Šlechtění řídící funkce pro hru Had pomocí genetického programování

Semestrální práce pro předmět **BI-ZUM** Autor: **Ondřej Šlejtr**

Téma

Semestrální práce se zabývá vyšlechtěním co nejefektivnější řídící funkce pro hru Had, v níž hráč (v našem případě vyšlechtěná funkce) na ohraničené obdélníkové ploše specifikované šachovnicí polí ovládá hada, který má za úkol nasbírat co nejvíce na ploše náhodně vytvořených kusů jídla, přičemž v jeden okamžik je na ploše pouze jeden kus jídla. Had se na ploše může pohybovat po řadách a sloupcích), nikoliv diagonálně, a hra končí ve chvíli, kdy had buďto narazí na hranici hrací plochy nebo do svého “ocasu”.

Obtíž spočívá v tom, že délka hada (nebo jeho “ocasu”) se s každým sebraným jídlem zvětšuje a postupně tak samotný had obsazuje stále větší a větší herní prostor, což značně omezuje manévrovací prostor a schopnost efektivně sbírat další jídlo.

Zvolená metoda vytvoření řídící funkce spočívá v konceptu genetického programování, které má za cíl vytvořit strom, jehož ohodnocení v kontextu aktuální herní situace by mělo dát hadovi co nejlepší možnou odpověď na to, zda při dalším herním tahu změnit svoje směřování (otočit se doprava nebo doleva), či zda pokračovat na zvolené přímé linii. Tyto stromy jsou postupně iterativně mutovány a kříženy, se základním kritériem vhodnosti zvoleným podle domény problému (zde podle počtu snědených jídel).

Velikost hrací plochy není vždy stejná.

Vybrané a zpracované řešení

Cílem řešení bylo vytvořit co nejkvalitnější řídící funkci, přesněji řečeno binární strom, který ve svých listech obsahuje terminály a v ostatních uzlech neterminály. Tento binární strom je ohodnecen podle níže specifikovaného fitness kritéria a pomocí algoritmu specifikovaného ve funkci evolve() dále iterativně upravován (podle uvedených parametrů).

Obecné fitness kritérium: Nasbírat co nejvíce kusů jídla před naražením do hrany hrací plochy, do sebe nebo před uběhnutím maximálního počtu tahů (zde zvoleného pro zamezení zacyklení).

Standartizace fitness kritéria: Jelikož každý běh je do značné míry randomizovaný (počáteční pozice hada, spawny jídla), zvolil jsem pro výpočet fitness řídící funkce průměr ze 4 jejích nezávislých běhů.

Parametry:

- **velikost hrací plochy**: ačkoliv může být obecně libovolná, pro standartizace fitness kritéria jsem zvolil plochu 10x10
- **maximum tahů**: 1000
- **počet generací**: 450
- **velikost populace**: 500
- **mutation:** 15%
- **crossover**: 80%
- **elitness**: 20%

Neterminály (funkce) stromu - všechny binární:

- **dangerRight, dangerLeft, dangerAhead** - rozhodne, zda při daném směřování had při dalším tahu narazí do překážky (sebe či zdi)
- **foodAhead** - rozhodne, zda je přímo před hadem při jeho aktuálním směřování jídlo
- **foodUp, foodLeft** - rozhodne, zda se další kus jídla nachází nad nebo nalevo od aktuální pozice hada
- **movingLeft, movingRight, movingUp, movingDown** - rozhodne, kterým směrem se had aktuálně pohybuje

Terminály stromu:

- **forward, turnLeft, turnRight** - symbolizují finální rozhodnutí hada, zda zabočit nebo pokračovat rovně

Inicializace šlechtění: Před započetím samotného algoritmu šlechtění je třeba vytvořit počáteční populaci. Z mnoha zvolených metod jsem zvolil **metodu FULL**, která vytvoří populaci stromů s pevně danou hloubkou.

Algoritmus šlechtění: Pro samotné šlechtění je klíčová iterativně (provádí přechod mezi jednotlivými generacemi populace) volaná funkce **evolve()**, která má následující kroky:

1. Z aktuální populace (vždy řazené podle fitness jednotlivých jedinců) vyber nejlepších X procent (parametr šlechtění eliteness) a označ je jako elitní jedince.
2. Přidávej k těmto elitním jedincům nové jedince, tak dlouho, dokud nedosáhneš původní velikosti populace. Tyto jedince získej takto:
    a. S určitou pravděpodobností (parametr šlechtění mutation) aplikuj na náhodného člena neelitní populace jeden z **operátorů mutace**
    b. S určitou pravděpodobností (parametr šlechtění crossover) vyber dva jedince z elitní populace a zkřiž je pomocí jednoho z **operátorů křízení**
    c. Ve zbytku případů vyber náhodného člena neelitní populace a označ ho jako elitního.
3. Proveď pro každého člena nové populace výpočet jejich fitness a následně je podle ní seřaď.
4. Celý postup opakuj, dokud nejsi na konci generační řady.

Na konci generační řady je pak provedeno finální ohodnocení nejlepších 5% jedinců, přičemž každý je spuštěn ve 100 instancích hry, z nichž je spočítána průměrná fitness a finální funkcí je zvolena ta s nejvyšším průměrem.

Operátory mutace:

1. **Subtree mutation**: nahrazení náhodného podstromu nově vygenerovaných stromem
2. **Point mutation:** nahrazení libovolného uzlu jiným, náhodným uzlem stejné arity
3. **Shrink mutation**: “ustřižení” podstromu a jeho nahrazzení terminálem
4. **Permutation mutation:** prohození potomků náhodně vybraného uzlu

Operátor křížení: Mezi dvěma vybranými předky je provedena výměna dvou náhodně zvolených podstromů, čímž vzniknou dva noví potomci.

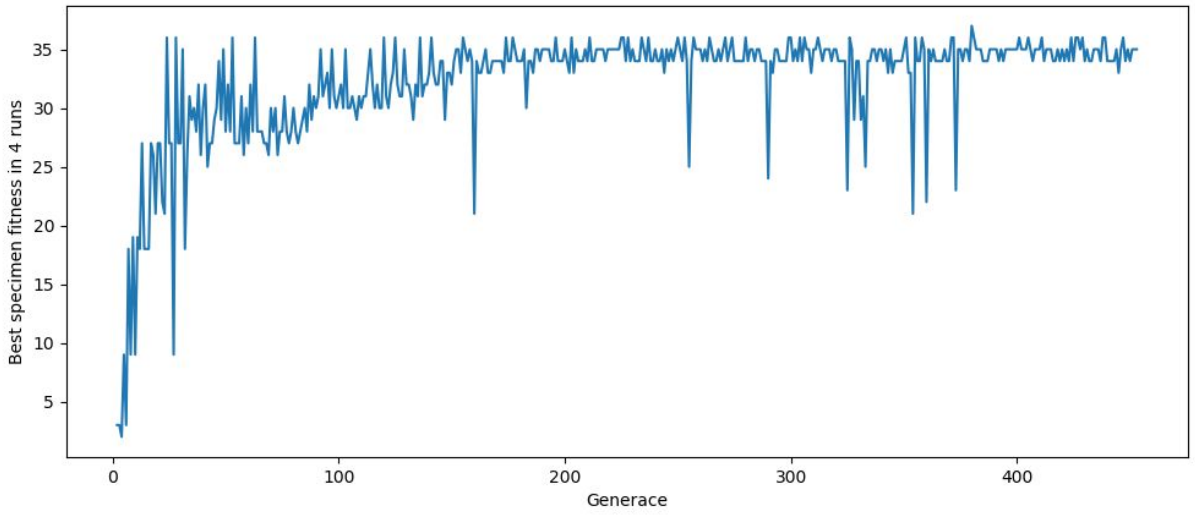
Implementace: Implementace šlechtění i samotné hry had proběhla v jazyce JavaScript z důvodu jednoduché grafického provedení pomocí canvas v HTML a snadné práce s funkcemi vyššího řádu. Zpětně ale považuji výběr netypovaného dynamického jazyka za chybu, neboť kontrola validity při kompilaci by značně urychlyla hledání malých, ale nepříjemných chyb v implementaci.

Výsledná funkce je na konci šlechtění serializována ve formátu JSON, touto formou tedy dochází k převodu funkce na “lidsky čitelný” fenotyp a zároveň je pak takovou funkcí možnou opět použití pro řízení hada.

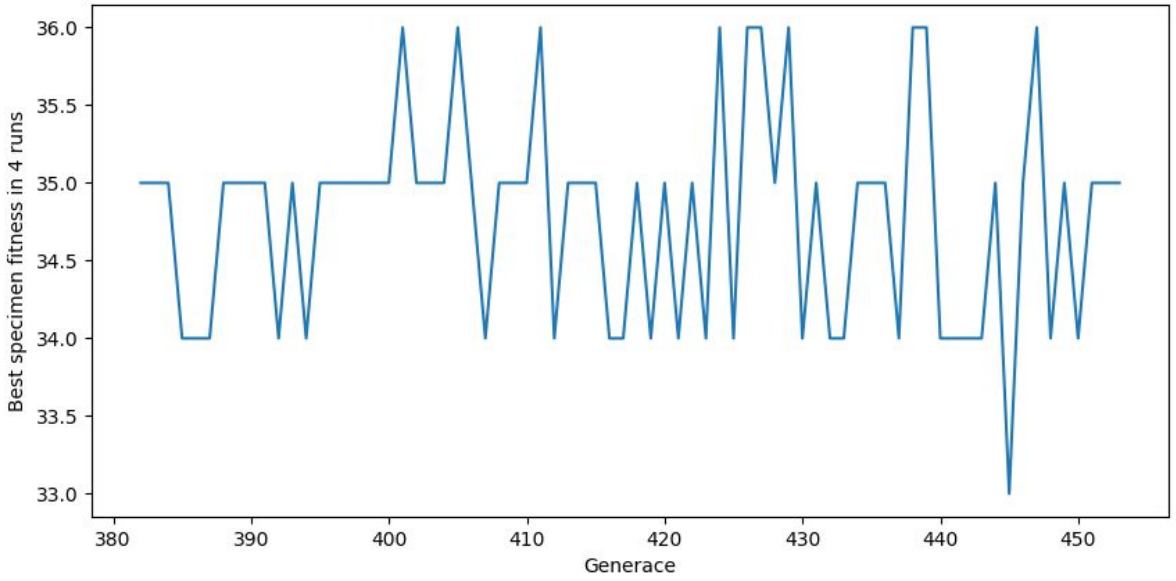
Výsledky experimentu

Ve své finální podobě vyšlechtěná řídící funkce v průměru 100 běhů **dokázala nasbírat 32 kusů jídla**. **Hloubka jejího stromu byla 14**.

Následuje graf vývoje fitness nejlepšího jedince populace v závislosti na generaci.



Jak je z grafu vidět, posledních circa 70 generací nepřineslo řešení žádných extrémní přínos.



Je ale třeba připomenout, že tyto průběžná měření jsou prováděna pouze na 4 běžích hry, přičemž výběr finálního jedince je prováděn pomocí celkem 100 běhů.

Pokud ale vybereme několik generací v rozmezí 380 až 440 a z nich provedeme výběr obdobný tomu finálnímu (výběr 5% vrcholných jedinců a jejich aplikaci na 100 instancí, následné zprůměrování a výběr nejlepšího), můžeme se na základě grafu vyřčít domněnku, že ačkoliv posledních několik generací nezlepšilo maximum generované funkce pro optimální jedince v optimální situaci, **další iterace zvýšila konzistentost generované funkce pro více situací**.