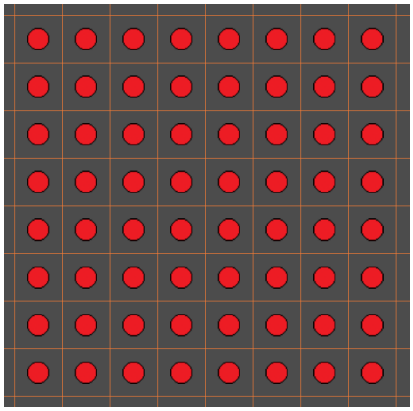
Custom A* Pathfinding v herním engine GODOT

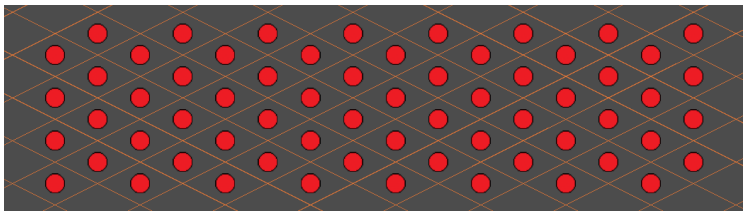
Cíl projektu

GODOT Engine má vlastní integrované varianty algoritmu A*, ty ale fungují buď na spojitém prostoru nebo pouze na čtvercové mřížce.

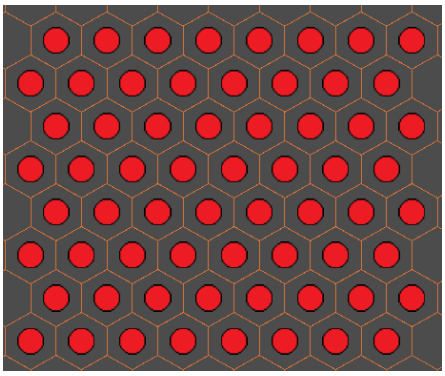
Cílem tohoto projektu je vytvořit vlastní variantu algoritmu A* která by mohla být použita i např. na izometrickou mřížku a po případné úpravě prohledávání sousedů i na prohledávání hexagonální mřížky.



Obrázek 1 Čtvercová mřížka



Obrázek 2 Izometrická mřížka



Obrázek 3 Hexagonální mřížka

GODOT Engine

Godot Engine je bezplatný a open source software vydaný pod licencí MIT.

Jako programovací jazyk využívá primárně vlastní jazyk GDScript. Ten je syntaxem velice podobný jazyku Python, na rozdíl od něj ale vyžaduje explicitní deklaraci proměnných.

Další možností je celá samostatná verze Godot, která pracuje s jazykem C#.

A* Algoritmus

- Pro prohledávání stavového prostoru se používá seznamů OPEN a CLOSED
  - Seznam OPEN je seznam uzlů k prohledávání
  - Seznam CLOSED je seznam již prohledaných uzlů
- Pro určení toho, který uzel prohledávat se využívá hodnotící funkce:

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

  - f – celková hodnota ceny uzlu
  - g – hodnota ceny přechodu z počátečního uzlu do aktuálního

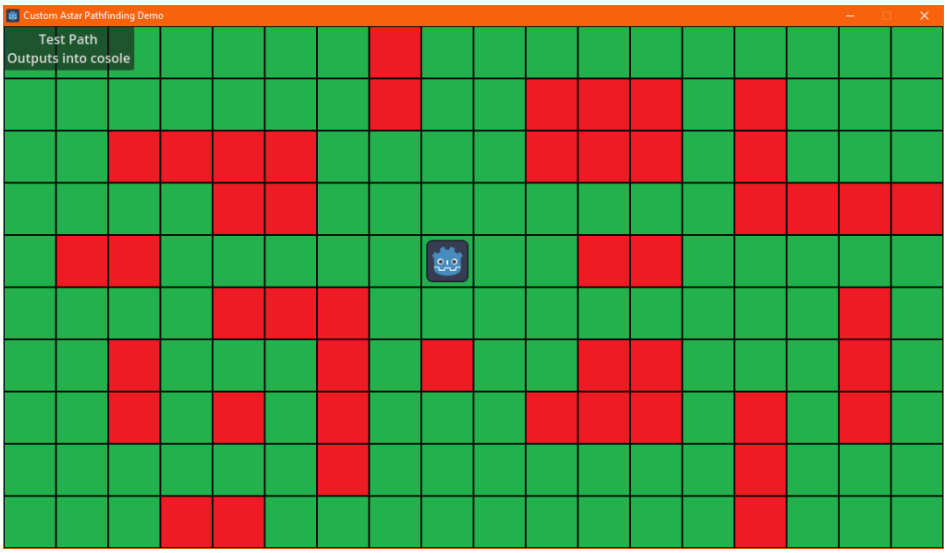
$$g(j) = g(i) + c(i,j)$$

    - $c(i, j)$ – cena přechodu z uzlu i do uzlu j
  - h – heuristická funkce určující většinou vzdálenost od cílového uzlu

Popis algoritmu:

1. Zapsání počátečního stavu do seznamu OPEN
2. Pokud je seznam OPEN prázdný neexistuje řešení a prohledávání je ukončeno
3. Vybrání stavu i s nejmenší hodnotou $f(i)$ ze seznamu OPEN. V případě většího množství stavů se stejnou hodnotou c zkontrolovat, zda není některý z nich cílový stav, pokud ano vybrat jej
4. Přesun vybraného uzlu z OPEN do CLOSED (vymazání z OPEN)
5. Je-li stav i cílový ukončit prohledávání – řešení nalezeno
6. Expanze stavu i. Pro každého následníka stavu i vypočítat hodnotu f
   - Pokud stav není ani v OPEN ani v CLOSED zařadit jej do OPEN
   - Pokud je stav j v OPEN nebo CLOSED ale s hodnotou f vyšší než právě vypočtená přepsat tuto hodnotu na právě vypočtenou a změnit jeho rodiče na aktuální uzel i. Přitom pokud je stav j v CLOSED přesunout jej zpět do OPEN a vymazat z CLOSED
7. Pokračování krokem č. 2.

Uživatelský popis aplikace



Obrázek 4 Ukázka spuštěné "hry"

Obrázek 5 "postavička" agenta

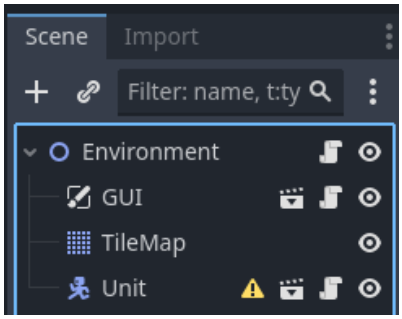
Popis obsahu okna

- Agent – „postavička“ která se pohybuje po cestě nalezené A*.
- Tlačítko „Test Path“ – spouští prohledávání mezi pevně nastavenými body (0, 0) a (15, 7), indexování začíná od 0 z levého horního rohu. Výstup se nepromítne na pohybu agenta, ale pouze se vypíše do konzole. Pro zobrazení konzole spusťte *Custom_Astar_Demo.console.exe*.
- Zelená pole – Pole volná pro pohyb.
- Červená pole – Překážky.

Ovládání dema

Ovládání je realizováno pouze levým tlačítkem myši, při kliknutí na jakékoliv pole se spustí prohledávání a pokud je nalezena cesta spustí se pohyb agenta.

Popis kódu



Obrázek 6 Strom prvků scény programu

Program obsahuje 4 skripty psané v jazyku GDScript:

- gui.gd – připojen k GUI.tscn
- Custom_AStar.gd - nepřipojen
- environment.gd – připojen k Environment.tscn
- unit.gd – připojen k Unit.tscn

Tyto skripty jsou připojovány k předefinovaným objektům, a dědí tím jeho vlastnosti a funkce.

gui.gd

```
1   extends CanvasLayer
2
3   @onready var env = $".."
4
5
6
7   func _on_button_pressed():
8       env.test_function()
```

- extends je funkce pro předání parametrů a funkcí objektu CanvasLayer do skriptu
- @onready je funkce která se spouští pokaždé když je daný objekt načten do aktuální scény v tomto případě se do proměnné env přenáší reference na celý objekt Environment.tscn ($“..“ je ukazatel kořenový prvek ve stromu prvků)
- Funkce _on_button_pressed() je spuštěna po přijetí signálu z tlačítka „Test Path“ a spouští testovací funkci objektu Environment.tscn.

unit.gd

```
1   extends CharacterBody2D
2
3   @onready var Tilemap = $"../TileMap"
4
5   const SPEED = 300.0          # unit movement speed
6   const min_target = 32        # minimal distance from current possition to target
7   const eps = 10               # distance under which the point is considered reached
8
9   var path = Array()           # Array for Path points
10  var move_to = Vector2()      # coordinates of current movement target
11  var to = Vector2()           # coordinates of current movement target (internal for function move())
```

- Extends – předání dat z objektu CharacterBody2D
- @onready var Tilemap – Tile map je objekt pro vytváření mřížky polí ať už se jedná o terén nebo jen pozadí. Zde je reference na tento objekt přiřazena proměnné pro pozdější použití
- Const – konstanta (nelze upravovat v kódu, pouze číst)
- Var – promněnná (lze upravovat uvnitř kódu)

- SPEED – rychlost pohybu agenta
- Min_target – minimální vzdálenost cíle pro spuštění hledání cesty
- Eps – tolerance odchylky od cílového bodu, aby byl uvažován jako dosažený
- Path – list vektorů pozic cesty do cílového bodu
- Move_to / to – vektory souřadnic pro aktuální bod cesty

```
15 func _ready():
16     path.clear()        # deletes all points from path
17 
18     # rounds global coordinates to map coordinates using
19     # transformation from global to local and back
20     position = Tilemap.map_to_local(Tilemap.local_to_map(position))
21 
22     move_to = position    # sets target to current possition
23     velocity = Vector2(0, 0)# sets speed to 0
```

- Path.clear() – funkce clear() je interní funkcí pro Array a odstraní všechny prvky tohoto listu
- Position – pozice objektu unit v pixelech v prostoru scény
- Map_to_local / local_to_map transformují souřadnice z lokálních souřadnic scény v pixelech na souřadnice v objektu Tilemap (každé pole má souřadnice jako by bylo jeden pixel) a zpět
- Move_to = position nastavuje jako cíl pohybu aktuální pozici (zabránění náhodného pohybu při startu)
- Velocity = … - nastavení rychlosti pohybu na 0

```
27 func _physics_process(delta):    # physics proces runs every frame
28 
29     # if distance from target is greater than minimal set velocity
30     if position.distance_to(move_to) > min_target:
31         velocity = position.direction_to(move_to) * SPEED
32 
33     move_and_slide()    # internal function for movement physics
34 
35     #if distance to target is smaller than tolerance stop movement
36     if position.distance_to(move_to) <= eps:
37         if path.is_empty():# if path array is empty stop movement
38             _ready()
39         else:        # else continue to next point
40             move_to = move(path)
```

- Ř. 30 – pokud je vzdálenost od cíle větší než nastavené minimum nastavit hodnotu rychlosti na daný směr k cíli a danou velikost (rychlost je 2 složkový vektor x a y)
- Ř. 33 – interní fyzikální funkce pro pohyb těles
- Ř. 36 – kontrola dosažení cíle
- Ř. 37 – pokud je list cesty prázdná spustí funkci _ready() která resetuje a zastaví pohyb agenta
- Ř. 39 – pokud cesta není prázdná nastaví další její bod jako nový cíl

```
44  func move(path):
45      if not path.is_empty():
46          to = path.pop_front()
47      else:
48          to = position
49      return to
50
51
52
53  func debug():
54      print('Unit path: ',path)
```

- Funkce move() vybírá první prvek ze seznamu cesty a přiřadí ho proměnné to kterou vrací jako výstup. Pokud je cesta prázdná nastaví cíl na aktuální pozici agenta
- Funkce debug() vypisuje do konzole cestu přijatou z algoritmu A*

environment.gd

```
1   extends Node2D
2
3   @onready var tilemap = $TileMap
4   @onready var unit = $Unit
5
6   var Astar = preload("res://Custom_AStar.gd").new()
7
8   var path = Array()
9
10
11  # DEBUG TOOLS
12  var debug_toggle = false
13  var debug
```

- extends – předání parametrů a funkcí objektu
- tilemap – objekt mapy mřížky polí scény
- unit – předání objektu Unit.tscn ze scény do proměnné
- Astar = preload(...).new() – preload načte objekt v závorkách do paměti, v tomto případě jde o skript algoritmu A*, a .new() vytvoří jeho instanci. Ta je poté předána do proměnné Astar
- path – list pro body cesty
- DEBUG TOOLS – proměnné pro ovládání debug režimu skriptu

```
17  func _ready():
18      unit.motion_mode = unit.MOTION_MODE_FLOATING
```

- _ready() – spuštěna při načtení do paměti
- Ř. 18 – nastavení pohybového režimu agenta tak aby korespondoval s pohybem při pohledu ze shora

```
22 func test_function():
23     var start = Vector2i(32, 32)
24     print(start)
25     var end = Vector2i(1000, 500)
26     print(end)
27 
28     debug = Astar.Find_path(tilemap, start, end, true)
29     debug_print(debug)
```

- Test_function() – funkce spouštěná z objektu GUI tlačítkem nikoliv kliknutím do prostoru. Jsou v ní pevně nastavené polohy startu a cíle cesty a funguje pro otestování funkčnosti algoritmu A* aniž by byl potřeba agent. Její výstup je vypsán pouze do konzole

```
33 func _unhandled_input(event):
34     if event is InputEventMouseButton and event.button_index == MOUSE_BUTTON_LEFT:
35         if event.is_pressed():
36             var target = get_local_mouse_position()
37             path.clear()
38             path = Astar.Find_path(tilemap, unit.position, target, debug_toggle)
39 
40             if debug_toggle:
41                 debug_print(path)
42                 if not path.is_empty():
43                     path = path.pop_front()
44 
45             unit.path = path
46             unit.debug()
```

- _unhadeled_input(event) - hlavní funkce tohoto skriptu. Je spouštěna automaticky interně při jakékoliv interakci s oknem programu.
- Ř. 34 a 35 – kontrola jaký vstup byl aktivován, v tomto případě, zda bylo stisknuto levé tlačítko myši
- Ř. 36 – získání lokální pozice kurzoru myši a její zapsání do proměnné target
- Ř. 37 a 38 – vymazání veškerého obsahu listu cesty a následné spuštění algoritmu A*. startovací bod je pozice agenta a cílový bod jsou souřadnice kurzoru
- Ř. 40–43 – pokud je aktivní debug režim výstup A* jsou dva listy. Zde jsou tyto listy odděleny
- Ř. 45 a 46 – předání listu cesty agentu a spuštění jeho funkce debug()

```
50 func debug_print(_debug):
51     if not _debug.is_empty():
52         print('Path with parents: ', _debug[1])
53         print('Outputed Path: ', _debug[0])
```

- Při debug režimu je výstup A* proměnná dvou listů cesty. Na první pozici je cesta v lokálních souřadnicích, která je poté předána agentu. Na druhé je dvousložkový list všech uzlů cesty, první složka je souřadnice uzlu a druhá souřadnice rodiče tohoto uzlu

Custom_AStar.gd

```gdscript
extends Object

class_name Custom_AStar

var Open = Array()
var Closed = Array()
var Path = Array()

var tile_cost = int()       # Cost of move onto this tile
var f = float()             # Total cost
var g = float()             # Cost to this tile
var h = float()             # Heuristic cost

var start_point = Vector2i()    # Global coordinates of starting point
var end_point = Vector2i()      # Global coordinates of starting point
var path = PackedVector2Array()     # Array of path point's coordinates

var coords = Vector2i()         # Coordinates vector
var walkable = int()            # Walkability of tile
var parent_coords = Vector2i()  # Parent tile coordinates
var Tile = [f , g, tile_cost, coords, parent_coords, walkable] # Data layout for each tile

var Tile_map                # Variable for Tilemap Object
```

- Extends – předání parametrů a funkcí obecného objektu „Object“ do tohoto skriptu
- Class_name – nastavení jména objektu reprezentovaného tímto skriptem
- Open – list uzlů k prohledání
- Closed – list již prohledaných uzlů
- Path – list uzlů cesty i s jejich rodiči
- Tile_cost – cena přechodu na tento uzel (pro toto demo jsou všechny oceněny na 1, v praxi by např. bažina mohla mít ohodnocení 5 a beton např. 2)
- F – celková hodnota uzlu
- G – dosavadní hodnota pro cestu do tohoto uzlu
- H – heuristická hodnota ceny, zde je použita Euklidovská vzdálenost od cíle
- Start_point – souřadnice startovního bodu prohledávání
- End_point – souřadnice cílového bodu prohledávání
- Coords – univerzální vektor pro přenášení souřadnic bodů
- Walkable – proměnná pro rozlišení dosažitelných uzlů od překážek
- Parent_coords – univerzální proměnná pro souřadnice rodiče uzlu
- Tile – „objekt“ pro skladování dat jednotlivých uzlů. Hodnoty indexů od 1. jsou následovné:
    0. f – celková cena uzlu – použito pro výběr uzlu k expanzi
    1. g – dosavadní cena cesty do tohoto uzlu
    2. tile_cost – cena přechodu na tento uzel
    3. coords – souřadnice uzlu
    4. parent_coords – souřadnice rodiče tohoto uzlu
    5. walkable – dosažitelnost uzlu
- Tile_map – proměnná pro objekt mapy polí mřížky

```
# Array for searching through neighboring tiles
const search_array = [Vector2i(1, 0), Vector2i(1, 1), Vector2i(0, 1), Vector2i(-1,1), \
                      Vector2i(-1,0), Vector2i(-1,-1), Vector2i(0,-1), Vector2i(1,-1)]

enum Tile_state { Walkable, Obstacle}
```

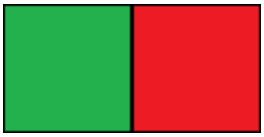
- Search_array – list pro prohledávání sousedů, tento funguje pro čtvercovou a izometrickou mřížku
- Tile_state – hromadná generace proměnných pro dosažitelnost uzlu

```
func clear():    # Clears all variables
    Open.clear()
    Closed.clear()
    Path.clear()
    tile_cost = 0
    f = 0
    g = 0
    h = 0
    start_point = Vector2i(0,0)
    end_point = Vector2i(0,0)
    path.clear()
    coords = Vector2i(0,0)
    walkable = 0
    parent_coords = Vector2i(0,0)
    Tile_map = null
```

- funkce clear() resetuje všechny výše popsané proměnné na základní hodnoty

```
func pull_Tile_map_data(tile_map, coords):
    var map_coords = coords
    var atlas_coords = Tile_map.get_cell_atlas_coords(0, map_coords)

    if atlas_coords[0] == Tile_state.Walkable:
        walkable = 0
    elif atlas_coords[0] == Tile_state.Obstacle:
        walkable = 1

    tile_cost = 1 # For this demo all walkable tiles have cost of 1

    Tile[0] = null    # Total cost (f) is initially unknown
    Tile[1] = 0
    Tile[2] = tile_cost
    Tile[3] = map_coords
    Tile[4] = null    # Parent tile is added later
    Tile[5] = walkable

    return Tile.duplicate()
```

- tato funkce „vytahuje“ data z objektu Tile_map, a to konkrétně dosažitelnost daného pole. Dále zapisuje do proměnné Tile základní hodnoty pro dané pole
- atlas_coords – Atlas je grafický soubor (např. PNG) ve kterém jsou vedle sebe umístěny jednotlivá pole. Souřadnice atlasu mají x a y složku, v našem případě má dosažitelné pole souřadnice (0, 0) a překážka (1, 0)



Obrázek 7 Atlas polí pro toto demo (zelené pole – dosažitelné, červené - překážka)

- aby nedocházelo k referování a poté přepisování již existujících hodnot polí tak tato funkce vrací kopii proměnné Tile pomocí integrované funkce duplicate()

```
73  func Search_neighbors(tile_map, current_coords):
74
75      # in square grid with diagonal movement there is always 8 neighbors => hardcoded number of neighbors
76      var neighbors_coords = [null, null, null, null, null, null, null, null]
77      var neighbors = [null, null, null, null, null, null, null, null]
78      var neighbors_out = Array()
79
80      for i in search_array.size():       # Collecting all neighbors
81          neighbors_coords[i] = current_coords + search_array[i]
82          neighbors[i] = pull_Tile_map_data(tile_map, neighbors_coords[i])
83
84      for n in neighbors.size():          # Outputed are only tiles that are walkable
85          if neighbors[n][5] == Tile_state.Walkable:
86              neighbors_out.append(neighbors[n])
87
88      return neighbors_out.duplicate()
```

- Tato funkce prohledává sousedy právě expandovaného uzlu. Pro toto demo je pevně nastaveno 8 sousedů (4 přímí a 4 diagonálně)
- Neighbors_coords – list souřadnic sousedů
- Neighbors – list vlastních proměnných Tile
- Neighbors_out – list uzlů které budou vráceny touto funkcí, není nutné vracet uzly které jsou překážky
- Ř. 80 – cyklus pro postupné získání proměnné Tile pro všechny sousedy
- Ř. 84 – cyklus pro vyfiltrování nedosažitelných uzlů

```
91  func calculate_cost(parent_cost, coords, end_point, tile_cost):
92
93      var c = Array()
94      c.resize(4)
95
96      c[0] = max(end_point[0], coords[0])
97      c[1] = min(end_point[0], coords[0])
98      c[2] = max(end_point[1], coords[1])
99      c[3] = min(end_point[1], coords[1])     # Calculations to secure positive numbers of h
100
101     h = sqrt((c[0] - c[1])**2 + (c[2] - c[3])**2)  # Heuristic function - Euclidian Distance
102
103     g = parent_cost + tile_cost             # New cost for the currently expanded tile
104
105     f = g + h                               # Total cost
106
107     var transfer = Vector2(f, g)
108
109     return transfer
```

- Tato funkce vypočítá hodnoty ohodnocení f a g pro daný uzel a vrátí tyto hodnoty pomocí pomocného vektoru transfer
- Ř. 93-99 seřazení a příprava souřadnic pro výpočet heuristické funkce.
- H – výpočet heuristické funkce, zde je to Euklidovská vzdálenost
- G – výpočet nové ceny tohoto uzlu
- F – celková cena uzlu

Hlavní funkce A - Find_path*

```
113 func Find_path(tile_map, start_point, end_point, debug = bool(false)):
114     print('\n A* START \n')
115     
116     clear()
117     var expanded_tile
118     Tile_map = tile_map
119     
120     start_point = Tile_map.local_to_map(start_point)
121     var start_tile = pull_Tile_map_data(Tile_map, start_point)
122     start_tile[0] = 0
123     start_tile[1] = 0
124     print('start_tile: ',start_tile)
125     Open.append(start_tile)
126     
127     #var debug_point = pull_Tile_map_data(Tile_map, Vector2i(0, 0))
128     #debug_point[0] = 1
129     #debug_point[1] = 0
130     #print('debug_point: ', debug_point)
131     #Open.append(debug_point)
132     
133     end_point = Tile_map.local_to_map(end_point)
134     var end_tile = pull_Tile_map_data(Tile_map, end_point)
135     end_tile[0] = 0
136     end_tile[1] = 0
137     print('end_tile: ', end_tile)
138     
139     if end_tile[5] == Tile_state.Obstacle:
140         return Path
```

- Ř. 116 – resetování všech proměnných
- expanded_tile – proměnná pro právě expandovaný uzel
- Ř. 120-137 – ze vstupních souřadnic startovního a koncového uzlu se získají a zapíšou jejich vlastnosti, startovní uzel se navíc zapíše do seznamu OPEN
- Ř. 139 – podmínka ukončení hledání v případě že koncový uzel je překážka

*Začátek hlavního cyklu A**

```
142 while not Open.is_empty(): # main loop of A*
143     Open.sort()
144     # end point check when multiple tiles have (aproximately) same cost
145     var count = Open.count(Open[0][0])
146     if count > 1:
147         for i in count:
148             if int(Open[i][3]) == int(end_point):
149                 expanded_tile = Open.pop_at(i)
150                 Closed.append(expanded_tile)
151     else:
152         # transfer curently expanded tile from open to closed
153         expanded_tile = Open.pop_front()
154         Closed.append(expanded_tile)
155 
156     #print('expanded_tile:', expanded_tile)
157 
158     # search end-condition check (end-point check)
159     if expanded_tile[3] == end_point:
160         print('\n END REACHED \n')
161         #return [Open, Closed]
162         break
163 
164     # expand current tile
165     var neighbors_list = Search_neighbors(Tile_map, expanded_tile[3])
```

- Hlavní while cyklus je opakován dokud není seznam OPEN prázdný nebo dokud není ukončen vnitřní podmínkou
- První krok je seřazení seznamu OPEN, jelikož je hodnota f jako první pro každý prvek seznamu je tento seznam seřazen podle této hodnoty
- Ř. 144-154 – kontrola seznamu OPEN na větší počet stejně ohodnocených uzlů, pokud je jeden z nich koncový stav bude vybrán ten, pokud ne je vybrán první prvek seznamu OPEN
- Ř. 158-162 – kontrola ukončovací podmínky. Pokud je právě expandovaný uzel koncový uzel prohledávání je ukončeno
- Ř. 165 – expanze aktuálního uzlu funkcí Search_neighbors

```
167     # Open and Closed check for all child tiles
168     if Open.size() > 0:    # Checking for ocurence of childs in OPEN
169         for o in Open.size():
170             for n in neighbors_list.size():
171                 if Open[o][3] == neighbors_list[n][3]:
172                     var cost = calculate_cost(expanded_tile[1], neighbors_list[n][3], \
173                                               end_point, neighbors_list[n][2])
174                     neighbors_list[n][0] = cost[0]
175                     neighbors_list[n][1] = cost[1]
176 
177                     if Open[o][0] > neighbors_list[n][0]:
178                         neighbors_list[n][4] = expanded_tile[3]
179                         Open[o] = neighbors_list.pop_at(n)
180                         break
181                     else:
182                         neighbors_list.remove_at(n)
183                         break
```

- Kontrola přítomnosti sousedů expandovaného uzlu v seznamu OPEN
- Pokud je nová vypočtená cena souseda menší než ta která je v seznamu OPEN nastaví se expandovaný uzel jako rodič tohoto souseda a takto upravený se nahradí v seznamu OPEN a odebere se ze seznamu sousedů.

- Pokud nová vypočtená cena není menší než již existující v seznamu OPEN tento soused se odebere ze seznamu sousedů

```
185    if Closed.size() > 0:    # Checking for ocurence of childs in CLOSED
186      for c in Closed.size():
187        for n in neighbors_list.size():
188          if Closed[c][3] == neighbors_list[n][3]:
189            var cost = calculate_cost(expanded_tile[1], neighbors_list[n][3], \
190                                      end_point, neighbors_list[n][2])
191            neighbors_list[n][0] = cost[0]
192            neighbors_list[n][1] = cost[1]
193
194            if Closed[c][0] > neighbors_list[n][0]:
195              neighbors_list[n][4] = expanded_tile[3]
196              Closed[c] = neighbors_list.pop_at(n)
197              break
198            else:
199              neighbors_list.remove_at(n)
200              break
```

- Zde se provádí identická kontrola, jako pro seznam OPEN, pro seznam CLOSED

```
202    if neighbors_list.size() > 0:    # Pushing the rest of children into OPEN
203      while neighbors_list.size() > 0:
204        for i in neighbors_list.size():
205          var cost = calculate_cost(expanded_tile[1], neighbors_list[i][3], \
206                                    end_point, neighbors_list[i][2])
207          neighbors_list[i][0] = cost[0]
208          neighbors_list[i][1] = cost[1]
209          neighbors_list[i][4] = expanded_tile[3]
210          Open.append(neighbors_list.pop_at(i))
211          break
212
213    # repeat (end of while here)
```

- Zapsání zbylých sousedů, kteří nejsou ani v OPEN ani v CLOSED do seznamu OPEN
- Poté je konec hlavního While cyklu – opakování, dokud není vnitřně ukončen

```
215    # pushing coordinates from Closed into Path
216    for i in Closed.size():        # searching and pushing end tile into Path first
217      if Closed[i][3] == end_point:
218        var node = [Closed[i][3], Closed[i][4]]
219        Path.append(node)
220        break
```

- Od řádku 215 do konce skriptu je zařazování uzlů do listu cesty a následné vrácení hodnoty této cesty do skriptu ze kterého byla zavolána
- Ř.216-220 – prohledání listu CLOSED pro koncový uzel, při jeho nalezení se zapíše do listu cesty jako první (na konec)

```
222    while Path.front()[0] != start_point:    # searching and pushing all other tiles depending on their parent
223      for i in Closed.size():
224        if Closed[i][3] == Path.front()[1]:
225          var node = [Closed[i][3], Closed[i][4]]
226          print('node: ', node)    # console printing of subsequent tiles
227          Path.push_front(node)
228          break
```

- Prohledání seznamu CLOSED a postupné řazení uzlů do seznamu cesty podle jejich rodičovského uzlu, toto prohledávání probíhá dokud není na začátku seznamu startovací uzel

```
230     var Path_out = Array()
231     for i in Path.size():                    # Transforming coordinates from internal to global
232         Path_out.append(Tile_map.map_to_local(Path[i][0]))
233
234     if debug:
235         var ret_deb = [Path_out, Path.duplicate()]
236         Path.clear()
237         return ret_deb
238     else:
239         return Path_out.duplicate()
240
```

- Path – seznam 2 složkových prvků [souřadnice uzlu, souřadnice rodiče]
- Path_out – seznam souřadnic jednotlivých uzlů
- Ř.230-232 – transformace souřadnic ze souřadnic mapy na lokální a zároveň přenášení souřadnic uzlů, už bez souřadnic rodičů, do seznamu Path_out
- Ř.234-239 – pokud je použito debug funkce je vrácen list listů ve kterém jsou obsaženy jak list s rodiči, tak list lokálních souřadnic bez rodičů. Pokud není použito debug funkce je vrácen pouze list lokálních souřadnic bez rodičů

Zdroje

A* - Přednášky od Pana doktora RNDr. Jiří Dvořák, CSc.

Godot engine - https://godotengine.org/ (celý Engine verze 4.2 má velikost 103 MB a je v podobě bez-instalační aplikace)

GDScript dokumetace - https://docs.godotengine.org/en/stable/