Hra v pyGletu - Had

František Vančát
ČVUT–FIT
vancafra@fit.cvut.cz

3. ledna 2021

1 Úvod

Hru had zná velké množství lidí. Nejpopulárnější je pravděpodobně verze hry na černobílé telefony Nokia. I přes to, že je tato hra přes dvacet let stará, má hra skvělý potenciál v mnoha ohledech. Jedním z nich je nechat ovládání hada na algoritmu. A přesně o toto jsem se pokusil.

V moji verzi hry se držím základních pravidel. Okraje mapy a tělo hada se počítají jako překážky. Jablko se vždy objeví v prázdném políčku a po jeho sebrání se had prodlouží o právě jednu kostičku a skóre se zvedne o jedna. Had začíná o délce jedna (tedy jen hlava) a může dosáhnout až délky, která se rovná počtu políček. Skóre je v intervalu od nuly (had bez těla narazí do stěny) až po počet hracích políček bez jedné (Had dosáhne maximální délky a kousne se do ocasu).

Jako řídící algoritmus jsem zvolil A*, který hledá nejkratší cestu k cíli. Podrobněji v sekci 4.

2 Varianty hry

Hra se dá změnit mnoha způsoby. Od jiných barev, přes hrací plochu až po jiná pravidla a ovládání. Ovšem co se týče základních parametrů, které ovlivňují chování řídícího algoritmu, nejvýznamnějším je rozměr hrací plochy.

Rozměr hrací plochy se u různých algoritmů projevuje jinak.

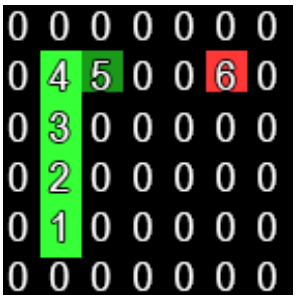
Převážně jde o velikost hrací plochy. Její zvětšení výrazně zpomaluje algoritmy z dvou hlavních důvodů. Prvním je rychlost a počet průchodů. Nějaké algoritmy nevolí nejrychlejší trasu a někdy neberou v potaz ani lokaci jablka, proto zvětšení hrací plochy může výrazně prodloužit dobu hraní. Druhým důvodem je rychlost výpočtu trasy. Některé algoritmy musí zvážit mnoho různých cest, než naleznou tu optimální a tyto výpočty musí dělat velmi často (minimálně jednou za každé snědené jablko).

Další významná součást rozměrů hrací plochy je lichý a sudý počet řádků, popřípadě sloupců. V některých algoritmech to může ovlivnit počítání trasy na okrajích hrací plochy. Pokud je průchod vytvořený hadem a okrajem mapy příliš malý, může se stát, že had zajede dovnitř a nebude se mít kde otočit a vyjet. Tudíž při špatném postupu může zajet do takzvané slepé uličky.

3 Reprezentace dat

V mojí hře za zmínku stojí hlavně tyto tři typy reprezentace dat.

Prvním je hrací plocha. Ta je reprezentována pomocí dvou-rozměrného NumPy pole. Jednotlivé prvky jsou celá nezáporná čísla. Číslo nula znázorňuje prázdné políčko. Největší číslo v poli znázorňuje jablko (vždy je o jedna větší, než délka hada). Druhé největší číslo znázorňuje hlavu hada a každé další číslo směrem k nule (bez nuly) je vzdalující tělo hada.



Obrázek 1: reprezentace dat v NumPy poli

Druhým je směr, nebo lépe, list směrů. Had se vždy pohne o jedno políčko. A může se pohnout jen 3 směry (s vyjímkou prvního kroku, kdy může všemi čtyřmi). Nikdy nelze změnit směr tak, aby had jel na druhou stranu. To by vedlo k okamžité smrti. Směr je reprezentován tuplem. První prvek je souřadnice v ose X a druhý v ose Y. Hodnota prvku je buď -1, 0, a nebo 1. Například pohyb vpravo je uložen jako (1, 0). Díky této reprezentaci je snadné udržet sekvenci směrů a zároveň udržet přehlednost. Řídícímu algoritmu tudíž stačí poslat hrací plochu a ten nám vráti seznam směrů, kterými se má had vydat, aby snědl jablko.

Poslední je pevná souřadnice. Opět je uložena v tuplu, ale tentokrát je v ní uložení přesná pozice v poli. Tudíž například horní políčko vpravo na ploše o rozměrech 15x15 je (14, 14). To je užitečné pro

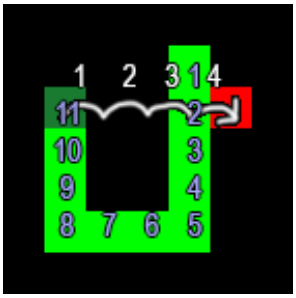
vykreslování do hrací plochy. Ze směrů na pevnou souřadnici a obráceně lze snadno a rychle převádět.

4 Metody/postupy/algoritmy

Má hra využívá algoritmus A*, neboli A-star. Jedná se o algoritmus, který nalezne nejkratší cestu k cíli a bere v potaz překážky, náročnost pohybu a přiblížení k cíli. Pohybová plocha je reprezentována mřížkou, což je pro hru had ideální, jelikož hrací plocha je obdélníková a had se hýbe po políčkách.

Hledání začíná u hlavy hada a rozšiřuje se dál. Postupně kontroluje políčko po políčku a zkontrolované políčka ukládá stranou. Dokud má co kontrolovat, tak hledá dál. Pro každé políčko kontroluje, zda je překážka. Pokud ano, tak ho přeskočí. Pokud není, tak zkontroluje délku trasy od hlavy (g) až na toto políčko, délku trasy k cíli (h) a jejich součet. Pokud se tato políčka již nevyskytují v seznamu zkontrolovaných a dokontrolovaných políček s nižším součtem tras (f), tak tato políčka přidá do seznamu ke kontrole a nastaví jim rodiče (políčko, ze kterého na toto nové přešlo). Takto pokračuje dál, dokud nenarazí na cílové políčko (jablko). Poté se přes uložené rodiče vrátí až k hlavě hada a uloží si seznam směrů, které had musí zvolit, aby se k jablku dostal.

Moje varianta pro hru Had má pár vlastností a modifikací. Výpočet vzdálenosti od hlavy nebo k cíli počítá jako součet vzdáleností po ose X a po ose Y. Pokud algoritmus při hledání narazí na tělo hada, zkontroluje, zda tam tělo pořád bude, až se had přiblíží. Díky reprezentaci dat stačí vzít příslušné číslo (specifikované souřadnicemi) z pole reprezentující herní situaci. Pokud je menší, než vzdálenost od hlavy, je toto políčko bezpečné a had do sebe při průchodu tímto políčkem nenarazí.



Obrázek 2: Předposlední část hada bere algoritmus jako volné

Na obrázku se hlava hada dostane na políčko, kde je nyní předposlední část hada až po třetím kroku, zatímco předposlední část hada zmizí po dvou pohybech hada. Tudíž algoritmu toto políčko považuje za volné.

Samotný A* algoritmus nedosahuje tak dobrých výsledků (viz. sekce 5), ale je skvělým základem pro další algoritmy. V celku bych rozdělil algoritmy na tři různé druhy. Algoritmy hledající efektivní cestu (nemusí to být vždy ta nejkratší), ty co využívající hamiltonovu smyčku (algoritmus, který projde každým políčkem právě jednou a skončí tam, kde začal) a jejich kombinace. Kombinace těchto dvou algoritmu se zdá být nejefektivnější, ale je potřeba ošetřit nějaké mezní případy a rozhodnout, kdy přesně se mají algoritmy přepínat.

Hamiltonova smyčka zaručí sto-procentní dokončení hry při každém průběhu, je ovšem ale velmi pomalý a pro netriviálně veliké hrací plochy sám o sobě nemůže efektivně fungovat. Algoritmy hledající nejefektivnější cesty jsou sice přímé, ale jsou složitější na výpočty a had může narazit ještě před dokončením hry.

Dalším problémem je předpovídání trasy k následujícímu jablku. Je totiž možné, že i když had najde cestu k jablku, odřízne si tím cestu k dalšímu jablku a narazí. Proto není vždy nejlepší volba hledání nejrychlejší trasy. Napomoct při řešení může několik principů. Jedním je snažit se kopírovat tvar hada a jet po okrajích. To nerozdělí volnou plochu na více dílů. Dále je třeba možné v určitých chvílích hledat naopak nejdelší trasu. To aby obklíčený had získal čas a vymotal se z uzavřeného prostoru. Pak ještě had může vybírat trasu tak, aby v nutných případech rozdělil trasu tak, aby se nacházel v segmentu, kde je více volných polí, než v segmentu druhém.

5 Výsledky

Obyčejný A* algoritmus nedosahuje příliš dobrých výsledků, ale slouží jako skvělý základ pro pokročilejší algoritmy. Abych získal potřebná data, nechal jsem počítač proběhnout osm různě velikých hracích ploch po sto pokusech. Tudíž v celku osm-set průběhů.

Při opravdu velké hrací ploše se občas stane, že se had na chvíli zasekne, protože A* má náročné výpočty a pro velké vzdálenosti nějakou dobu trvá, než nalezne trasu. Tento algoritmus by šel do budoucna optimalizovat ošetřováním krajních případů a lepšího výběrů preferovaných políček. Za nějakých předpokladů by nějaké výpočty mohli běžet méněkrát, ale to by vyžadovalo hlubší průzkum chování a zkoumání preferovaných výběrů.

hrací plocha	5x5	10x10	15x15	20x20
maximum	24	99	224	399
nejhorší	4	7	11	19
nejhorší %	16.67%	7.07%	4.91%	4.76%
nejlepší	18	40	65	93
nejlepší %	75.00%	40.40%	29.02%	23.31%
průměr	10.27	23.87	35.95	47.71
průměr %	42.79%	24.11%	16.05%	11.96%

hrací plocha	25x25	30x30	35x35	40x40
maximum	624	899	1224	1599
nejhorší	25	21	38	33
nejhorší %	4.01%	2.34%	3.10%	2.06%
nejlepší	101	119	152	151
nejlepší %	16.19%	13.24%	12.42%	9.44%
průměr	61.51	74.83	84.91	99.59
průměr %	9.86%	8.32%	6.94%	6.23%

6 Závěr

Ukázalo se, že samotný A* nedosahuje tak dobrých výsledků. Ovšem pro další algoritmy, které jsou jak rychlé, tak spolehlivé, je tento algoritmus, jakožto základ, téměř nutný. Jde optimalizovat ošetřením nějakých krajních případů, které vyplývají z principu fungování hada a některých výjimečných situací, které mohou nastat. Ale většina problému současného algoritmu může být vylepšena implementací dalšího algoritmu a jejich vhodným přepínáním dosáhnout mnohem lepších výsledků.

Tento algoritmus je velmi efektivní na začátku hry, kdy se k jablku blíží velmi přímo, zatímco Hamiltonovy algoritmy pozici jablka někdy až ignorují.

Tato aplikace by šla snadno využít jako základ pro nějaké další algoritmy, kterými bych se v budoucnu mohl zabývat.

I přes jednoduchý princip hry je problematika hledání optimální cesty velmi zajímavá a mnohdy i velmi složitá. Dají se do ní implementovat různě náročné algoritmy, které se od sebe velmi liší, tudíž skvěle slouží pro lidi, kteří se o podobné problematice chtějí naučit více.

V budoucnu se k této problematice určitě vrátím, jelikož mám spoustu nápadů, které bych chtěl do hry implementovat.

7 Zdroje

CODECOMPILE. A* Pathfinding Tutorial. YouTube [online]. 2012, 28. 7. [cit. 2021-01-03]. Dostupné z: https://www.youtube.com/watch?v=KNXfSOx4eEE

ALPHAPHOENIX. How to Win Snake: The UNKILLABLE Snake AI. YouTube [online]. 2020, 16. 2. 2020 [cit. 2021-01-03]. Dostupné z: https://www.youtube.com/watch?v=TOpBcfbAgPg

CODE BULLET. I Created a PERFECT SNAKE A.I. YouTube [online]. 2019, 28. 11. [cit. 2021-01-03]. Dostupné z: https://www.youtube.com/watch?v=tjQIO1rqTBE

BELWARIAR, Rachit. A* Search Algorithm. GeeksForGeeks [online]. 2018, 7. 9. [cit. 2021-01-03]. Dostupné z: https://www.geeksforgeeks.org/a-search-a

GLEN, Stephanie. Hamiltonian Cycle: Simple Definition and Example. StatisticsHowTo [online]. 2017, 12. 11. [cit. 2021-01-03]. Dostupné z: https://www.statisticshowto.com/hamiltonian-cycle/

KONG, Shu a Joan Aguilar MAYANS. Automated Snake Game Solvers via AI Search Algorithms [online]. 2016 [cit. 2021-01-03]. Dostupné z: https://cpb-us-e2.wpmucdn.com/sites.uci.edu/dist/5/1894/files/2016/12/AutomatedSnakeGameSolvers.pdf. University of California, Irvine.