PATHFINDING A GENEROVÁNÍ MAPY

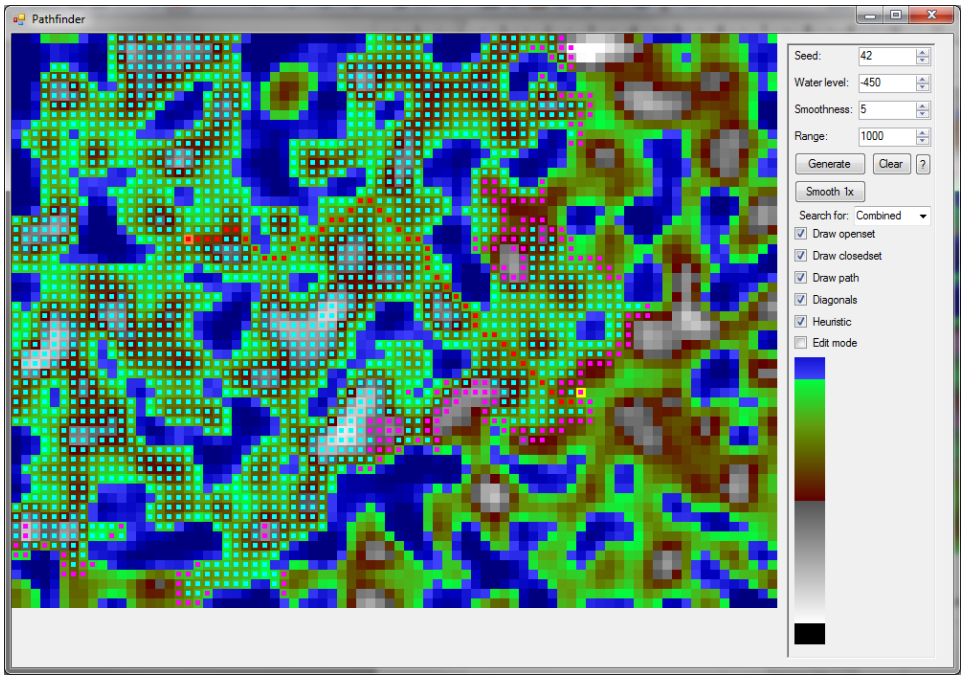
Luboš Běhounek
2. ročník
AI3
lubos.behounek@uhk.cz

Abstrakt
Tato práce se zabývá jednoduchým generováním výškové mapy pomocí generování náhodných hodnot a jejich rozostřením a následným grafickým zobrazením v ukázkovém programu, dále se zabývá hledáním nejvýhodnější cesty v této vygenerované výškové mapě za využití algoritmu A* s heuristikou i bez s možností nastavení různých parametrů – například jestli se lze pohybovat diagonálně, podle jaké funkce se má cesta hledat apod. To je vše ukázáno v ukázkovém programu, který navíc nabízí i možnost editace vlastní mapy.

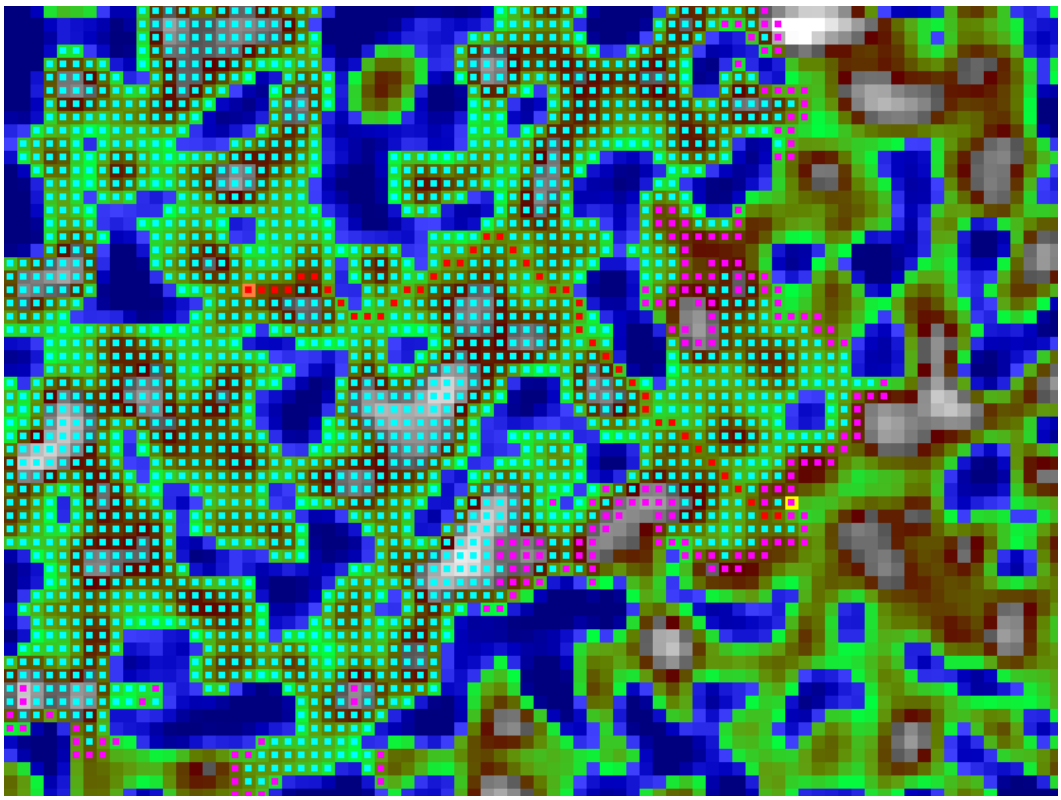
Úvod
Generování map se používá obvykle pro různé zobrazení terénu – například různé umělé scény, pro počítačové triky ve filmech a v počítačových hrách (kde není použito modelování ruční).

K tomuto účelu jsem v jazyce C# vytvořil jednoduchý ukázkový program, který dokáže generovat pseudonáhodné mapy a nechá uživatele je upravovat nebo si uživatel může vytvořit mapu zcela vlastní. Dále může uživatel nastavovat základní parametry zobrazování výsledků a měnit základní nastavení prohledávacího algoritmu. Samotné algoritmy jsou zde pak i popsány.

Popis ukázkového programu
Ukázkový program se skládá z hlavního okna (obr. 1), na kterém se nachází grafické 2D zobrazení terénu v pohledu zhora (vlevo, obr. 2a) a uživatelské rozhraní sloužící k nastavení různých hodnot, generování nové mapy a nastavení hledacího algoritmu (napravo, obr. 2b). Program se ovládá klasicky myší a klávesnicí, tlačítky myši uživatel zadá start a cíl a může měnit různá nastavení programu, pokud se přepne do „Edit módu“, tak může kliknutím nebo tažením myši editovat mapu.



Obr. 1: Hlavní okno programu



Obr. 2a: Grafické znázornění mapy

Obr. 2b: Uživatelské rozhraní

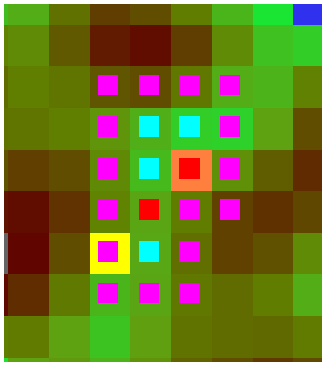
V uživatelském rozhraní (obr. 2a) se nacházejí tyto komponenty:

- nastavení semínka funkce random pro generování výšek („Seed“)
- nastavovat výšku hladiny vody („Water level“)
- nastavovat stupeň vyhlazení terénu („Smoothness“)
- nastavovat rozsah (maximální výšku) terénu („Range“)
- tlačítko pro generování nového terénu podle zadaných parametrů („Generate“)

- tlačítko pro vyčištění mapy na nejnižší hodnotu („Clear“)
- tlačítko pro zobrazení informací o programu („?“)
- tlačítko pro rychlé vyhlazení terénu o jeden stupěň („Smooth 1x“)
- rozbalovací box, kde se vybírá způsob počítání obtížnosti cesty („Search for“), v nabídce jsou možnosti „Combined“ – bere se v potaz vzdálenost i převýšení, „Elevation“ - uvažuje se pouze převýšení, „Shortest“ - hledá se pouze nejkratší cesta
- zaškrtávátko indikující, zda se mají zobrazit hraniční pole („Draw openset“)
- zaškrtávátko indikující, zda se mají zobrazit prohledané pole („Draw closedset“)
- zaškrtávátko indikující, zda se má zobrazit výsledná cesta („Draw path“)
- zaškrtávátko indikující, zda se lze pohybovat i po diagonálách („Diagonals“)
- zaškrtávátko indikující, zda má být využito přednastavené heuristiky („Heuristic“)
- zaškrtávátko indikující, zda je možné editovat terén („Edit mode“)
- grafický prvek umožňující výběr výšky štětce pro editaci terénu, zároveň slouží jako legenda (modrá barva = voda, výška <=0, zelená-hnědá barva = tráva/hlína, výška 1-127, šedá – bílá barva = kamení/sníh, výška 128-255, černá barva = neprůchozí políčko, výška > 256)

V grafickém zobrazení terénu se kromě polí terénu mohou vyskytovat následující grafické prvky (Obr. 3) :

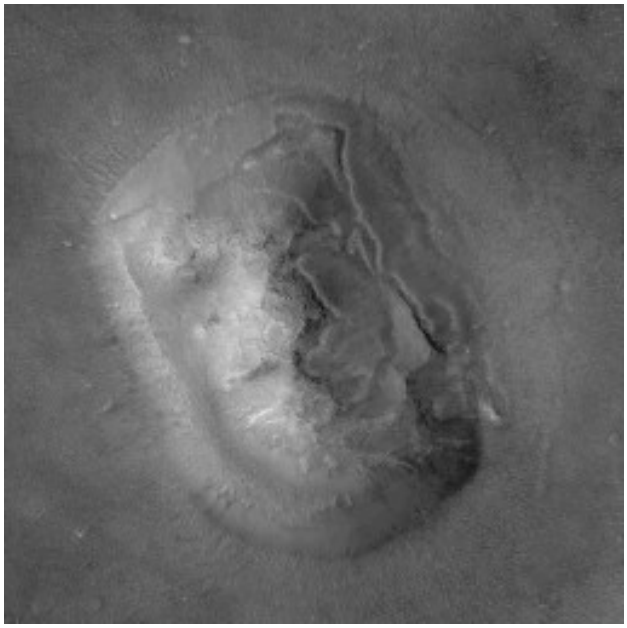
- oranžové políčko označující startovní pozici
- žluté políčko označující cílovou pozici
- červená kostička vyznačující nalezenou cestu
- azurová kostička označující prohledaná pole (tzv. closed set)
- růžová kostička označující okrajová (předběžná) pole (tzv. open set)



Obr. 3: Další grafické prvky

Generování mapy

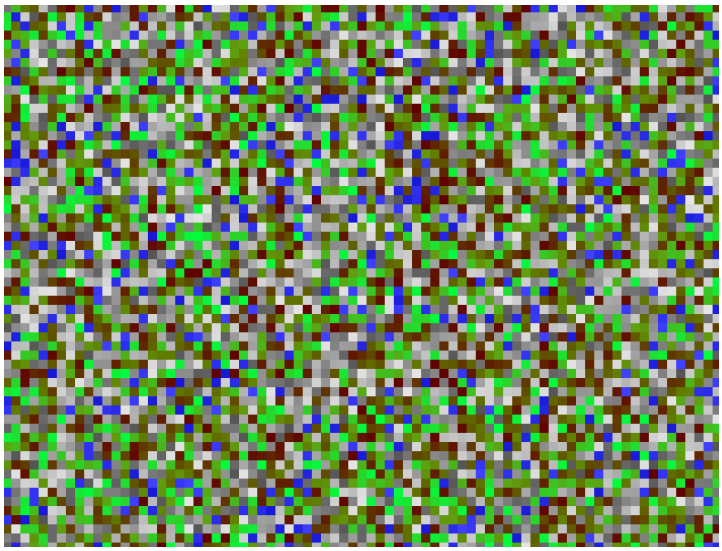
Pro uchovávání 3D map podobných terénu se dnes většinou využívá tzv. heightmapa (výšková mapa), což je dvourozměrné pole, ve kterém každý záznam určuje výšku nějakého bodu terénu, jeho pozice je určena souřadnicemi v poli a jako třetí souřadnice je použita právě hodnota v poli na této pozici, například pokud hodnota pole[4,8]=5, pak pozice bodu je (4,8,5). Pro případné dopočítání pozice bodů, které v poli nejsou, se pak používá interpolace z pozic bodů okolních. Na obr. 4 je vidět ukázková heightmapa ve stupních šedé.



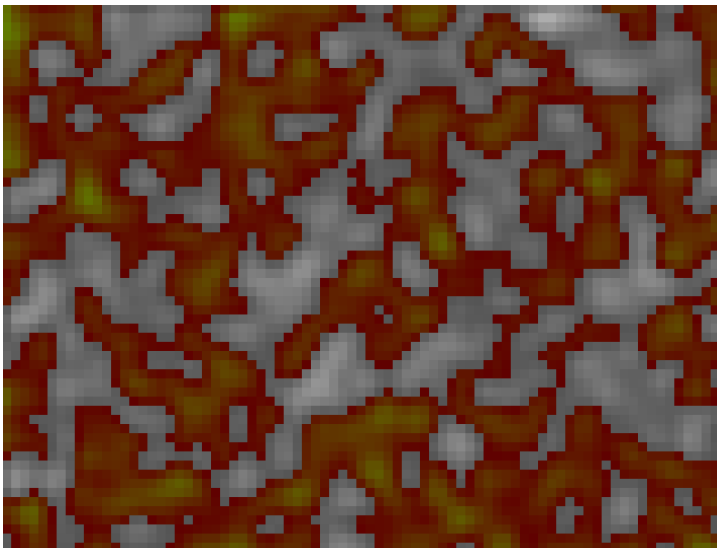
Obr. 4: Ukázka heightmapy [1]

Při generování výškové mapy lze postupovat mnoha způsoby, mezi základní patří ruční editace, která sice většinou vypadá nejlépe, ale může být pro velké mapy pracná. Proto se někdy přistupuje k procedurálnímu generování map, které v závislosti na kvalitě algoritmu dokáží vygenerovat více či méně realisticky vypadající mapu, v závislosti na kvalitě algoritmu a vhodně nastavených parametrech. V dnešní době se pro tvorbu výškových map často využívají fraktály a Perlinův šum (perlin noise[2]), já jsem si pro ukázkovou aplikaci z důvodu jednoduchosti vybral následující algoritmus, který je pro ukázkovou aplikaci plně dostačující.

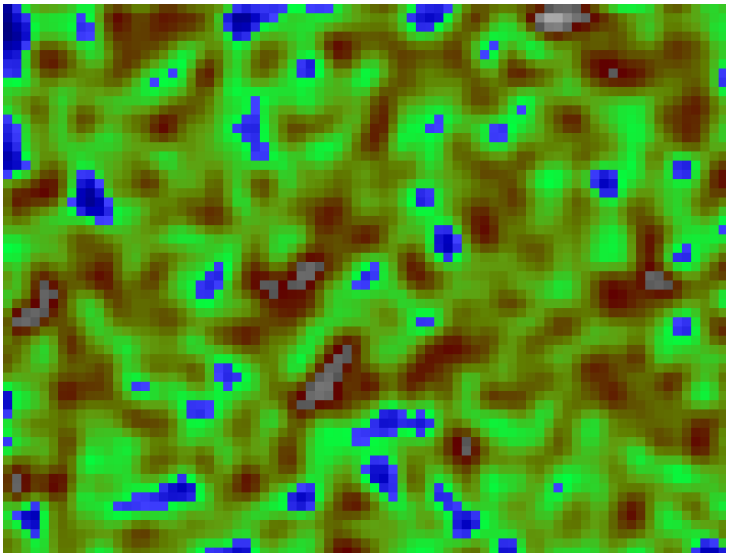
Pro každý bod v dvourozměrném poli vygeneruji náhodné číslo v rozsahu 0-V_{max}, kde V_{max} je maximální výška terénu, vznikne terén, který je ale příliš členitý (zrnitý), viz obr. 5a, proto několikrát aplikuji ještě jednoduché rozostření, které pro každý záznam spočítá průměr z jeho okolí (obr. 5b) a nakonec ještě celý terén o určitou hodnotu snížím, abych mohl určovat množství vody na mapě, viz obr. 5c (voda je všude, kde výška<0).



Obr. 5a: Terén vygenerovaný z náhodných čísel



Obr. 5b: Terén po pětinásobné aplikaci rozmazání



Obr. 5c: Terén po pětinásobném rozmazání a snížení výšky

Dále může uživatel v ukázkovém programu editovat každé pole mapy pomocí myši při zatrženém zaškrtávátku „Edit mode“ a vytvářet tak libovolnou mapu.

Hledání cesty

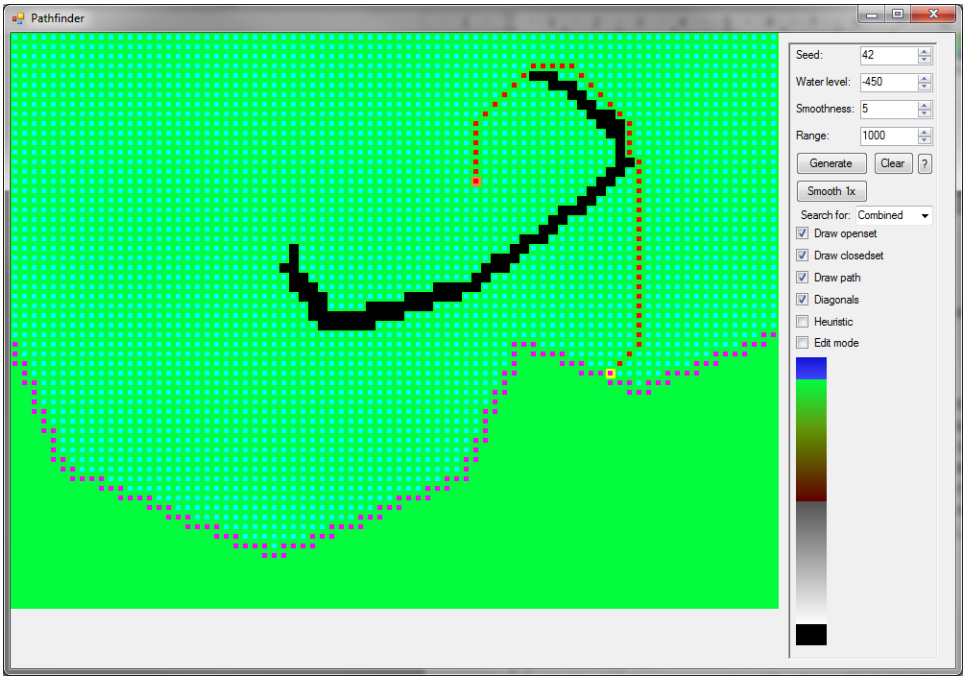
Pro hledání cesty jsem si zvolil algoritmus A*, což je v podstatě rozšíření Dijsktrova algoritmu[3] o heuristiku. Pokud je u heuristické funkce dodržena podmínka, že odhad nepřesáhne reálnou vzdálenost, tak je nalezená cesta vždy optimální, tedy že má nejmenší cenu.

Popis algoritmu: do množiny předběžných políček si uloží startovní políčko a pak začne tuto množinu cyklicky zpracovávat – vezme políčko s nejmenší odhadovanou celkovou cenou cesty, odebere ho z předběžných políček, přidá ho do již vyhodnocených a pokud toto políčko není cíl, tak vyhodnotí všechny sousedy a ti, kteří ještě nejsou ani v předběžných ani v již zpracovaných políčkách přidá do předběžných políček a případně přenastaví předchůdce, pokud je nově nalezená cesta lepší. Délka cesty na toto políčko (y) se spočítá jako celková cena cesty na políčko předchůdce (x) + cena přechodu z políčka x na políčko y.

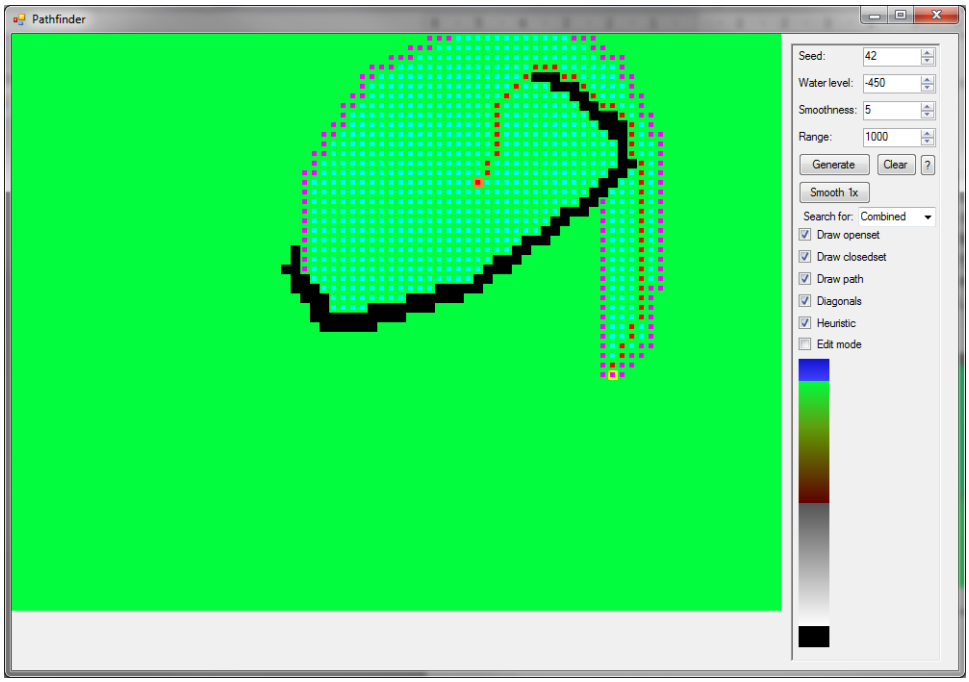
Pesudokód (základ první a druhé funkce převzat z [4]) :

```
function A*(start,cil)
     closedset := prazdna mnozina policek // jiz vyhodnocena policka
     openset := mnozina obsahujici start // mnozina predbeznych policek, která
                                          budou vyhodnocena
     priselz := prazdna mapa // mapa, kde je kazdemu policku prirazen predchudce,
                             z ktereho prisel

     gskore[start] := 0    // cena od startu podel nejlepsi zname cesty
     hskore[start] := heuristic(start, goal) // heuristicky odhad vzdalenosti
     fskore[start] := hskore[start]    // odhadovana celkova cena od startu do
                                          cile pres policko y

     while openset neni prazdny // dokud openset není prazdny
         x := policko z opensetu s nejnizsi hodnotou f_score[]
         if x = cil // cesta nalezena
             return zjisticestu(priselz, priselz[cil])

         remove x z openset // odebrani x z opensetu a prirazeni do closedsetu
         pridej x do closedset
         pro vsechny sousedy y okolo pole x // se vsemi sousedy provest
                                               nasledujici
             if y je v closedset // pokud je v closed setu (uz byl vyhodnocen),
                                    tak ho preskocit
                 continue
             predbeznegskore := gskore[x] + cena(x,y)

             if y neni v openset // pokud soused není ještě pridan, tak ho pridat
                 pridej y do openset
                 predbeznejelepsi := true
             else if predbeznegskore < gskore[y] // pokud uz pridan byl a tudyma
                                                    je to kratsi
                 predbeznejelepsi := true
             else
                 predbeznejelepsi := false

             if predbeznejelepsi = true
                 priselz[y] := x
                 gskore[y] := predbeznegskore
                 hskore[y] := heuristic(y, cil)
                 fskore[y] := gskore[y] + hskore[y]

     cesta nenalezena
     return

function zjisticestu(priselz, aktualnipolicko) // zjisti a vrati celou cestu
     if priselz obsahuje aktualnipolicko
         p = zjisticestu(priselz, priselz[aktualnipolicko])
         return (p + aktualnipolicko)
     else
         return aktualnipolicko

function heuristic(policko, cil) // heuristicka funkce
     return vzdusnavzdalenost(policko, cil)

function cena(policko1, policko2) // funkce pro zjisteni ceny cesty mezi
                                     sousednimi policky
     return rozdilvysek(policko, cil) // rozdil vysek
     return vzdalenostpolicek(policko, cil) // 1, nebo sqrt(2) na diagonale
     return rozdilvysek(policko, cil)+vzdalenostpolicek(policko, cil) // soucet
                                                                        obou
```

Pro heuristickou funkci H(n) políčka n jsem použil výpočet, který vrací vzdušnou vzdálenost vypočítávané pozice k cíli, tato funkce nikdy nevrátí hodnotu větší než je výsledná vzdálenost, takže výsledek je optimální a hledání je většinou značně urychleno, protože jsou prohledáváná pole směrem k cíli. V ukázkovém programu lze tuto funkci vypnout (H(n)=0) a porovnat tak, o kolik políček více musel algoritmus vyhodnotit, než se dostal do cíle, rozdíl je vidět na obr. 6a a obr. 6b (lze postřehnout, že každá heuristická funkce může vracet trochu jinou výslednou cestu, ale celková cena těchto cest je vždy stejná).



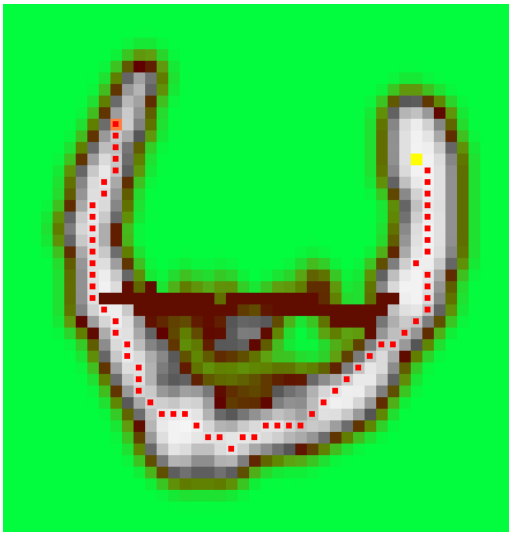
Obr. 6a: Vyhledávání s H(n) = 0



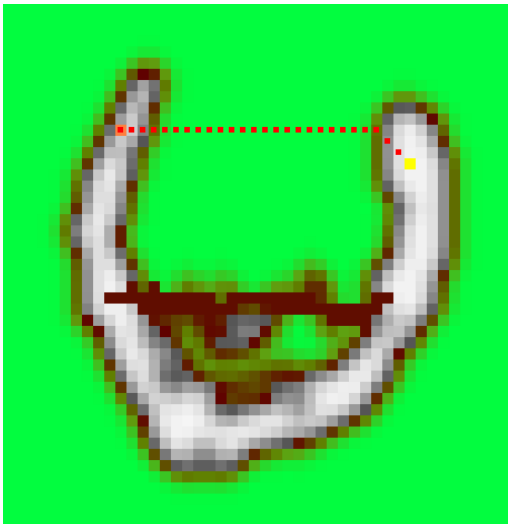
Obr. 6b: Vyhledávání s H(n) = dist(n,cil)

Cena přechodu mezi sousedními políčky se vypočítává v závislosti na nastavení rozbalovníčku „Search for“ jedním z následujících algoritmů:

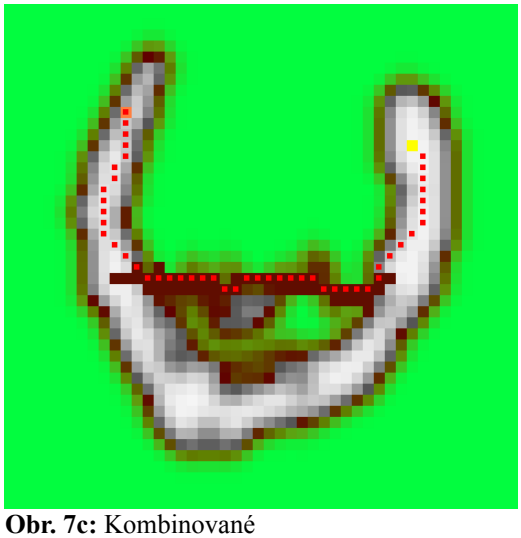
- výškový rozdíl (obr. 7a)
- vzdálenost (1, nebo $\sqrt{2}$ na diagonále) (obr. 7b)
- kombinovaná (výškový rozdíl + vzdálenost políček) (obr. 7c)



Obr. 7a: Minimální výškový rozdíl



Obr. 7b: Minimální vzdálenost



Obr. 7c: Kombinované

Informační zdroje

[1] Ski Industry Net: http://www.skindustry.net/medem/files/Temp/Redrobes/Forum/BetaDemo1/HeightMapFaceMars.png. 30.4.2011

[2] Noise Machine: http://www.noisemachine.com/talk1/index.html. 30.4.2011

[3] Rashid Bin Muhhamad's home page: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijkstraAlgor.htm. 30.4.2011

[4] Wikipedia: http://en.wikipedia.org/wiki/A*_search_algorithm. 30.4.2011