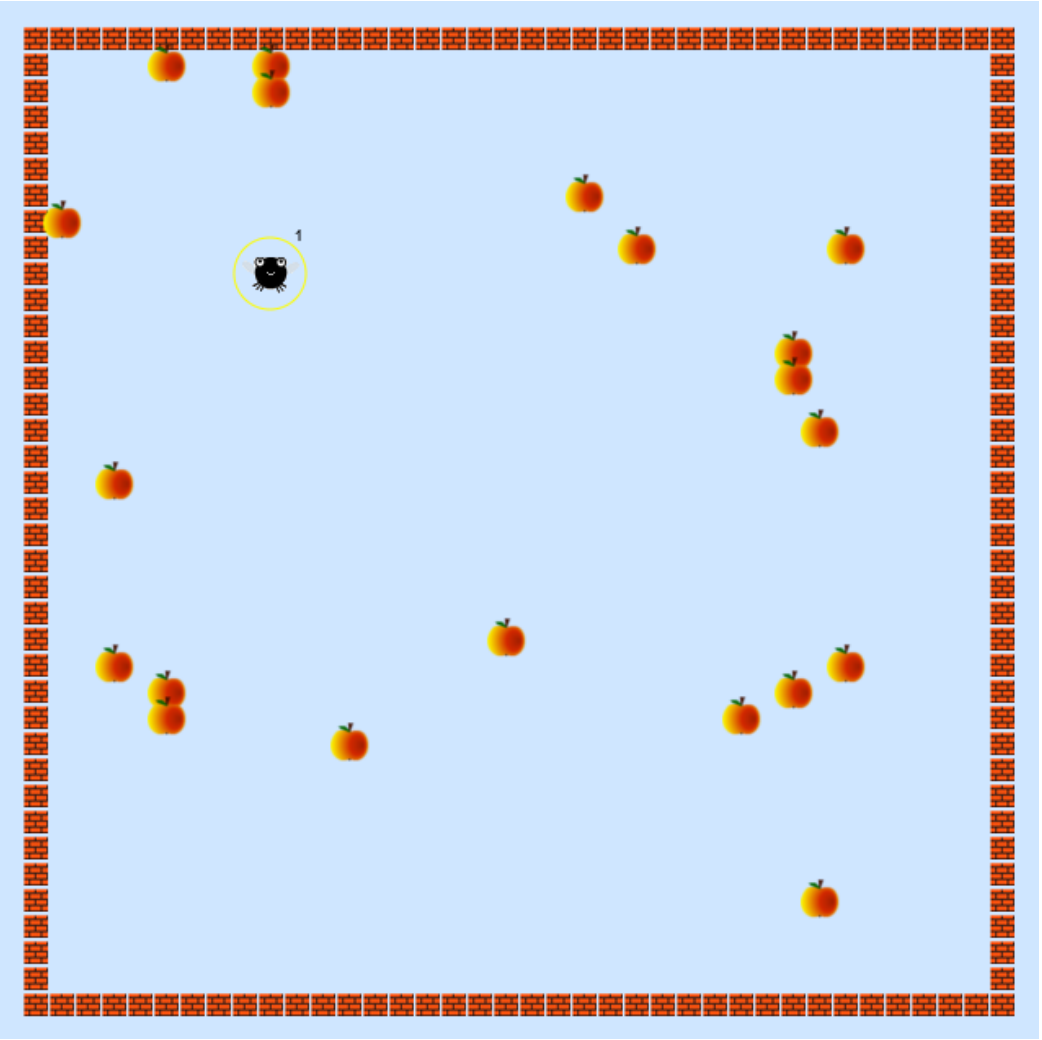
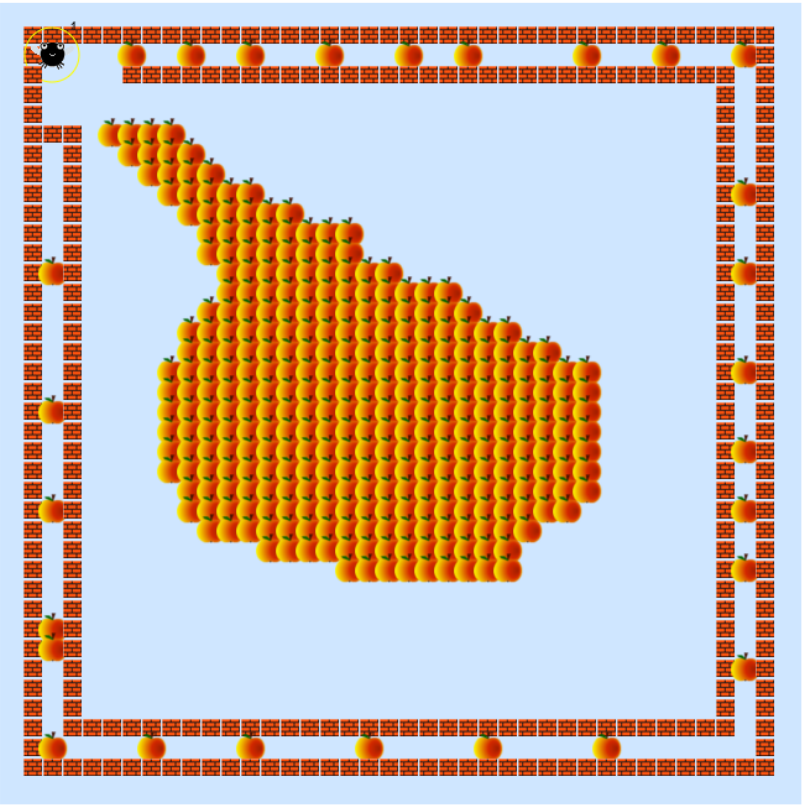
Mouchy a Jablka



Tomáš Křen

Pravidla vesmíru much



- Tři druhy hmoty
  - Moucha
  - Jablko
  - Zeď
- Mouchy a jablka mají energii
- Moucha má dva druhy tahu
  - Přesun (určen směrem)
  - Rozdělení (odštěpí se zní dcera)
    - Určeno směrem, energii dcery a vnitřním stavem dcery

Přesun

- Při přesunu na políčko s jablkem moucha jablko sní a přičte si jeho energii
- Při přesunu na políčko s mouchou přežije ta s větší energií a získává součet jejich energií
- Při přesunu na políčko se zdí se přesun neprovede

Fitness

- Součet energií mouchy a jejích potomků po předem stanoveném počtu kroků
- Přes několik testovacích levelů

Použitá metoda : Typované GP

- Používám mnou navržený systém na typované funkcionální genetické programování
  - Jedinci termy (=programy) v typovaném lambda kalkulu
  - Vstup : fitness funkce a množina otypovaných symbolů konstant / funkčních symbolů (stavební bloky programů)
  - Důraz na algoritmus generování termů
    - Určeno prohledávácí strategií
      - Umožňuje jak systematické prohledávání tak zobecnění klasického Kozovského prohledávání
  - Křížení umožněné díky převodu termů do SKI kombinatorického kalkulu (tím zmizí proměnné)

Vstupy a výstupy mouchy

- Vstup
  - Minulý směr pohybu
  - Energie mouchy
  - Zda se povedla minulá akce
  - Užitečné předpřipravené info
    - Nejbližší jablko
      - Jeho energie, směr a vzdálenost
    - Nejbližší moucha (...)
    - těžiště jablek (...)
  - Vnitřní stav (sada registrů)
  - Různé další
- Výstup
  - Tah
    - Přesun
    - Rozdělení
  - Následující vnitřní stav

Stavební bloky

```
( "output_"          , move_typ :-> regs_typ :-> output_typ        ) ,

( "if'"              , bool_typ :-> output_typ :-> output_typ :-> output_typ ),
( "if'"              , bool_typ :-> move_typ :-> move_typ :-> move_typ ),
( "if'"              , bool_typ :-> dir_typ :-> dir_typ :-> dir_typ ),
( "if'"              , bool_typ :-> int_typ :-> int_typ :-> int_typ ),
( "if'"              , bool_typ :-> dist_typ :-> dist_typ :-> dist_typ ),
( "if'"              , bool_typ :-> regs_typ :-> regs_typ :-> regs_typ ),



( "easySplit"        , input_typ :-> move_typ ),
( "travel_"          , dir_typ :-> move_typ                        ) ,
( "split_"           , dir_typ :-> int_typ :-> regs_typ :-> move_typ ),


( "myEnergy_"        , input_typ :-> int_typ                        ),
( "myLastTravel_"    , input_typ :-> dir_typ                       ) ,
( "myWasSuccess_"    , input_typ :-> bool_typ                      ),


( "nAppleDir_"       , input_typ :-> dir_typ                       ),

( "nAppleDir_"       ,  input_typ :-> dir_typ                      ),
( "nAppleDist_"      ,  input_typ :-> dist_typ                     ),
( "nAppleEnergy_"    ,  input_typ :-> int_typ                      ),
( "nFlyDir_"         ,  input_typ :-> dir_typ                      ),
( "nFlyDist_"        ,  input_typ :-> dist_typ                     ),
( "nFlyEnergy_"      ,  input_typ :-> int_typ                      ),

( "cAppleDir_"       , input_typ :-> dir_typ                        ),
( "cAppleDist_"      , input_typ :-> dist_typ                       ),

( "myRegs_"          , input_typ :-> regs_typ                      ) ,

( "xGet_"            , input_typ :-> int_typ                      ),
( "yGet_"            , input_typ :-> int_typ                      ),
( "zGet_"            , input_typ :-> int_typ                      ),
( "dGet_"            , input_typ :-> dir_typ                      ),

( "xSet_"            , int_typ :-> regs_typ :-> regs_typ          ),
( "ySet_"            , int_typ :-> regs_typ :-> regs_typ          ),
( "zSet_"            , int_typ :-> regs_typ :-> regs_typ          ),
( "dSet_"            , dir_typ :-> regs_typ :-> regs_typ          ),

( "xInc_"            , regs_typ :-> regs_typ                      ),
( "yInc_"            , regs_typ :-> regs_typ                      ),
( "zInc_"            , regs_typ :-> regs_typ                      ),


( "rotCW_"           , dir_typ :-> dir_typ                         ) ,

( "dUp"              , dir_typ                                     ) ,
( "dDown"            , dir_typ                                     ) ,
( "dLeft"            , dir_typ                                     ) ,
( "dRight"           , dir_typ                                     ) ,

( "(==)" , int_typ :-> int_typ :-> bool_typ ),
( "(<=)" , int_typ :-> int_typ :-> bool_typ ),

( "0"                , int_typ ),
( "1"                , int_typ ),
( "2"                , int_typ )
```

Zvolený postup

- Nejprve jsem vytvořil simulátor prostředí a určil si vstupy a výstupy programu řídícího mouchu
- Pak jsem zkusil napsat ručně několik různě složitých programů pro mouchu
- Všiml jsem si jaké funkce jsem využil napříč programy a ty jsem použil jako množinu stavebních bloků
- Na základě obdržených řešení jsem dále ztěžoval fitness přidáváním levelů tak aby je tyto mouchy řišili neefektivně (Ďábel)

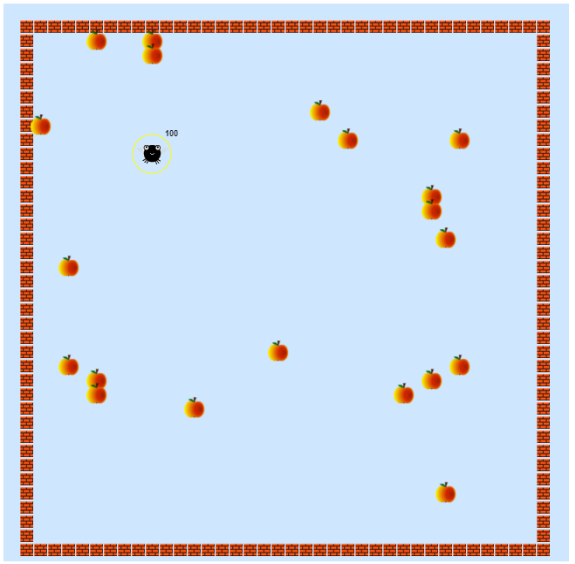
Pokus 1

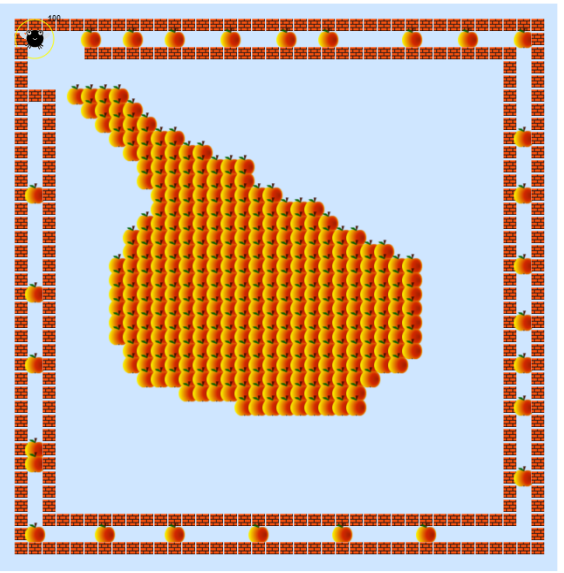
- 25 generací populace 1000 jedinců 5 běhů
- Okolo 10-té generace mouchy často „zmoudří“
- ALE : Nemnoží se
- Proto v pokusu 2 trochu množení napomůžeme

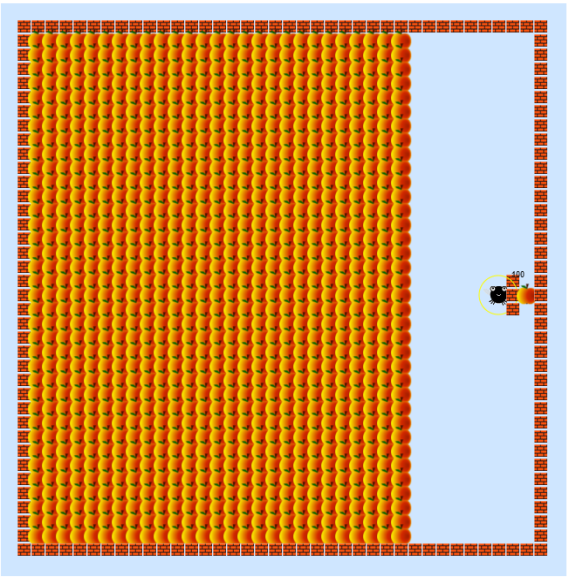
Pokus 2

- 30 gen 1000 pop 5 běhů
- Počáteční energie mouchy je nyní 100 místo 1
  - Aby si nemusela složitě šetřit než se může smysluplně množit
- Přidávám jedenu funkci do stavebních bloků – méně obecný konstruktor tahu dělení (vždy rodí dolu, vždy dá dceři polovinu energie)
- Důsledek: mouchy se opravdu začnou množit a chovají se zajímavěji – což je cílem :)

Ukázka...

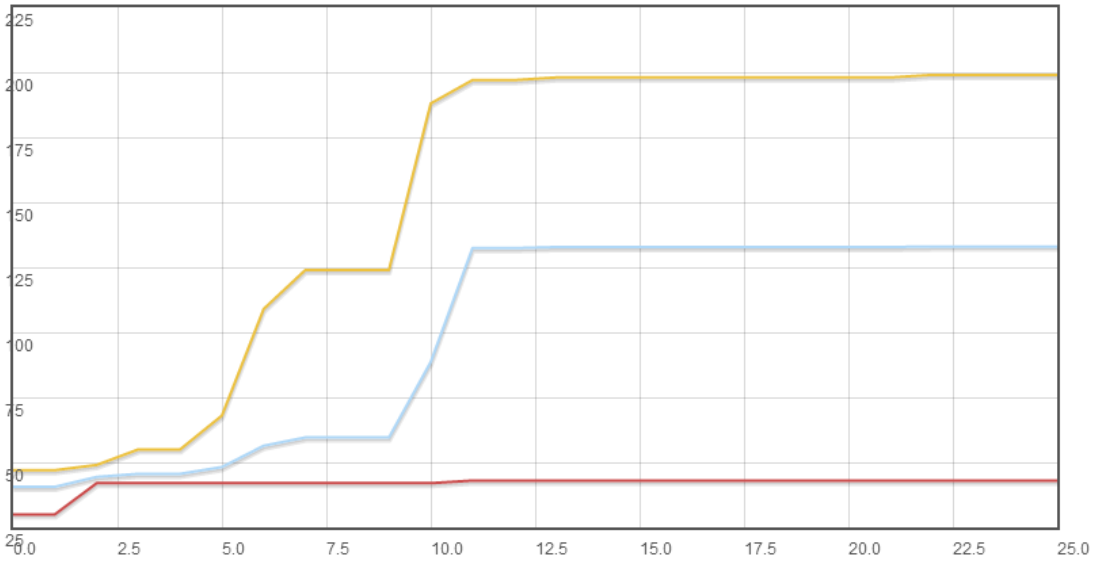




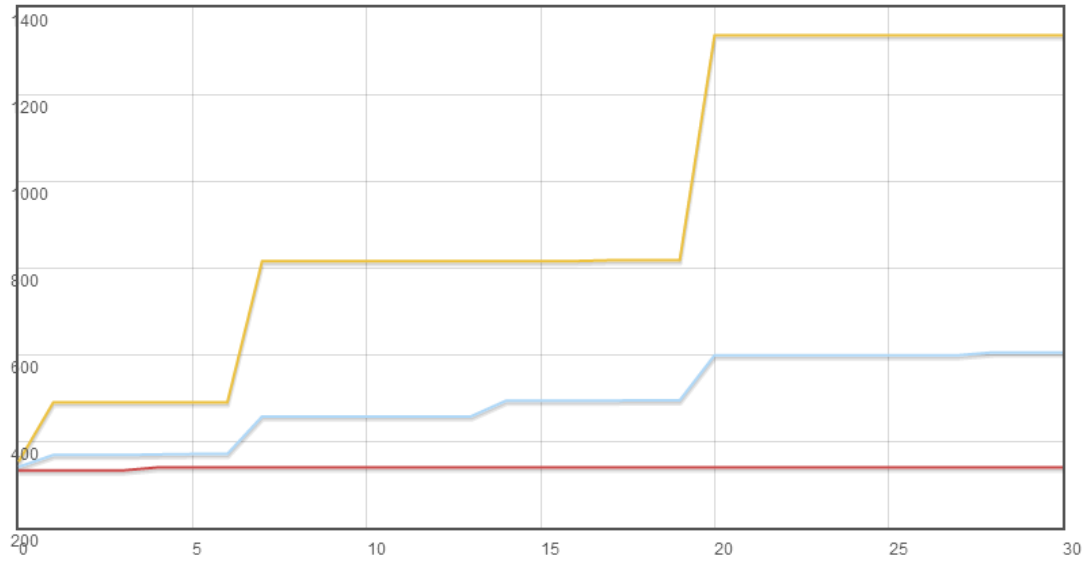


Grafy

Pokus1



Pokus2



Napříč pěti běhy sledován vývoj nejlepšího jedince

Žlutá – nejlepší z nejlepších v jednotlivých generacích

Modra – průměr z nejlepších

Červená – nejhorší z nejlepších

Ukázky kódu jedinců

(Jeden z povedených běhů v druhém pokusu)

```
 \ x0 -> if' (myWasSuccess_ x0) (output_
(travel_ (rotCW_ dDown)) (zSet_ (if' ((<=) 2
(myEnergy_ x0)) 1 1) (yInc_ (yInc_ (yInc_
(myRegs_ x0)))))) (output_ (travel_ (nAppleDir_
x0)) (dSet_ dDown (xSet_ 1 (myRegs_ x0))))
```

```
 \ x0 -> if' (myWasSuccess_ x0) (output_
(easySplit x0) (ySet_ (nAppleEnergy_ x0) (yInc_
(dSet_ dLeft (myRegs_ x0))))) (output_ (travel_
dLeft) (myRegs_ x0))
```

```
  \ x0 -> if' (myWasSuccess_ x0) (output_
(split_ dUp (myEnergy_ x0) (myRegs_ x0))
(myRegs_ x0)) (output_ (travel_ (if' ((<=)
(yGet_ x0) 1) dLeft (rot
 CW_ dDown))) (xSet_ 2 (xInc_ (myRegs_ x0))))
```

```
 \ x0 -> if' (myWasSuccess_ x0) (output_
(split_ dUp (myEnergy_ x0) (myRegs_ x0))
(myRegs_ x0)) (output_ (travel_ (dGet_ x0))
(dSet_ dLeft (dSet_ (dGet_ x0) (zSet_ 2
(xInc_ (myRegs_ x0))))))
```

```
\ x0 -> if' (myWasSuccess_ x0) (output_
(easySplit x0) (zInc_ (myRegs_ x0) ))
(output_ (travel_ (if' ((<=) (yGet_ x0) 1)
dLeft (if' ((<=) (yGet_ x0) 1) dLeft
(nAppleDir_ x0)))) (xSet_ 2 (xInc_ (xInc_
(yInc_ (zInc_ (xSet_ (yGet_ x0) (xInc_
(dSet_ dLeft (myRegs_ x0)))))))))))
```

Je sympatické, že tyto kódy ani netrpí přílišným bloatem.