyto *elementární funkce* na *kData* nahlíží jako na jednu z těchto věcí:

- Seznam, jehož položky jsou *KData*
- Číslo
- Symbol

Seznam je zapisován podobně jako v jazyce LISP, to znamená na začátku seznamu je levá závorka následovaná mezerou, pak následuje výčet prvků oddělených mezerou a na konci seznamu je pravá závorka předcházená mezerou. Příklad několika seznamů:

- ( a b c )
- ( 1 2 3 )
- ( )
- ( ( a b c ) ( 1 2 3 ) ( ) x y z ( ( ) ( ) ) )

Číslem se rozumí přirozené číslo (včetně nuly). Například:

- 0
- 42

Symbolem se rozumí libovolný textový řetězec, který:

- není číslo nebo seznam
- a neobsahuje mezeru

Například:

- `x`
- `foo`
- `λ`
- `_`

Pokud se díváme na *kData* jako na textový řetězec, tak může nabývat pouze tyto právě popsané hodnoty.<sup>2</sup>

### 1.3.1 předmět *krabice*

Pro každá *kData* ve formě seznam existuje předmět *krabice*, který:

- Vypadá jako otevřená krabice.
- Má jako *kData* jemu odpovídající seznam.

### 1.3.2 Předmět *symbol*

Pro každá *kData* ve formě symbol existuje předmět *symbol*, který:

- Vypadá jako malá oranžová krabička s nápisem odpovídajícím tomu danému symbolu.
- Má jako *kData* jemu odpovídající symbol.

*Symbol @err* se používá jako chybový výstup funkcí.

### 1.3.3 Předmět *číslo*

Pro každá *kData* ve formě číslo existuje předmět *číslo*, který:

- Vypadá jako malý modrý trojúhelník s nápisem odpovídajícím tomu danému číslu.
- Má jako *kData* jemu odpovídající číslo.

### 1.3.4 Důležité *elementární funkce pro práci s krabicemi, čísly a symboly*

Všechny *elementární funkce* vracejí jako výstup buď *krabici*, *číslo* nebo *symbol*, v závislosti na tom, jaká jsou *kData* vstupu. Výjimku tvoří situace, kdy je vstupem nějaký *panáček*. Potom funkce vrátí to, jak je definována, ale nejčastěji *panáčka*, pokud má výstup *kData* seznam.

---

<sup>2</sup>To znamená:

- buď je *kData* text bez mezer a závorek obsahující aspoň jeden nečíselný znak [Symbol]
- nebo je *kData* text pouze z číselných znaků [Číslo]
- nebo je *kData* dobře uzávorkovaný text (s mezerami po respektive před závorkou a s neopakujícími se závorkami bezprostředně po sobě) [Seznam]

**2Arg typ A** Tato rozdvojka požaduje na vstupu dvouprvkový seznam. Pokud ho dostane, vrací na výstup1 první prvek a na výstup2 druhý prvek. Pokud ho nedostane, vrací na obou výstupech chybový symbol.

**2Arg typ V** Tato binární operace vrací seznam ( vstup1 vstup2 ) .

**+** Binární operace. Pokud jsou vstup1 i vstup2 čísla, pak je sečte. Jinak se chová jako zřetězovač seznamů (vrací seznam který nejprve obsahuje prvky vstup1 a následně vstup2). Pokud některý vstup není seznam, předpokládá místo něj seznam obsahující pouze tento vstup.

**: typ A** Rozdvojka, která oddělí hlavu seznamu (první prvek) od těla seznamu (seznam neobsahující první prvek). Hlava seznamu je vrácena jako výstup1, tělo jako výstup2. Pokud je vstupem prázdný seznam, je vrácen pouze výstup2, kterým je prázdný seznam. Pokud je vstupem něco jiného než seznam, je tento předmět poslán na jako jediný výstup, výstup1.

**: typ V** Binární operace, která předpokládá na vstupu2 seznam. Vrací seznam, který obsahuje jako hlavu vstup1 a jako tělo vstup2. Pokud je vstup2 něco jiného než seznam, potom se jako vstup2 předpokládá místo něj seznam obsahující pouze tento vstup2.

**head** Rozdvojka, která předpokládá na vstupu seznam. Na výstup1 vrací hlavu seznamu, na výstup2 vrací celý vstup. Pokud vstupem není seznam, potom vrací pouze výstup2.

**copy** Tato rozdvojka vrací na obou výstupech vstup.

**isNull** Rozdvojka, která vrací vstup na jeden z výstupů, pokud je vstup prázdný seznam, vrací na výstup2. Pokud není vstup prázdný seznam, vrací na výstup1.

**zamíchej** Unární operace, která předpokládá na vstupu seznam. Vrací seznam stejných prvků, ale v náhodném pořadí. Pokud nedostane seznam, vrací vstup.

**obal** Tato unární operace vrací seznam ( vstup ) .

**interval** Tato binární operace předpokládá na obou vstupech číslo. Vrací seznam ( vstup1 , vstup1 + 1 , vstup1 + 2 , ... , vstup2 ) . Pokud nedostane na některém vstupu číslo, bere tento vstup za nulu.

**laťka** Křížovatka, která předpokládá na vstup1 číslo a na vstup2 seznam čísel. Na výstup1 je vrácen seznam čísel ze vstup2 menších nebo rovných vstup1 (v původním pořadí). Na výstup2 je vrácen zbytek seznamu (v původním pořadí). Cokoli co není číslo není menší nebo rovno než libovolné číslo. Pokud vstup2 není seznam, pak se předpokládá vstup2 ( `vstup2` ).

**spojSymboly** Binární operace, na obou vstupech předpokládá symboly. Pokud je dostane vrátí zřetězení těchto dvou symbolů. (Např. pokud je vstup1 symbol `foo` a vstup2 symbol `bar`, pak vrátí symbol `foobar`.) Jinak vrátí chybový symbol.

## 1.4 *KData* jako výraz Lambda kalkulu

Další možností jak interpretovat *kData*, kterou Kutil nabízí, je jako výrazy Lambda kalkulu<sup>3</sup>.

### 1.4.1 Definice výrazu Lambda kalkulu

Výraz Lambda kalkulu je definován následovně:

- $\langle \text{výraz} \rangle := \langle \text{proměnná} \rangle \mid \langle \text{funkce} \rangle \mid \langle \text{aplikace} \rangle$
- $\langle \text{funkce} \rangle := ( \lambda \langle \text{proměnná} \rangle . \langle \text{výraz} \rangle )$
- $\langle \text{aplikace} \rangle := ( \langle \text{výraz} \rangle \langle \text{výraz} \rangle )$

Jako proměnná může vystupovat jakýkoliv textový řetězec bez *bílých znaků* a znaků *lambda* „ $\lambda$ “, *tečka* „ $.$ “, *levá závorka* „ $($ “ a *pravá závorka* „ $)$ “.

Dále používáme zkratky:

$$(V_1 V_2 V_3 \cdots V_n) \equiv (\cdots ((V_1 V_2) V_3) \cdots V_n)$$

$$(\lambda x_1 x_2 \cdots x_n . V) \equiv (\lambda x_1 . (\lambda x_2 . (\cdots (\lambda x_n . V) \cdots)))$$

Kde  $V_1, \dots, V_n$  jsou výrazy a  $x_1, \dots, x_n$  jsou proměnné.

Pro zachování korespondence *kData* s výrazy Lambda kalkulu, požadujeme u funkcí a aplikací vnější závorky. Potom je každá funkce a aplikace zapsána jako seznam. A samostatné proměnné jsou symboly (nebo čísla, čísla však není doporučeno používat jako proměnné). Tím dostáváme že předměty, které reprezentují Lambda výrazy jsou nejčastěji *krabice*, případně *symboly* (nebo nedoporučovaná *čísla*).

<sup>3</sup>Hezké shrnutí toho, co je Lambda kalkulus z [tutorLC]: „The lambda calculus can be called the smallest universal programming language of the world. The lambda calculus consists of a single transformation rule (variable substitution) and a single function definition scheme. It was introduced in the 1930s by Alonzo Church as a way of formalizing the concept of effective computability. The calculus is universal in the sense that any computable function can be expressed and evaluated using this formalism. It is thus equivalent to Turing machines. However, the lambda calculus emphasizes the use of transformation rules and does not care about the actual machine implementing them. It is an approach more related to software than to hardware.“

### 1.4.2 Definice volných proměnných

*Množina volných proměnných výrazu  $E$ , značená  $FV(E)$ , má následující indukativní definici:*

$$\begin{aligned} FV(x) &= \{x\} \\ FV((M\ N)) &= FV(M) \cup FV(N) \\ FV((\lambda x. M)) &= FV(M) \setminus \{x\} \end{aligned}$$

### 1.4.3 Definice substituce

Výsledek *substituce*  $N$  za *volné výskyty proměnné  $x$  ve výrazu  $M$* , značeno  $M[x := N]$ , je definována následovně:

$$\begin{aligned} x[x := N] &\equiv N \\ y[x := N] &\equiv y, \text{ jestliže } \neg(x \equiv y) \\ (M_1\ M_2)[x := N] &\equiv (M_1[x := N])(M_2[x := N]) \\ (\lambda y. M_1)[x := N] &\equiv (\lambda y. (M_1[x := N])) \end{aligned}$$

### 1.4.4 Beta konverze

Nejdůležitějším axiomatickým schématem Lambda kalkulu je takzvaná beta konverze:

$$((\lambda x. M)\ N) = M[x := N]$$

### 1.4.5 Důležité *elementární funkce* pro práci s výrazy Lambda kalkulu

**beta** Unární operace, která interpretuje svůj vstup jako výraz Lambda kalkulu a vrátí jemu ekvivalentní výraz vzniklý použitím *beta konverze*.

**fullBeta** Unární operace, která opakovaně provádí beta konverzi, do té doby než výraz před a po konverzi jsou totožné. To však nedělá do nekonečna, ale maximálně tolikrát kolikrát je nastaveno v systému (např. 1000 krát).

**apply** Binární operace, která interpretuje své vstupy jako lambda výrazy. Vrací výraz, který vznikne po beta konverzi výrazu  $(\text{vstup1}\ \text{vstup2})$ . Pokud je vstup1 *číslo*, vrací chybový symbol. Pokud je vstup1 *symbol*, potom má tato binární operace speciální chování popsané v části 1.5.2.

**numeral** Unární operace, která předpokládá jako vstup *číslo*. Pokud ho nedostane vrací chybový symbol. Pokud ho dostane vrací výraz Lambda kalkulu odpovídající *Churchově numerálu*, konkrétně:

- $0 \equiv (\lambda f\ x. x)$
- $1 \equiv (\lambda f\ x. (f\ x))$
- $2 \equiv (\lambda f\ x. f\ (f\ x))$
- $3 \equiv (\lambda f\ x. f\ (f\ (f\ x)))$



- $4 \equiv (\lambda f x . f (f (f (f x))))$
- a tak dále.

## 1.5 Kontext

*Kontext* představuje realizaci mechanismu umožňujícího přiřadit *symbolu* nějakou hodnotu v podobě *kData*, nebo nějakou *uživatelskou funkci*.

### 1.5.1 Přiřazení *kData*-hodnoty k *symbolu*

**Binární operace** := Pokud je vstup1 symbol, pak se do kontextu pod jméno tohoto symbolu uloží kData vstupu2 a výstupem se stává vstup1.

Pokud jsou vstup1 i vstup2 seznamy stejné délky, pak se se všemi prvky seznamu vstup1 a odpovídajícími prvky v seznamu vstup2 rekurzivně provede tato binární operace a výstupem se stává vstup1 (pokud ale výstupem některého rekurzivního volání není chybový symbol, v tom případě je výstupem chybový symbol).

V ostatních případech je výstup chybový symbol.

*Několik příkladů:*

- vstup1 = x , vstup2 = 42  
⇒ záznam v kontextu: „x=42“ , výstup = x
- vstup1 = x , vstup2 = 42  
⇒ záznam v kontextu: „x=42“ , výstup = x  
*opětovné použití operace := v té samé dimenzi:*  
vstup1 = x , vstup2 = 23  
⇒ záznam v kontextu: „x=23“ , výstup = x
- vstup1 = ( x y z ) , vstup2 = ( 1 2 3 )  
⇒ záznam v kontextu: „x=1, y=2, z=3“ , výstup = ( x y z )
- vstup1 = ( x y z ) , vstup2 = ( 1 2 )  
⇒ záznam v kontextu: „“ , výstup = @err
- vstup1 = ( x ( a b ) ) , vstup2 = ( 42 ( 1 2 ) )  
⇒ záznam v kontextu: „x=42, a=1, b=2“ , výstup = ( x ( a b ) )
- vstup1 = 42 , vstup2 = 23  
⇒ záznam v kontextu: „“ , výstup = @err

**Unární operace *dosad*** Slouží k vyzvednutí kDat z kontextu.

Pokud je vstupem symbol, nahlédne do kontextu pod tento symbol. Pokud je v něm záznam o tomto symbolu, vrátí předmět s kDaty stejnými jako jsou kData u tohoto symbolu v kontextu. Pokud v něm záznam není, vrátí *dosad* původní vstup.

Pokud je vstupem seznam, chová se obdobně, jenže pro jednotlivé prvky seznamu. Vrací seznam s dosazenými hodnotami.

Pokud je vstupem něco jiného (číslo), vrací jako výstup původní vstup.  
*Několik příkladů:*

- vstup =  $x$  , záznam v kontextu: „ $x=1$ “  
 $\Rightarrow$  výstup = 1
- vstup =  $x$  , záznam v kontextu: „ $y=1$ “  
 $\Rightarrow$  výstup =  $x$
- vstup = (  $a$   $b$   $c$  ) , záznam v kontextu: „ $a=42$  ,  $b=23$  ,  $c=101$ “  
 $\Rightarrow$  výstup = ( 42 23 101 )
- vstup = (  $a$   $b$   $c$  ) , záznam v kontextu: „ $a=42$  ,  $c=23$ “  
 $\Rightarrow$  výstup = ( 42  $b$  23 )
- vstup = (  $a$   $b$  1 (  $c$   $d$  ) ) , záznam v kontextu: „ $a=42$  ,  $d=23$ “  
 $\Rightarrow$  výstup = ( 42  $b$  1 (  $c$  23 ) )

### 1.5.2 Přiřazení *uživatelské funkce* k *symbolu*

Každá uživatelská funkce má své jméno (které je symbol), jednak je napsáno na této funkci jako popis, dále pak představuje *kData* tohoto předmětu. Funkce jsou přes svá jména přístupné prostřednictvím kontextu. K tomuto přístupu slouží binární operace *apply*, o které jsme mluvili již v části 1.4.5.

**apply** Pokud je této binární operaci předán jako vstup1 *symbol* jenž je stejný jako jméno nějaké funkce v kontextu, potom je spuštěna *uživatelská funkce* s daným jménem a se vstupem jímž je vstup2. Pokud taková funkce v kontextu není, pak je vrácen seznam ( vstup1 vstup2 ). Chování při jiných okolnostech je popsáno v části 1.4.5.

### 1.5.3 Rozsah platnosti kontextu

Přiřazení *kData*-hodnoty (respektive *uživatelské funkce*) k *symbolu* má rozsah platnosti v dimenzi  $D$ , kde došlo k použití binární operace  $:=$  (respektive v dimenzi  $D$ , kde se nachází daná *uživatelská funkce*) a v dimenzích *uživatelských funkcí*, které se nacházejí v dimenzi  $D$ . Tyto dimenze budeme nazývat *poddimenze dimenze D*. Říkáme, že poddimenze dimenze  $D$  dědí kontext dimenze  $D$ . Tento zděděný kontext pak mohou rozšířit o své použití operace  $:=$  či o své vnitřní uživatelské funkce.

Dále pak změny kontextů v poddimenzích dimenze  $D$  neovlivňují kontext dimenze  $D$ .

## 2 Uživatelské rozhraní

Uživatelské rozhraní programu Kutil sestává ze dvou typů oken:

- *Ovládací okno*, které je jedno pro jednu běžící instanci programu.




- *Okno reality*, kterých může být případně větší počet.




## 2.1 Ovládací okno

Ovládací okno je rozděleno do záložek. Jsou tři základní typy záložek:

- Ovládání (záložka se symbolem )
- Konzole (záložka se symbolem )
- Paleta předmětů (záložky se symboly , ,  $2+2=4$ , , ,  $\lambda$ )

### 2.1.1 Ovládání






**Tlačítka** ,  a  Práce v programu Kutil spočívá ve vkládání předmětů do interaktivního fyzikálního světa. Tento svět má tři základní stavy, které lze měnit třemi tlačítky:


- *PLAY* (tlačítko )
- *PAUSE* (tlačítko )
- *STOP* (tlačítko )


Po spuštění je program ve stavu *STOP*. V tomto stavu je fyzikální svět „zamrzlý“. Můžeme do něj vkládat nové předměty, přesouvat ty, které tam už jsou či je mazat.

Pokud se program dostane do stavu *PLAY*, fyzikální svět „rozmrzne“ a nastartuje se fyzikální simulace. Ze stavu *PLAY* se může program dostat do stavu *PAUSE*. Ten je podobný stavu *STOP*, fyzikální svět „zamrzne“ a můžeme do něj vkládat, přesouvat a mazat. Opětovným uvedením do stavu *PLAY* se svět zase rozběhne.


Pokud se program nachází ve stavu *PLAY* nebo *PAUSE* můžeme ho uvést do stavu *STOP*. V tom případě se rozložení předmětů vrátí do takového rozložení, ve které bylo při posledním stavu *STOP* a fyzikální svět znovu „zamrzne“.

**Tlačítka** , ,  a  Stisknutím tlačítka  se vytvoří nové *okno reality*. To představuje nový fyzikální svět, nezávislý na chodu jiných fyzikálních světů. (Více o *oknu reality* viz 2.2.)

Pokud je program ve stavu *STOP*, je možno uložit stav *reality* do souboru pomocí tlačítka . Více o formátu uloženého souboru viz 2.3. Více o tom, co přesně se ukládá viz 3.12.

Tlačítko  naproti tomu umožňuje otevřít dříve uložené soubory (obsah souboru se otevře do právě aktivního *okna reality*).

Tlačítko  vypne celý program.

**Výběr aktuální reality a dimenze** Na spodní části záložky ovládání se nalézají dva combo boxy ukazující, která *realita* (první combo box), respektive která *dimenze* (druhý) jsou právě aktuální. (To znamená, které *okno reality* je aktivní, respektive, která *dimenze* je právě vykreslována v aktivním *okně reality*). Dále umožňují změnu *aktuální reality* respektive *dimenze* (stačí vybrat z combo boxu požadovanou *realitu* respektive *dimenzi* a stisknout tlačítko ).

### 2.1.2 Konzole

Primární účel konzole je možnost vkládat do Kutila předměty na základě *kDat*. Pokud do ní uživatel napíše *kData* ve správném formátu a stiskne klávesu ENTER, může okamžitě vložit předmět odpovídající těmto *kDatům* (stačí přejet kurzorem myši na *okno reality*, do které chce předmět vložit). Více o předmětech odpovídajících určitým *kDatům* viz 1.3. Pokud uživatel napíše *kData* v nesprávném formátu, pak se vkládá *chybový symbol*.

Konzole navíc doplňuje mezery kolem symbolů: „(“ , „)”“ , „.”“ a „λ“. Písmeno „L“ je automaticky nahrazeno symbolem „λ“ (aby bylo možné pohodlně psát lambda výrazy). Pokud jsou vynechány nejnějnější závorky, pak je konzole doplní.

Sekundárním účelem konzole je možnost zadávat Kutilovy textové příkazy. Každý příkaz začíná symbolem „:“ za kterým bezprostředně následuje jméno příkazu, po něm případně následuje mezera a vstupní argumenty příkazu oddělené mezerou.

Zatím je implementován jen malý počet příkazů (a k tomu většina z nich neužitečných):

```
:beta <lambda-výraz> Do výstupu konzole se vypíše lambda výraz po jednom provedení beta konverze.
```

```
:echo <libovolný-text> Do výstupu konzole se vypíše zadaný <libovolný-text>.
```

```
:print-lambda Do výstupu konzole se vypíše symbol „λ“.
```

```
:exit Ukončí program. (Stejně funguje i :quit)
```

Dále pak konzole funguje jako textový výstup programu (pro logování nestandardních situací atd.).

### 2.1.3 Palety předmětů

Záložky s paletami předmětů obsahují tlačítka, každé toto tlačítko představuje nějaký předmět určený ke vložení. Poté co klikneme na tlačítko, stačí přejet kurzorem myši na *okno reality*, do které chceme předmět vložit. Ve chvíli kdy najedeme kurzorem na okno reality, začne se již do okna vykreslovat vkládaný předmět. Po kliknutí se předmět vloží do fyzikálního světa.



Tato paleta obsahuje základní obecné konstrukce Kutila.

**FUNKCE** Přidá novou *uživatelskou funkci*. Viz 1.2

**REKURZE** Přidá předmět *rekurze*. Viz 1.2

**:=, dosad'** Viz 1.5.1

**apply** Viz 1.5.2 a 1.4.5

**2arg typ A, 2arg typ V, copy** Viz 1.3.4



Tato paleta obsahuje předměty pro práci s *krabicemi*.

**KRABICE** *prázdná krabice (krabice s kData ())*

**head, : typ A, : typ V, +, isNull, 2arg typ A, 2arg typ V, zamíchej, obal**  
Vše viz 1.3.4



Tato paleta obsahuje předměty pro práci s čísly.

**0, 1, 2, ... ,10** vloží odpovídající *číslo*

**+, interval, laťka** Viz 1.3.4



Tato paleta obsahuje *panáčky*, to znamená předměty, které lze při běhu fyzikální simulace ovládat. Zatím obsahuje Kutil pouze jednoho panáčka:

**Budha** jednoduchý sedící poletující panáček



Tato paleta obsahuje jednoduché předměty:

**Míček** jednoduchý pohyblivý předmět

**Kostka** jednoduchý pohyblivý předmět

**Deska** jednoduchý statický předmět

**Domina** řada pohyblivých předmětů

**Houpačka** statický předmět s dynamickou částí

**$\lambda$**  Tato paleta obsahuje předměty pro práci s Lambda kalkulem a symboly.

**$\lambda$ ,  $x$ ,  $y$ ,  $z$ ,  $f$ , **f1**, **f2**, **f3**,  $a$ ,  $b$ ,  $c$**  Vloží odpovídající symbol

**spojSymboly** Viz 1.3.4

**apply** Viz 1.4.5 a 1.5.2

**beta**, **fullBeta**, **numeral** Viz 1.4.5

**SELF** Vloží (  $\lambda x . x x$  ), což odpovídá lambda výrazu pro sebe-aplikaci.

**S** Vloží (  $\lambda a b c . b ( a b c )$  ), což odpovídá lambda výrazu pro následníka Churchova numerálu.

## 2.2 Okno reality a ovládání pomocí kláves

Každé realitě odpovídá jedno *okno reality*. Toto okno zobrazuje aktuální dimenzi této reality.

Kliknutím na libovolný předmět se tento předmět označí (jeho okraj se zbarví do červena) a napravo od něj se zobrazí textová reprezentace jeho *kData*. Stisknutím klávesy ESC se předmět opět odznačí.

Pokud je nějaký předmět označen, tahem myši za zmáčknutého tlačítka je možno ho přesunout. Zmáčknutím klávesy DELETE předmět odstraníme.

Pokud je realita ve stavu *STOP*, je možno procházet napříč dimenzemi jednotlivých uživatelských funkcí a tím editovat obsah těchto funkcí: pokud chceme vstoupit do dimenze nějaké funkce, označíme ji a stiskneme klávesu ENTER. Pokud se chceme naopak vrátit o úroveň více, stiskneme tlačítko BACKSPACE.


Klávesou P se přepíná mezi stavem *PLAY* a *PAUSE*.

Klávesou S se přepne do stavu *STOP*.

Klávesou O se simulace pohne o jeden krok a nastaví se stav *PAUSE*.

*Panáčky* se ovládají pomocí kláves NAHORU, DOLU, DOPRAVA, DOLEVA.

## 2.3 XML formát uložených vynálezů

Pokud uložíme vynález do souboru pomocí tlačítka , pak je tento soubor typu XML. Všem předmětům, kromě *uživatelských funkcí*, odpovídá jeden tag následujícího formátu:

```
<jméno-předmětu x="x-pos" y="y-pos" kdata="textová-kData" />
```

Naproti tomu *uživatelská funkce*, má následující formát:

```
<funkce x="x-pos" y="y-pos" kdata="jméno-funkce">vnitřní-předměty-funkce</funkce>
```

Jak u funkce tak u ostatních předmětů jsou atributy *x* a *y* povinné, atribut *kdata* je nepovinný, pokud není uveden předpokládá se hodnota „( )“.

*Příklad uloženého souboru:*

```
<?xml version="1.0" encoding="UTF-8"?>
<vynalez>
  <funkce x="291" y="307" kdata="f1">
```

```

        <copy x="292" y="254"/>
        <dva-arg-tyt-v x="290" y="387"/>
    </funkce>
    <krabice x="292" y="148" kdata="( 1 2 3 )"/>
</vynalez>

```

### 3 Zavedení implementačních termínů

Nyní zavedeme termíny, ve kterých budeme operovat v částech textu zabývajících se implementací. Tato část se snaží problém popsat „odzola nahoru“, tedy od popisu elementárních částí po celkový obraz. Naproti tomu přístup v části 4 zabývající se implementací je „odshora dolů“, tedy začínající u popisu tříd zastřešujících celý program a končící u elementárních tříd.

Tyto termíny nejsou nutně vyčerpávající nebo absolutně přesné, jejich cílem je hlavně snazší orientace v dalším textu.

#### 3.1 Tvar

*Tvar* udává dvě následující věci:

- Rovinný geometrický tvar (např. obdélník, polygon, kruh)
- Vykreslovaný obrázek

#### 3.2 KData

*KData* představuje jednu z následujících datových struktur:

- Seznam, jehož položky jsou *KData*
- Symbol
- Číslo

#### 3.3 Částice

Každá *částice* má:

- má *Tvar*
- má pozici v rovině
- má úhel, pod kterým je natočena
- má rychlost
- zná *věc* (viz 3.6), do které náleží

A navíc může mít další nepovinné atributy.

Dále rozlišujeme dva základní druhy *částic*:

- *Pohyblivé*: tzn. ty kterým se může měnit pozice, úhel a rychlost
- *Statické*: tzn. ty kterým se nemůže měnit pozice a úhel. A jejichž rychlost je nulová.

### 3.4 Fyzikální simulace

*Fyzikální simulace* je tvořena:

- několika *částicemi*
- několika *posluchači kolizí*

*Fyzikální simulace* umožňuje, aby do ní byly vkládány *částice*. Tyto částice pak při každém *kroku fyzikální simulace* společně navzájem interagují, podle pravidel této *fyzikální simulace*. Tato interakce má za důsledek změnu atributů (jako jsou pozice, natočení, rychlost, atd.) částic obsažených v této simulaci.

Další pravidla do *fyzikální simulace* můžeme přidávat přidáním odpovídajícího *posluchače kolizí*.

#### 3.4.1 Posluchač kolizí

Pokud dojde ke kolizi dvou *částic* v nějaké *fyzikální simulaci*, jsou spuštěny všechny posluchače kolizí v této *fyzikální simulaci* a je jim umožněno, aby podle svého reagovali na tuto kolizi.

### 3.5 Plán

*Plán* je tvořen:

- několika různými *částicemi*
- implicitními *KDaty*

### 3.6 Věc

*Věc* lze označit za konkrétní realizace *plánu*.

- Její *částice* jsou součástí nějaké běžící *fyzikální simulace*. ( Tato *fyzikální simulace* je v nějaké *dimenzi*, co je to *dimenze* viz dále v 3.9).
- Její *KData* mohou být měněna.
- Může být přenesena z jedné *dimenze* do jiné (viz 3.9 a 3.10).

### 3.7 Plán funkce

*Plán funkce* sestává z několika *plánů*.



### 3.8 Kontext

*Kontext* sestává ze dvou seznamů položek indexovaných pomocí textových řetězců, konkrétně:

- seznam několika *KData*
- seznam několika *plánů funkcí*

### 3.9 Dimenze

*Dimenzi* lze označit za konkrétní realizaci *plánu funkce*.

- má *fyzikální simulaci* (nastartovanou s *částicemi* z *plánů* obsažených v *planu funkce*, jehož je realizací)
- má *kontext*

Stav nějaké konkrétní *Dimenze* je to, co je vykreslováno do okna grafického rozhraní simulace. (Více o uživatelském rozhraní v části 2.)

### 3.10 Teleport

*Teleport* je speciální druh *posluchače kolizí*.

- zná *dimenzi* označovanou jako *vstupní* (tj. *dimenze* jejíž *fyzikální simulace* obsahuje tohoto *posluchače kolizí*)
- zná *dimenzi* označovanou jako *výstupní*
- zná *částici* označovanou jako *vstupní brána*

Pokud dojde ke kolizi nějaké *částice* se *vstupní branou*, je *věc*, ke které náleží tato *částice*, přenesena do *výstupní dimenze* (tedy tato *částice* společně s ostatními *částicemi* náležejícími do této *věci* je přenesena do *fyzikální simulace* výstupní *dimenze*).

### 3.11 Funkce

*Funkce* je speciální druh *plánu*, který je velmi důležitý, proto jí popíšeme podrobněji.

Je tvořena třemi *částicemi*:

- *Vstup*
- *Výstup*
- *Tělo*

Její implicitní *KData* jsou její *jméno*.

Navíc oproti normálnímu *plánu*:

- má *plán funkce*

Uvažujme věc  $V$ , kterážto je realizací nějaké *funkce*  $F$ . Při vložení této věci do fyzikální simulace se navíc vloží do této fyzikální simulace i *teleport*, který při kolizi nějaké částice  $\check{c}$  se vstupem provede následující akci:

- Vytvoří nový *kontext*  $K$ , kterýžto je kopie *kontextu dimenze*, ve které se nachází věc  $V$ . A následně tento *kontext* ještě rozšíří o *plány-funkcí funkcí* obsažených v *plánu-funkce funkce*  $F$ .<sup>4</sup>
- Vytvoří novou *dimenzi*  $D$  podle *plánu-funkce funkce*  $F$  a s *kontextem*  $K$ .
- Na samotný vykreslovaný spodek této *dimenze*  $D$  vloží ještě částici *Podlaha*, kterážto bude sloužit jako vstupní brána pro *teleport Zpět*.
- Věc  $VP$ , ke které náleží částice  $\check{c}$ , je přenesena do nově vytvořené *dimenze*  $D$ , konkrétně nahoru doprostřed vykreslované oblasti.
- *Teleport Zpět* funguje obdobně jako právě popisovaný *teleport* s tím rozdílem, že jeho vstupní brána je částice *Podlaha* a přenáší kousek pod výstup funkce  $F$  (respektive přesněji pod částici výstup náležející věci  $V$ , kterážto je realizací funkce  $F$ ). Při teleportování zpět se navíc ukončí provádění *dimenze*  $D$ .

Pro názornější představu toho, jak *funkce* funguje, popíšeme trochu lidštějším jazykem, jak „to vypadá, když něco spadne do funkce“ :

*Funkce* vypadá jako čtvercová „krabíčka“. Pokud něco spadne na její vršek (stane se to „vstupem“ této *funkce*), zmizí to z *dimenze*, ve které to bylo. Aktuální okno začne zobrazovat novou, právě vzniklou *dimenzi*. Ta je postavena podle *plánu funkce* této *funkce*. Nahoře uprostřed se objeví to, co spadlo do *funkce*. Následně se v této *dimenzi* něco děje (v závislosti na tom co v ní je za předměty, tedy v závislosti na *plánu funkce*) až do té doby než něco dalšího spadne na spodek okna *dimenze*. V tuto chvíli se přepne okno znovu na zobrazování původní *dimenze* a vnitřní *dimenze* je ukončena. To co spadlo na spodek *dimenze* (stane se to „výstupem“ této *funkce*) vypadne u spodku „krabíčky“.

### 3.12 Realita

*Realita* je soustava vzájemně propojených *dimenzí*. Dále je velmi důležitá v grafickém výstupu, neboť právě každé jedno okno simulace odpovídá jedné konkrétní *realitě*. Každá *realita*:

- má seznam svých *dimenzí*
- zná *aktuální dimenzi*: to je ta z jejích *dimenzí*, která se právě vykresluje

---

<sup>4</sup>Zde jsme psali plán funkce s pomlčkou aby bylo zřejmé, kde končí a začíná který termín.

- zná *main dimenzi* : to je ta z *dimenzí*, ve které se nastarovala simulace. Ve stavu *STOP* totiž můžeme procházet stromem dimenzí, tím že vstupujeme či vystupujeme z jednotlivých funkcí. Ta dimenze, ve které nastartujeme simulaci se stává main dimenzí. A ačkoliv by se jednalo o vnitřek nějaké funkce, je tato dimenze poslední (tedy pokud jí něco spadne na podlahu, zůstane to ležet místo toho aby to bylo vráceno jako výstup do nadřazené dimenze). Také pokud uložíme stav vynálezu do souboru, uloží se pouze aktuálně zobrazovaná dimenze a její funkce (to znamená, že se neuloží nadřazená struktura).
- má *stav*, který nabývá jedné z hodnot *PLAY* , *PAUSE* , *STOP*
- má okno, do kterého vykresluje *aktuální dimenzi*

## 4 Implementace v jazyku Java

Tato část nás uvádí, v kontextu termínů zavedených v části 3, do implementace programu Kutil v jazyce Java.

**Tuto část napíšu až budu mít hotovou Javadoc dokumentaci programu, jelikož předpokládám že to zní bude velice vycházet.**

## 5 Programy s podobným zaměřením

### 5.1 The Incredible Machine

Myšlenka hry Kutil je inspirována hlavně sérií her The Incredible Machine. Cílem her z této série je řešit různé úkoly. (Úkoly jako např. přesunout míč do košíku, rozsvítit svíčku, odstartovat rachejtli atd.) Úkoly jsou řešeny přidáváním součástek do již rozestavěného stroje z nabídky součástek, které jsou k dispozici pro tento úkol. Ve chvíli, kdy je hráč spokojen se stavem, do kterého přivedl svůj vynález, může odstartovat simulaci a následně vidí, zda se mu úkol povedlo splnit. K dispozici je také svobodný herní mód, kde hráč není omezován počtem součástek, které může použít a také nemá zadán žádný konkrétní úkol, který má splnit.

Hráč má k dispozici velké množství různých součástek, které spolu interagují různými způsoby. Jsou to například: nejrůznější míče; ozubená kola, a další součástky pro manipulaci s rotační energií; lasery; zdi; výbušniny; zvířata s jednoduchým chováním (myš běží za sýrem, když ho vidí, kočka za myší atd.); převodníky různých druhů energie (např: klec s myší na kterou když něco spadne tak začne běhat, což roztáčí kolo, na které jde přidělat guma, kterou lze napojit na ozubené kolo, které se pak točí; nebo naopak krabice na níž jde připojit gumu, která když se točí, tak z krabice vyletí šasek na pérku, který může něco odhodit.) a další součástky.

Kutil se od TIM liší hlavně v tom, na jaký typ součástek klade důraz. Ve hře Kutil jsou klíčové součástky inspirované programovacími konstrukcemi. Tedy

především možnost zavírat už hotové vynálezy do součástí použitelných v jiných vynálezech. Další rozdíl oproti TIM je ten, že v Kutilovi se neklade takový důraz na velikou rozmanitost předmětů. Ze součástí objevujících se v TIM budou důležité hlavně pohyblivé součástky jako míče, statické součástky jako stěny, polo pohyblivé součástky jako jsou různé houpačky a dále asi i ozubená kola a jiné součástky na převádění rotační energie.

Co se týče herních módů, bude Kutil obsahovat oba zmiňované herní módy hry TIM.

## 5.2 Další podobné hry

Zde je seznam několika dalších her, které již nejsou tolik podobné Kutilovi, jako je TIM.

- Phun
- Crayon Physics
- Widget Workshop
- Armadillo Run

# 6 Využívané zdroje

## 6.1 phys2d

Pro fyzikální simulaci je používána jednoduchá knihovna phys2d. Tato knihovna je poskytována pod licencí *BSD License*. [phys2d]

## 6.2 XmlWriter

Pro pohodlnější výstup do formátu XML byla použita knihovna XmlWriter. [XmlWriter]

## 6.3 Metoda aktivního vykreslování

V programu je použita metoda aktivního vykreslování popsaná v knize [Davison].

# 7 OS, jazyk, vývojové prostředí

Program je napsán v jazyku Java, vyvíjen v prostředí NetBeans IDE a je nezávislý na platformě uživatele. Obrázky použité v grafice programu jsou nakresleny v programu Macromedia Flash 8.

## 8 Co zbývá dodělat a možná rozšíření programu

### 8.1 Nejvyšší priorita

- Javadoc dokumentace

### 8.2 Střední priorita

- Stavění strojů Kutila na základě výrazů Lambda kalkulu, za využití převodu na kombinátory z combinatory logic (**TODO** něco víc o tomhle napsat)
- Podpora pro herní prvky (tzn. hlavně možnost tvořit úkoly pro hráče a možnost omezit paletu přístupných věcí)

### 8.3 Nižší priorita

- Rozšířené možnosti pro práci s panáčkem (po stisknutí klávesy X „vsát“ do panáčka nejbližší pohyblivý předmět atd.) tak aby při využití podpory herních prvků a této rozšířené práce s panáčkem bylo možné dělat jednoduché hříčky.
- *Stroj času* : to je součástka, která má dva ovládací prvky: "save-tlačítko" a "load-vstup". Reakcí na událost, která provede save je, že se zaznamená současný stav celého vynálezu (variantou je, že není žádné save-tlačítko ale místo toho se save provede automaticky při startu vynálezu). Load-vstup je podobný vstupu u funkce, s tím rozdílem, že při "vstupu objektu do stroje času" není vytvořena nová dimenze s nějakým vynálezem, ale místo toho je restartována současná dimenze do pozice, ve které byla uložena strojem času. Tato pozice se liší od původně uloženého stavu o to, že nyní navíc obsahuje objekt, který vstoupil do stroje času. Stroj času představuje jakousi analogii s cyklem v programovacím světě, přičemž na objekt, jež cestuje časem, se můžeme dívat jako na jistou formu iterátoru.
- Podpora pro uživatelem definované (napsané v jazyku Java) *elementární funkce*.
- Přejmenování *uživatelských funkcí*.

## Reference

- [Davison] Davison, Andrew; (2006), Programování dokonalých her v Javě
- [introLC] Barendregt, Henk; Barendsen, Erik (March 2000), Introduction to Lambda Calculus
- [tutorLC] Rojas, Raul; (1997/98), A Tutorial Introduction to the Lambda Calculus

- [phys2d] Phys2D - a 2D physics engine based on the work of Erin Catto.  
<http://www.cokeandcode.com/phys2d/>
- [XmlWriter] XmlWriter - A Writer style API for outputting XML by Henri Yandell.  
<http://www.generationjava.com/projects/XmlWriter.shtml>