Kurz LSL skriptování

Shiny Iceberg

2009

součást cyklu **Nejsme jelita**

Kurz LSL skriptování

Shiny Iceberg

v Second Life od roku 2006

shiny.iceberg@virtualmagazine.cz

Aktuální projekty

virtualmagazine.cz

Urbanica, Shinyland

Bwindi Orphans

cyklus Nejsme jelita

Organizační body

Průběh lekce

- bude trvat zhruba 90 minut
- pokud nekladete dotazy, vypněte si mikrofon
- příklady si klidně zkoušejte přímo v hledišti

Vaše otázky

- můžete se ptát na konci každého snímku nebo na konci celé přednášky
- dotazy mohou být přes voice nebo IM
- při psaní IM z posledních řad použijte Shout

To nejdůležitější z minula

- skript pracuje v objektu a umožňuje interakci s okolím, avatary, jinými objekty či s Internetem
- pokud je objekt se skriptem v inventory, je skript pozastaven, ale pamatuje si aktuální State
- skript se skládá z States, Events, Variables, Constants, Flow Control, Functions, Operators
- States reprezentuje stav objektu, Events událost, která nastala

Plán přednášky

1. Fungování a struktura skriptu
2. Z čeho se skládá skript
3. Vlastnosti objektu
4. Pohyb objektu
5. Pose bally
6. Komunikace skriptu
7. Inventory objektu
8. Detektory
9. Particles
10. Příklady a dokumentace

Z čeho se skript skládá

Každý **skript** obsahuje několik základních elementů

- STATES – stav
- EVENTS – událost
- VARIABLES – proměnné různých typů
- CONSTANTS – konstanty
- FLOW CONTROL – řízení běhu skriptu
- FUNCTIONS – definované kusy kódu
- OPERATORS – znaky operace

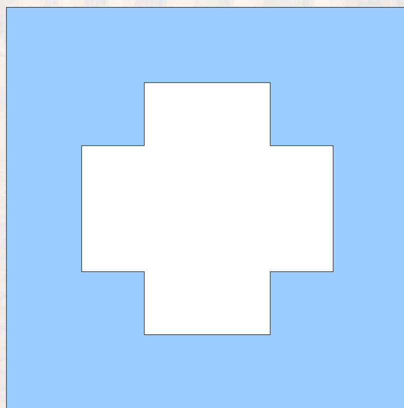
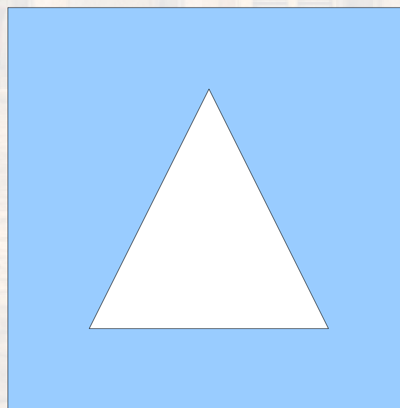
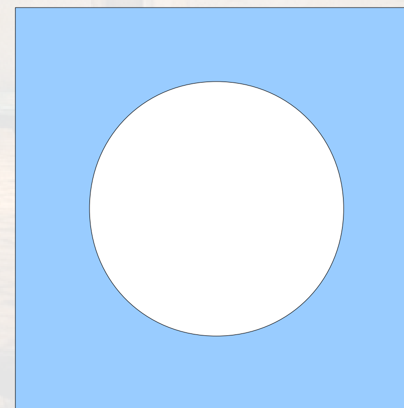
Láska na první pohled

```
default
{
    state_entry()
    {
        llSay(0, "Dobry den!");
    }

    touch_start(integer total_number)
    {
        string Text = "Kliknul jsi" + " na mne.";
        if ( SQRT2 > 2 )
        {
            llSay(0, Text );
        }
    }
}
```

1. STATES
2. EVENTS
3. VARIABLES
4. CONSTANTS
5. FLOW CONTROL
6. FUNCTIONS
7. OPERATORS

Variables (proměnné)

- proměnná - funguje jako krabice s různě tvarovaným výřezem na horní straně
- můžete do ní vložit a nebo z ní vyndat objekt (hodnotu) pouze určitého typu

Variables (proměnné)

Typy proměnných

- *integer* celé číslo
- *float* reálné číslo
- *string* řetězec textu
- *key* identifikátor objektů, textur či avatarů
- *vector* skupina tří čísel
- *rotation* speciální proměnná pro rotování
- *list* seznam proměnných různých typů

Kód s proměnnými

```
default
{
    state_entry()
    {
        string MojePromenna;

        MojePromenna = "První text";

        llOwnerSay(MojePromenna);

        Moje Promenna = "Druhý text";

        llOwnerSay(MojePromenna);
    }
}
```

→ Vypíše : **První text**

→ Vypíše : **Druhý text**

Počáteční hodnota

```
default
{
    state_entry()
    {
        string MojePromenna = "První text";

        llOwnerSay(MojePromenna);

        Moje Promenna = "Druhý text";

        llOwnerSay(MojePromenna);
    }
}
```

Vypíše : **První text**

Vypíše : **Druhý text**

Práce s proměnnými

- Jméno proměnné může obsahovat písmena, čísla (ale ne na začátku) a znak _, ostatní znaky jsou ignorovány, maximální délka 256 znaků

- Většina funkcí v LSL používá proměnné jako vstupy pro svou činnost

- Technika *typecasting* převádí proměnné ruzných typů mezi sebou

Typecasting

- převod mezi různými typy hodnot se provádí uvedením nového typu hodnoty před proměnnou

```
string MujText = "Zkušební text";
integer MojeCislo = 5;
string DruheCislo = (string)MojeCislo;

llOwnerSay( MujText );
llOwnerSay( (string)MojeCislo );
llOwnerSay( DruheCislo );
```

Globální vs. lokální

Globální proměnné

- definované před state *default*
- platí v celém skriptu, ve všech state
- pro jejich hodnotu nemůže být použit *výraz*

Lokální proměnné

- definované kdekoliv mezi { a } - uvnitř state, event nebo jen bloku
- má platnost pouze v tomto bloku
- hodnotu lze naplnit *výrazem*

Globální vs. lokální

```
string Autor1 = "Shiny Iceberg";
string Region1 = llGetRegionName();
string Region2;

default
{
    state_entry()
    {
        Region2 = llGetRegionName();
        string Autor2 = "Nejsme jelita";
    }

    touch_start(integer total_number)
    {
        llOwnerSay( Autor1 );
        llOwnerSay( Autor2 );

        llOwnerSay( Region1 );
        llOwnerSay( Region2 );
    }
}
```

globální

lokální

ŠPATNĚ (string Region1 = llGetRegionName();)

ŠPATNĚ (llOwnerSay(Autor2);)

Variable *integer*

- popisuje celé číslo (bez desetinné čárky) v rozpětí
  -2147483648 až 2147483647 (32 bit)

- pro fajnšmekry: mohou být vkládány i čísla v
  hexadecimálním tvaru (např. 0x2bf)

```
integer Test1 = -21;

integer Test2 = 123456;

integer Test3 = 0x2fd00;
```

Variable *float*

- popisuje reálné číslo s plovoucí desetinnou čárkou, zhruba ± $\sim 10^{-44.85}$ až $\sim 10^{38.53}$ (32 bit)
- operace s proměnnou *float* způsobuje zaokrouhlování (zanedbatelné)
- pro fajnšmekry: mohou být vkládány i čísla v exponenciálním zápisu (např. 1.123E-2)

```
float test1 = -3.87745;

float test2 = 12.0;
```

Variable *string*

- obsahuje řetězec znaků, délka je omezena jen celkovou pamětí pro skript (16 kB)
- řetězec se při zadávání ohraničuje znakem “
- speciální znaky \t (4 mezery), \n (nový řádek), \“ (uvozovky), \\ (lomítko samotné)

```
string test1 = "Tento text se \n vypíše na dvě řádky!";

string test2 = "Nový řádek se vytvoří pomocí znaku \\ a písmene n";
```

Variable *key*

- obsahuje unikátní identifikátor (UUID) objektu, avatara, skupiny či položky v inventory ve tvaru 00000000-0000-0000-0000-000000000000

- UUID se může měnit při vytváření kopií a nebo
  - notecard: při editaci
  - objekt: při rezznutí

```
key NejsmeJelita = "a83f9888-9607-d9cc-b471-83e7c041fc3d";

key Notecard = "a6979a7a-01c9-4f62-5881-3e886ac0b37c";
```

Variable *vector*

- jde o tři čísla typu *float* spojené dohromady ve tvaru < x , y , z > a tato trojice umožňuje vyjádřit například polohu, rychlost, barvu, měřítko atd.

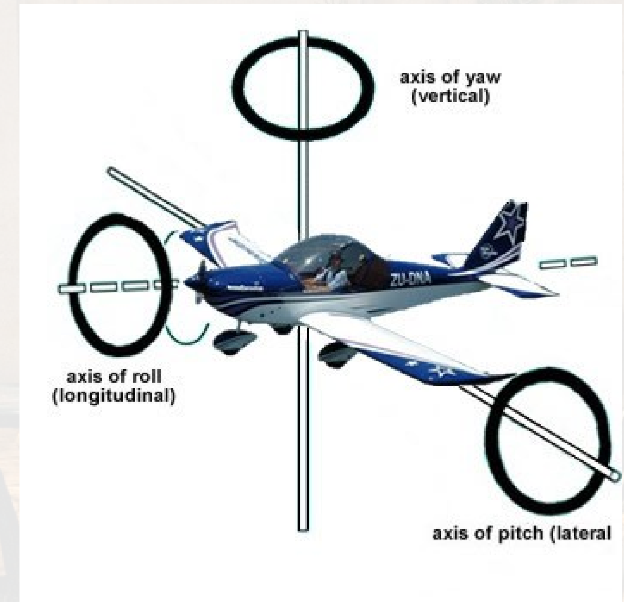
- pomocí tečkové notace mohu měnit jednotlivé složky vektoru, viz příklad

```
vector Poloha = <122.7, 98.8, 24.5>;

Poloha.z = 28.1;
```

Variable *rotation*

- rotace není v LSL vyjádřena pomocí klasického otoření podle os X, Y a Z, ale pomocí takzvaných kvaternionů - tvoří ji 4 hodnoty <X, Y, Z, S>

- naštěstí lze klasickou XYZ rotaci jednoduše převádět na a z kvaternionů pomocí dvou funkcí llEuler2Rot a llRot2Euler



- podrobnější vysvětlení bude uvedeno u funkcí pro rotaci objektu

Variable *list*

- jde o seznam (až 1000+) hodnot různých typů, zapisovaných mezi znaky [a] a oddělených čárkou, výhodné pro generované seznamy s předem neznámou délkou

- na čtení, hledání, přidávání, mazání nebo třídění jednotlivých položek se používají speciální funkce, např. *llList2Integer* nebo *llListSort*

```
list test1 = [ "somestring" , 12 , 3.0 , <3.3,4,0> , <0,0,0,1>];

list visitors = [   "Ama Omega",        "2009-06-12",
                    "Catherine Omega", "2009-06-28",
                    "Ezhar Fairlight", "2009-06-30"
                ];
```

Konstanty

- v podstatě jde o „proměnnou se stále stejnou hodnotou“
- pomáhají dělat kód skriptu čitelnější
- většina konstant je definována v rámci funkce
- některé konstanty, např. *PI, SQRT2, TRUE, FALSE, ZERO_VECTOR či DEG_TO_RAD* jsou všeobecně platné
- příklad použití:

```
float UhelDeg = 30.0;             // reprezentuje 30 stupňů
float UhelRad = UhelDeg * DEG_TO_RAD;   // 0.52 radiánů
```

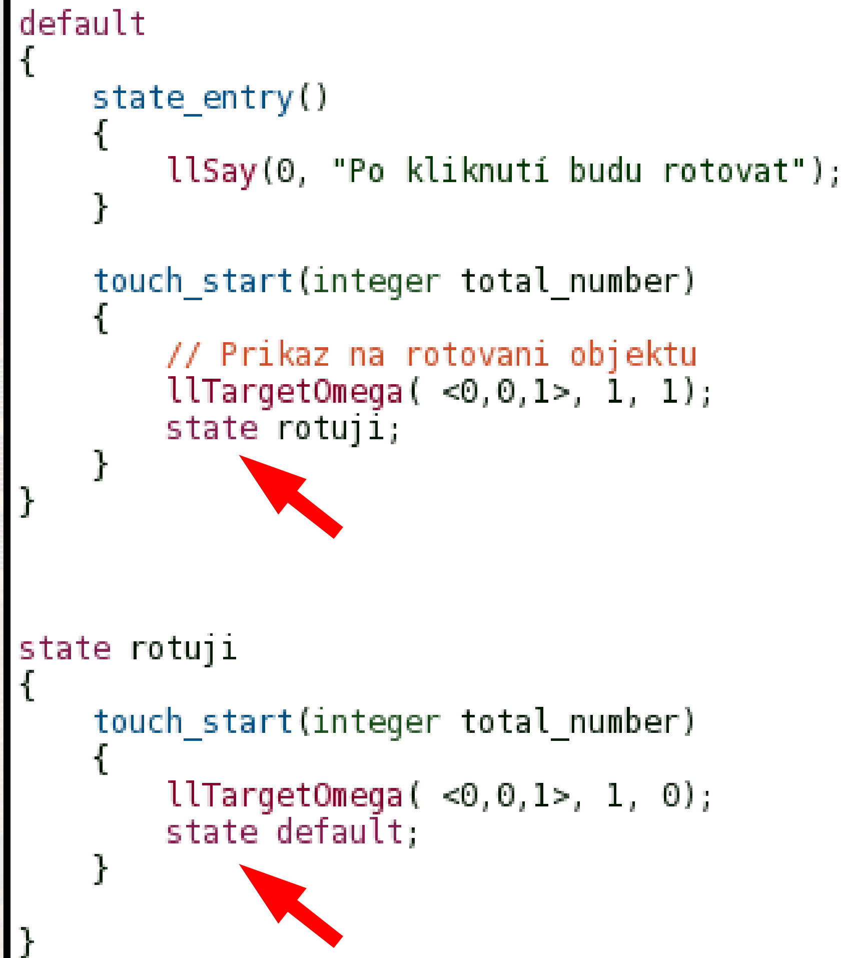
Kontrola běhu skriptu

- existuje několik příkazů, které řídí způsob, jak, kdy a co se ve skriptu vykonává

- **state**	přechod do jiného state
- **if … else**	větvení podle podmínky
- **for**	cyklus s daným počtem opaková
- **while**	cyklus, testování na začátku
- **do … while**	cyklus, testování na konci

state

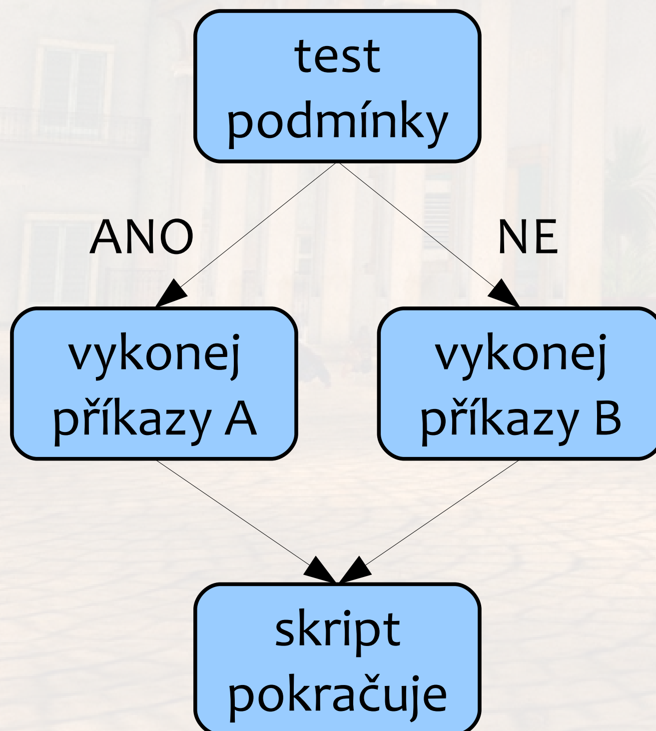
- již jsme probírali v 1. lekci
- slouží na změnu aktuální *state*, který vyjadřuje stav objektu



```
default
{
    state_entry()
    {
        llSay(0, "Po kliknutí budu rotovat");
    }

    touch_start(integer total_number)
    {
        // Prikaz na rotovani objektu
        llTargetOmega( <0,0,1>, 1, 1);
        state rotuji;
    }
}


state rotuji
{
    touch_start(integer total_number)
    {
        llTargetOmega( <0,0,1>, 1, 0);
        state default;
    }
}
```

if … else

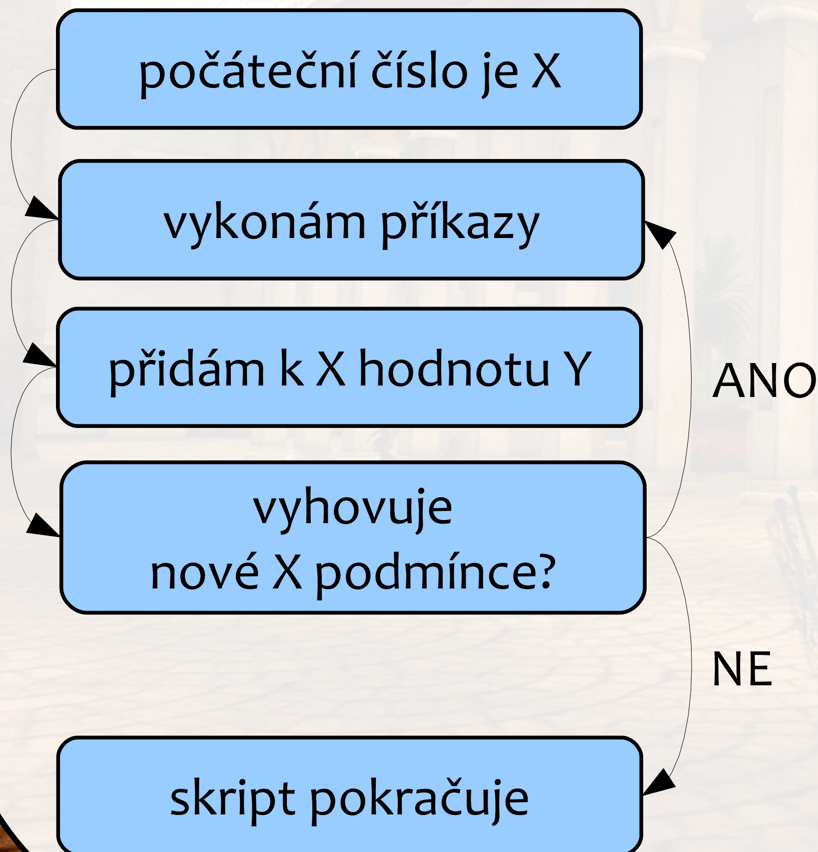
- lze vyjádřit jako: *pokud je něco pravda, dělej A, v opačném případě dělej B (anebo nic)*




```
default
{
    state_entry()
    {

        string Region = llGetRegionName();

        if ( Region == "Shinyland" )
        {
            llOwnerSay("Jsem doma");
        } else {
            llOwnerSay("Chci domů");
        }

    }
}
```

for

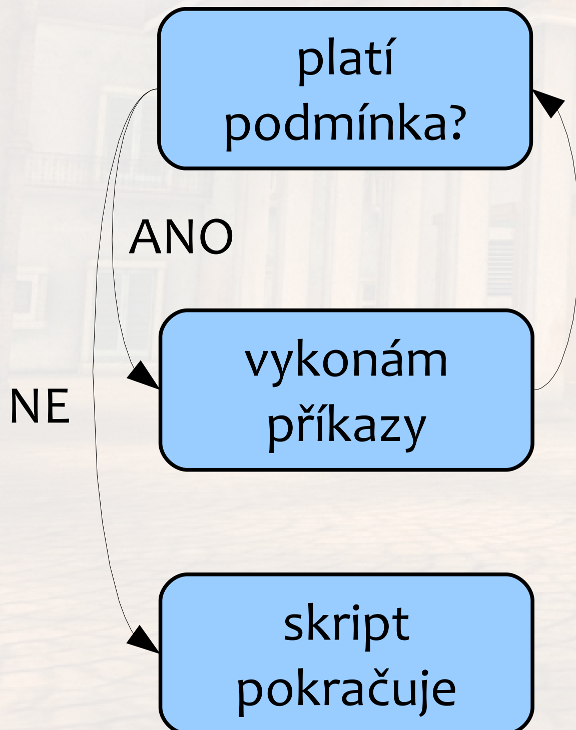
- lze vyjádřit jako: *vykonej daný blok příkazů opakovaně x-krát za sebou*




```
default
{
    state_entry()
    {

        integer N;
        list Ovoce = ["jablko", "hruška", "švestka"];

        integer Druhy = llGetListLength(Ovoce);

        for (N = 0; N < Druhy; N = N + 1)
        {
            llOwnerSay( llList2String(Ovoce, N) );
        }


    }
}
```

while

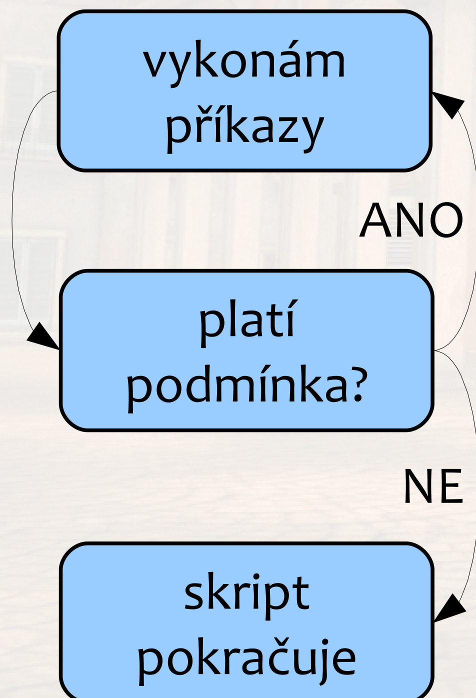
- lze vyjádřit jako: *dokud platí podmínka, vykonávej opakovaně blok příkazů*




```
default
{
    state_entry()
    {
        vector Slunce = llGetSunDirection();

        while( Slunce.z <= 0 )
        {
            llOwnerSay("Čekáme na ráno");
            llSleep(300);
            Slunce = llGetSunDirection();
        }

        llOwnerSay("Dobrý den světe");
    }
}
```

do … while

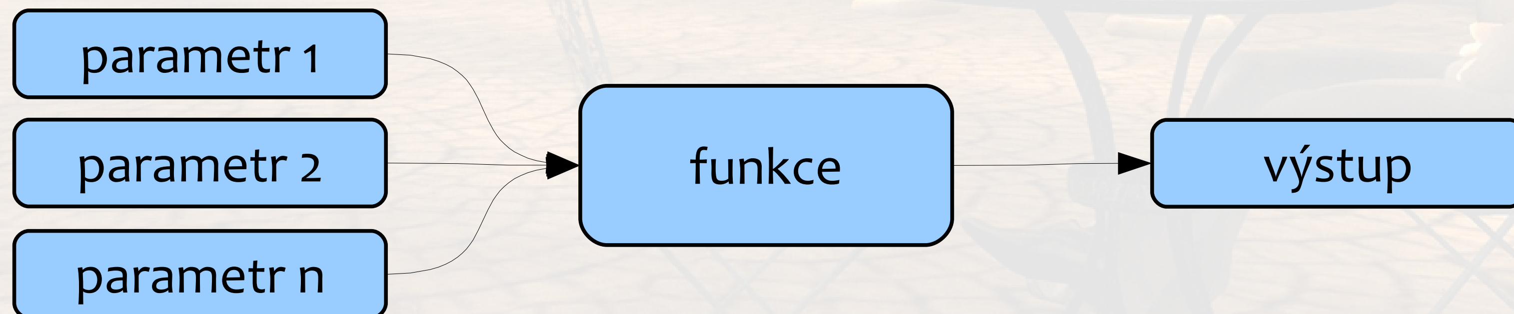
- proběhne alespoň 1x, lze vyjádřit jako: *vykonávej opakovaně blok příkazů, dokud platí podmínka*




```
default
{
    state_entry()
    {
        vector Slunce = llGetSunDirection();

        do
        {
            llOwnerSay("Čekáme na ráno");
            llSleep(300);
            Slunce = llGetSunDirection();
        }
        while( Slunce.z <= 0 );

        llOwnerSay("Dobrý den světe");
    }
}
```

Funkce - předdefinované

- několik set funkcí je již součástí LL, poznáte je podle názvu funkce *llNazevFunkce*
- v kódu je použiji pomocí jména funkce a vstupních parametrů
- vstupní parametry i výstup mohou být prázdné, např. *llOwnerSay* (pouze vstup) nebo *llGetSunDirection* (pouze výstup) či *llDie* (zcela bez parametrů)



Funkce - předdefinované

- kompletní seznam funkcí a vstupních / výstupních parametrů je k nalezení na LSLwiki na *http://www.lslwiki.net/lslwiki/wakka.php?wakka=functions*

- několik příkladů:

```
default
{
    state_entry()
    {

        llSleep(10);
        llSay(0, "Dobrý den světe");

        vector BarvaStrany = llGetColor(0);
        list JmenoParcely = llGetParcelDetails( llGetPos() , [PARCEL_DETAILS_NAME] );

        llSay(0, (string)JmenoParcely);
    }
}
```

Funkce - uživatelské

- kromě předdefinovaných funkcí je možné vytvořit i své vlastní podle stejného principu vstup - výstup
- tato technika se používá hlavně na opakovaně používanou sekvenci příkazů
- účelem je úsporný zápis kódu, který je rozdělen do menších a lépe přehledných bloků
- definice uživatelské funkce musí obsahovat jméno funkce, definici vstupních parametrů, sekvenci příkazů a definici výstupní hodnoty, definuje se před *state*

Funkce - uživatelské

typ vrácené hodnoty

jméno funkce

vstupní parametry

```
string TestAlkoholika ( float LitruPiva , integer Panaku )
{
    float Alkohol = LitruPiva + (float)Panaku / 2;
    if (Alkohol > 4.0)
    {
        return "Ty ji máš jak z praku";
    } else {
        return "Objednej ještě rundu";
    }
}

default
{
    state_entry()
    {
        string Pijan1 = TestAlkoholika(0.5, 0);
        string Pijan2 = TestAlkoholika(2.5, 2);
        string Pijan3 = TestAlkoholika(4.5, 6);

        llOwnerSay(Pijan1);
        llOwnerSay(Pijan2);
        llOwnerSay(Pijan3);
    }
}
```

definice funkce

vrácení hodnoty

Operátory

- matematické a logické operace s hodnotami, zde jsou uvedeny jen ty nejdůležitější

```
float a = a + 3.6 * 0.5;
```

přiřazení hodnoty, sčítání a násobení, obdobně - či /

```
if (a == 2.8)  { příkazy }
```

porovnávání hodnot, obdobně != pro nerovnost, adekvátně <, >, <=, >=

```
if ( (a > 2.1) && (a < 4.3) )  { příkazy }
```

dvojznak || znamená NEBO
dvojznak && znamená A ZÁROVEŇ

Otázky a diskuze