Snake game

Marek Firla
200798
28.5.2022

Úvod

Had je druh počítačové arkádové hry, ve které hráč ovládá pohyby hada na obdélníkové hrací ploše. Hrací plocha je buď ohraničená, nebo se jedná o nekonečnou plochu, kdy se had zanořením na jednom okraji objeví na protějším. Hráč nejdříve ovládá tečku, která se pohybuje vpřed. Cílem hry je dosáhnout co nejvyššího skóre, které se zvyšuje postupným sbíráním „jídla“, předmětů náhodně se generujících na herní ploše. Po každém úspěšně sesbíraným „jídla“ se přičte skóre a prodlouží se délka hada o jeden dílek. Hra končí, pokud had narazí do sebe sama.

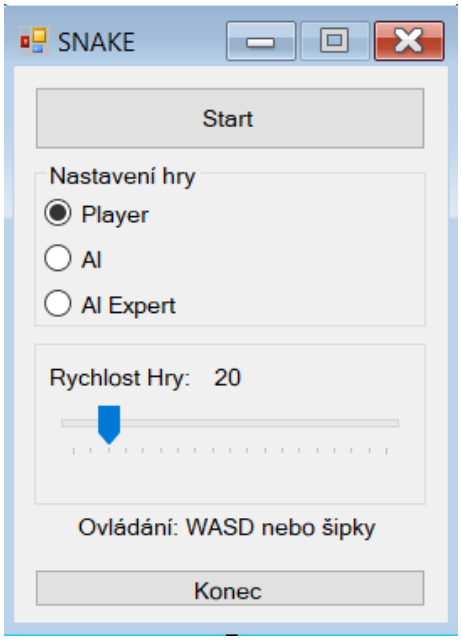
Historie

Historie herního konceptu sahá do 70. let, přesněji do roku 1976, kdy firma Gremlin Industries představila arkádovou hru Blockade pro dva hráče. Každý hráč ovládal svého hada pevné délky a jeden had chtěl svým tělem zablokovat pohyb toho druhého.

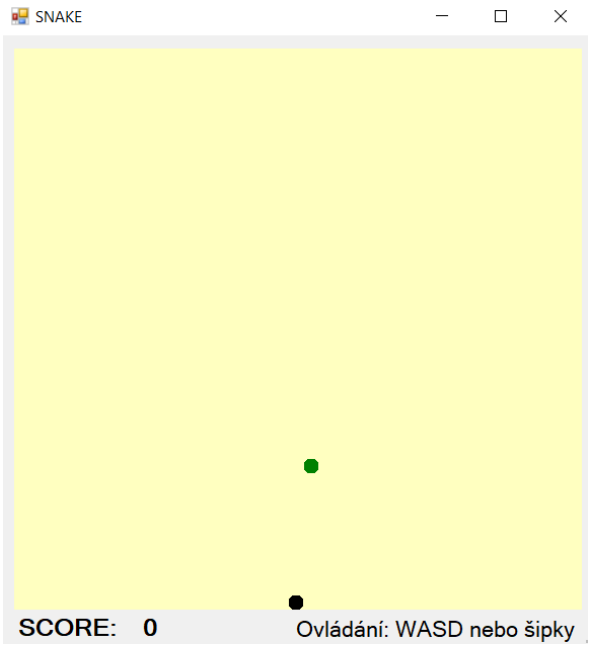
První had pro jednoho hráče „Snake Byte“, byl vytvořen v roce 1982 pro 8bitové počítače Atari, Apple II a VIC-20. Had v ní jedl jablka, aby dokončil úroveň, přičemž rostl na délce. V roce 1982 pak ve hře „Nibbler“ jsou do hracího pole přidány další překážky. Hru následně zpopularizovala v 90. letech firma Nokia, která nabízela vlastní verzi hry ve svých mobilních telefonech.

Popis a ovládání programu

Menu se skládá z tlačítka pro start hry, nastavení typu hráče, nastavení rychlosti hry a tlačítka pro ukončení programu. Tlačítko start spustí samostatné okno se hrou. Nastavení typu hráče určí, kdo bude samotného hada ovládat, a to buď to hráč nebo algoritmus A*. Posuvník s nastavením rychlosti hry určuje, jak rychle se had skrze hrací pole pohybuje. Tlačítko konec celou aplikaci ukončí.



Samotné okno se hrou obsahuje herní pole a počítadlo score. Pro hru reálného hráče so poté had ovládá pomocí WASD nebo šipek. Pokud se v menu v položce nastavení typu hráče zvolí místo položky hráč položka AI nebo AI expert, had se bude pohybovat zcela sám a sbírat po hrací ploše „jídlo“. Po skončení hry lze poté hru restartovat stisknutím klávesy enter nebo hru ukončit a vrátit se do menu stiskem klávesy escape.



Algoritmus A*

Je počítačový algoritmus k hledání nejkratší cesty v kladně ohodnoceném grafu. Vychází s Dijkstrova algoritmu a přidává k němu heuristiku.

Historie

S první verzí A* algoritmu s názvem A1 přišel v roce 1964 Nils Nilsson a byla to úprava Dijkstrova algoritmu o přidání heuristiky. Další vylepšení přišlo roku 1967 od E. Harta, tento algoritmus nazval A2. Poslední obecné vylepšení poté vytvořil Bertram Raphael v roce 1968. Dokázal, že jeho verze algoritmu A2 je nejlepší algoritmus pro dané podmínky a přejmenoval jej tedy na A*, kde * znamená unixovou shellovou syntaxi, tj. algoritmus A a všechny další verze.

Popis algoritmu

Algoritmus vychází s Dijstrova algoritmu je v principu shodný s prohledáváním do šířky. Rozdílem je, že používá frontu prioritní, ve které jsou cesty seřazeny podle hodnoty funkce f. Tato funkce je definována pro každou cestu p a je součtem tzv. heuristické funkce (h) posledního uzlu cesty p a její zbývající délky (g). Čím je hodnota funkce f(p) nižší, tím vyšší má daná cesta p prioritu.

Pořadí cest ve frontě je určeno následující funkcí:

$f(x) = h(x) + g(x)$

$f(x) - p\check{r}edpokladan\acute{a}\ d\acute{e}lka\ cesty$

$h(x) - heuristick\acute{a}\ funkce$

$g(x) - d\acute{e}lka\ cesty\ do\ uzlu$

Heuristická funkce musí být přípustná. To znamená, že její hodnota pro libovolný uzel musí být větší než nula, protože její hodnota nikdy nemůže být větší, než je skutečná vzdálenost z daného uzlu do cíle. Heuristická funkce se vybírá na základě znalosti struktury problému například euklidovská metrika.

Kroky algoritmu

1. Vytvoř prázdnou množinu cest F.
2. Do množiny F vlož cestu nulové délky obsahující počáteční uzel s.
3. Dokud není množina F prázdná, opakuj:
    a. Z množiny F vyber nejkratší cestu p (s nejnižší hodnotou f(p)) a odeber ji.
    b. Končí-li cesta v cílovém uzlu, vrať ji a ukonči výpočet.
    c. Vytvoř nové cesty použitím všech možných operátorů na koncový uzel cesty p, které neobsahují smyčky.
    d. Jestliže dvě cesty končí ve stejném uzlu, odstraň všechny kromě té nejkratší (s nejnižší hodnotou f(x)).
    e. Přidej cestu p do množiny F.
    f. Je-li množina F prázdná, oznam, že žádná cesta z počátečního do cílového uzlu neexistuje.

Pseudokód

```
function A*(start, cíl)
    closedset := prázdná množina                    // Množina již uzavřených uzlů.
    openset := množina obsahující pouze počáteční uzel // Množina otevřených uzlů.
    g_skore[start] := 0                             // Délka aktuální optimální cesty.
    h_skore[start] := heuristický_odhad_vzdálenosti(start, cíl)
    f_skore[start] := h_skore[start]                // Předpokládaná délka cesty mezi startem a cílem jdoucí přes y.
    while openset is not empty
        x := otevřený uzel s nejmenší hodnotou f_skore[]
        if x = cíl
            return rekonstruuj_cestu(přišel_z[cíl])
        vyjmi x z openset
        přidej x do closedset
        for each y in sousední_uzly(x)
            if y in closedset
                continue
            stávající_g_skore := g_skore[x] + d(x, y)

            if y not in openset
                add y to openset
                stávající_je_lepší := true
            elseif stávající_g_skore < g_skore[y]
                stávající_je_lepší := true
            else
                stávající_je_lepší := false
            if stávající_je_lepší = true
                přišel_z[y] := x
                g_skore[y] := stávající_g_skore
                h_skore[y] := heuristický_odhad_vzdálenosti(y, cíl)
                f_skore[y] := g_skore[y] + h_skore[y]
    return failure

function rekonstruuj_cestu(aktuální_uzel)
    if přišel_z[aktuální_uzel] is set
        p = rekonstruuj_cestu(přišel_z[aktuální_uzel])
        return (p + aktuální_uzel)
    else
        return aktuální_uzel
```

Popis zdrojového kódu

Celý program je psaný v jazyce C# s použitím WindowsForms pro tvorbu uživatelského rozhraní.

Vstupem do programu je třída s názvem Program zde se jen povolí vytvořit vizuální prvky pro GUI a spustí se základní menu.

V základním menu se vytvoří prvky pro nastavení rychlosti a typu hry a dále pak dvojice tlačítek pro spuštění hry a pro ukončení aplikace (Start, Konec). Tlačítko start nejprve uloží zvolené nastavení a poté spustí samotnou hru. Tlačítko Konec aplikaci ukončí.

Pro zápis a čtení hodnot pro hru se stará třída Nastavení. Proměnné jsou: SirkaPolicka, DelkaPolicka, definuje velkost mřížky, na které se bude had pohybovat. RychlostHry určuje rychlost pohybu hada. Score je počet bodů cekem. Body inkrement Score za „Jídlo“. HrajeAI, HrajeAIExpert udává typ hráče AI (AI najde cestu pouze při vygenerování jídla, ExpertAI hledá cestu při každém pohybu), GameOver udává, zda se nemá hra ukončit. Smery definují možné směry pohybu hada.

Třída PoziceObjektu pak obsahuje informace o X a Y pozici objektu (dílek hada nebo „jídlo“) v herním poli, Dále pak hodnoty nutné pro A* algoritmus f(x) = h(x) + g(x). A metodu pro výpočet funkce h(x).

Třída Ovládání pak slouží k čtení výstupu z klávesnice.

Třída Game poté stará o chod samotné hry.

- Při spuštění se inicializují komponenty, a to herní plochu (GameBox) a počitadlo skoré (Score), tabulku která po ukončení hry vypíše maximální scóre (EndScreen) a Timer (GameTimer). Ten volá metodu Aktualizace v pravidelných intervalech nastavených podle proměnné RychlostHry. Poté se zavolá Start čímž se zahájí hra.
  Vytvoří se list Had, kde se ukládají jednotlivé články hada (Had), list pohybu, který kde se ukládá výstup s A*(pohyby) a Jídlo typu PoziceObjektu.
- **Aktualizace** - vyhodnocuje, zda se hra neukončila, sleduje vstupy, zajištuje pohyb hada voláním metody Pohyb.
- **Start** - nastaví výchozí hodnoty a zavolá metodu GenerujJídlo.
- **GenerujJídlo** - vygeneruje náhodnou pozici a uloží ji do proměnné jídlo, pokud je HrajeAI pravdivé tak se spustí Astar a najde nejkratší cestu k němu od hlavy hada a uloží ji do listu pohyby.
- **Jez** - vytvoří nový článek těla hada a přidá ho na konec listu a přičte se skóre a zavolá metodu GenerujJídlo.
- **Pohyb** - zajišťuje pohyb hada změnou pozic článků v listu Had, metoda taktéž kontroluje kolize hlavy hada s jídlem (pokud nastane zavolá metodu Jez) a s ostatními články hada (zvolá metodu Konec).
- **Konec** - změní jen proměnou GameOver na true.
- **GenerujHerniPole** - zjistí aktuální pozice článků hada a uloží reprezentaci do listu, tento list je vstup pro Astar.
- **Astar** – A* algoritmus, který najde nejkratší cestu od startu (první článek hada, první pozice v listu Had) po cíl (Jídlo) s překážkami danými mapou (vstup funkce), pokud nejkratší cestu najde, převede cestu na směry kudy se musíme pohybovat a ten vrátí jako list pohyby.
- **GrafikaHada** – Pomocí třídy Graphic (po tvorbu jednoduché grafiky vykreslíme na herním plánu (GameBox) jednoduchou reprezentaci hada podle pozic uložených v listu Had.

Závěr

A*, který ovládá hada dosahuje poměrně vysokého skoré, ovšem není úplně ideální, jelikož nejkratší cesta není vždy nejlepší z hlediska strategie a často se stává že had sám sebe uzavře, popřípadě k jídlu neexistuje cesta. Algoritmus by se dal vylepšit o to část programu, která bude řídit hada, pokud nebude cesta k jídlu nalezena.

Zdroje

https://cs.wikipedia.org/wiki/A*¨

https://www.geeksforgeeks.org/a-search-algorithm/

https://dotnetcoretutorials.com/2020/07/25/a-search-pathfinding-algorithm-in-c/

https://www.mooict.com/c-tutorial-create-a-classic-snake-game-in-visual-studio/