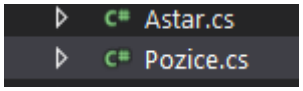
První průběžná práce INUI1

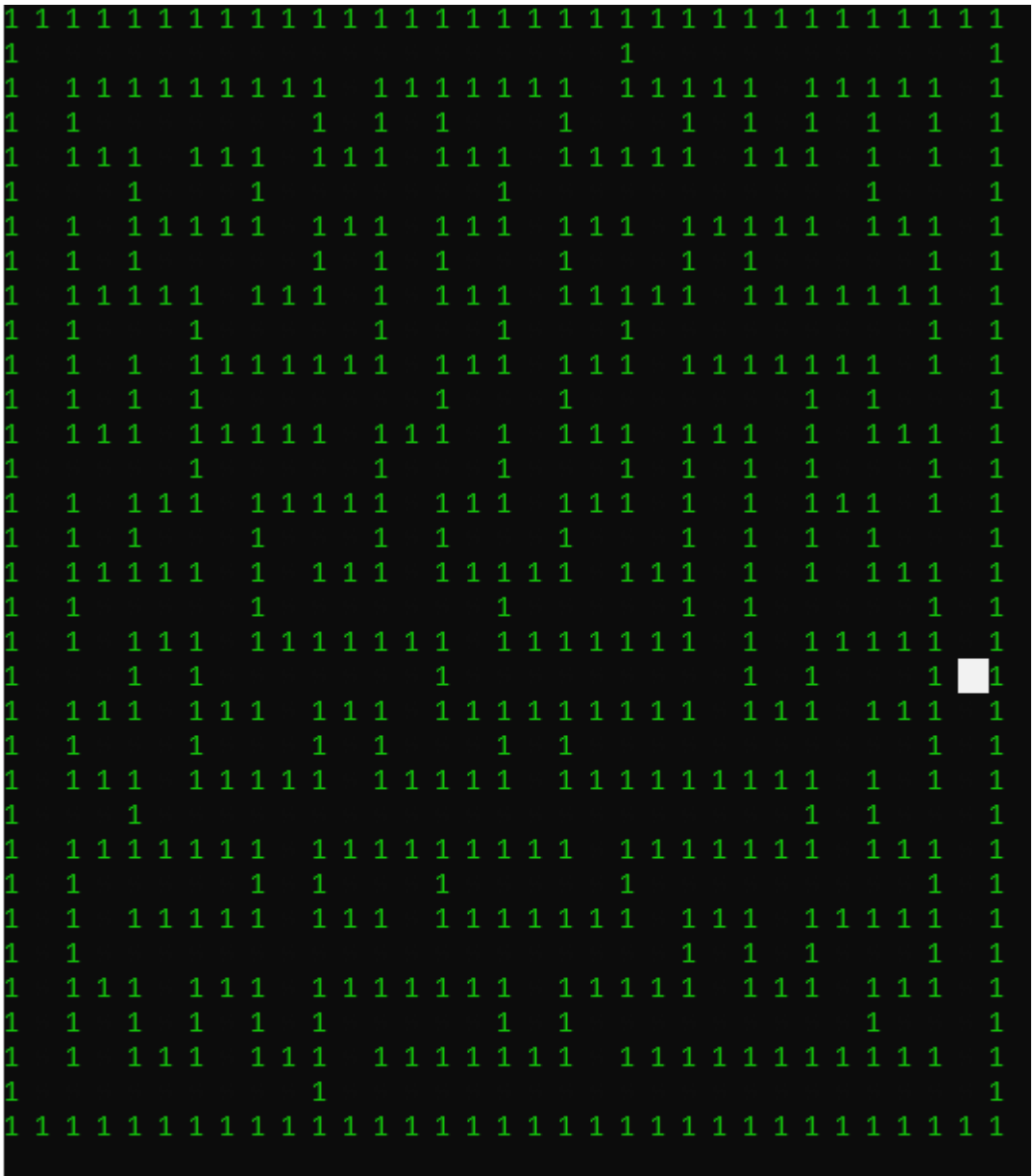
Autor práce: Miroslav Hájek

Složení projektu

- Pro řešení semestrální práce jsem se rozhodl využít programovacího jazyku C# a IDE Visual Studio pro vývoj
- Projekt se skládá se dvou tříd
    - **Astar** (třída, která zajišťuje implementaci A* algoritmu)
    - **Pozice** (vnitřní třída, která se využívá pro popis pozice agenta v hrací desce, tak aby bylo možné ukládat všechny informace o pozici do jediné ,,proměnné,,)



- Hlavní třída **program**.cs obsahuje pouze vytvoření instance **Astar** a vyvolání metody **PohybujSe**, která realizuje posun po hrací desce.
- Hrací desku bludiště realizuji pomocí metody, která rozloží obrázek na jednotlivé body a vytvoří matici složenou z 0 a 1, která složí pro běh programu.
- **Při spuštění programu je MAZE1.bmp** hledán na cestě D:\MAZE1.bmp, proto je nutné pro správný běh programu vložit obrázek na tuto pozici, aby bylo možné jej načíst a spustit hru. Případně je nutné v kódu programu **upravit cestu na 34 řádku třídy Astar**, tak aby odpovídala uložení obrázku.
- Program je demonstrován vykreslením do konzole, tak, že ukazuje celou herní plochu a také aktuální pozici agenta v hracím poli (**bílá kostička**). Hranice bludiště, jsou naznačeny pomocí čísel 1.
- Současně je pod tímto hracím polem vypsán aktuální obsah kolekce **NEXT**, aby bylo možné pozorovat, jaké nové pozice byli objeveny a zda byli opravdu poté navštíveny.
- Program končí po nalezení cílové pozice, která je v tomto případě nastavena v pravém dolním rohu. Informaci o vítezství označuje červeně zvýrazněný řádek v obrázku níže.
- Program nalezne řešení rychleji, než například prosté prohledávání do šířky nebo hloubky, avšak vzhledem k jednoduché heuristické funkci, je tato odlišnost malá. V případě použití lepší heuristické funkce a mnohem většího bludiště, by byly kvality algoritmu lépe viditelné.
- **Spuštění programu**
    - **Windows 10**
        - **Adresář lze stáhnout:** https://drive.google.com/drive/folders/1Ld3dCUTM1gW52THS1UcUOo1Sswk_vqT-?usp=sharing
        - Tento soubor se nachází v adresáři aplikace, konkrétně v Bludiste\Bludiste\bin\Debug\bludiste.exe
    - **Linux (Ubuntu)**
        - **Využití technologie Mono, která slouží pro multiplatformní aplikaci napsanou v C# (www.mono-project.com)**
        - Poté v termínálu linux zapsat následující příkazy
            - **sudo apt-get install mono-devel**
            - **sudo apt-get install libgdiplus**
        - A samotné spuštění programu pomocí příkazu
            - **mono bludiste.exe**



```
----------NEXT----------
|X      |Y      |AktCena|CelkCena
|X: 31  |Y: 19  | 151   |489
|X: 31  |Y: 31  | 163   |489
|X: 30  |Y: 29  | 161   |492
|X: 19  |Y: 21  | 142   |492
|X: 23  |Y: 17  | 143   |495
|X: 21  |Y: 19  | 144   |498
VYHRA ... nasel jsem cil. Cil jsem nalezl za 122
```

Seznam částí programu

Třídy a jejich metody a atributy

1) **Astar**
   a. **Metody**
      i. Astar()
      ii. `int HeuristickaFunkce(int x, int y, int AktualniCenaCesty)`
      iii. `bool Vyhra(int X, int Y, int beh)`
      iv. `void PohybujSe(int pocetBehu)`
      v. `int[] ExpandujVrcholy(int X, int Y)`
      vi. `bool Duplicita(int x, int y)`
      vii. `void DefinujHraciPole()`
      viii. `void ZobrazHraciPole(int x, int y)`
   b. **Atributy**
      i. `Bitmap Image`
      ii. `Pozice agent`
      iii. `int[,] HraciDeska`
      iv. `int[] Cil,`
      v. `List<Pozice> Stack`
      vi. `List<Pozice> Next`
      vii. `int IDVrcholu`
2) **Pozice**
   a. **Metody**
      i. `Pozice(int id,int x, int y)`
      ii. `Pozice(int id, int aktualniCenaCesty, int x, int y, int smerNatoceni, int celkovaCena)`
      iii. `int[] Pohyb(int smer)`
      iv. `int[] ValidujPozici(int smer)`
      v. `override string ToString()`
   b. **Atributy**
      i. `int ID`
      ii. `int SmerNatoceni`
      iii. `int[] AktPozice`
      iv. `int AktCenaCesty`
      v. `int CelkovaCenaCesty`

Funkce programu

1) Zavolání metody **DefinujHraciPole**, která projde každý pixel obrázku MAZE1.bmp a v případě, že je barva pixelu rovna černé, tak uloží do matice **HraciDeska** 1, v opačném případě, tedy rovno bílé pole, uloží 0.

```
275        private void DefinujHraciPole()
```

2) Definuji v konstruktoru Astar.cs zásobníky **STACK**, NEXT. První zmíněný je zásobník pro uložení navštívených pozic a **NEXT** slouží pro uložení budoucích pozic, objevených při expanzi vrcholu. Současně definuji **AGENTA (políčko, které se pohybuje po hrací desce)**, který je třídy Pozice a obsahuje všechny základní informace o svém ID, aktuální ceně cesty, která se skládá z ceny jednotlivých pohybů a otočení na hrací desce, svých souřadnicích, směru natočení **(0 – sever, 1 – západ, 2 – východ, 3 - jih**) a celkové ceně cesty, která se skládá s aktuální + heuristická funkce.

```
38            Stack = new List<Pozice>();
39            Next = new List<Pozice>();
40            agent = new Pozice(IDVrcholu, 1, 5, 19, 0,1);
```

3) Základní metodou **Astar.cs** je **PohybujSe**, která volá všechny ostatní metody této třídy a současně i metody v **Pozice.cs**. Přijímá jako vstupní parametr počet běhů, tedy počet pohybů agenta v hracím poli. Nejprve se zavolá metoda **ExpandujVrcholy** (viz. **Ad 4**). Získané vhodné směry z této metody uložím do proměnné **expanduj_pomocna (obsahuje 4 hodnoty 0 / 1)**.

```
78        public void PohybujSe(int pocetBehu)
```

Poté procházím všechny hodnoty v této proměnné a pokud narazím na 1 – tedy vhodný směr, tak zavolám na Agentovi metodu **Pohyb (metoda třídy POZICE).**

```
54        public int[] Pohyb(int smer)
```

Tato metoda zavolá metodu **ValidujPozici (metoda třídy POZICE)**, která pro daný směr spočítá souřadnice další souřadnice, které je možné z aktuálního vrcholu, tedy pozice AGENTA.

```
92        private int[] ValidujPozici(int smer)
```

Získané souřadnice uložím do pole **report**, které metoda vrací na pozici 1 a 2. Dále je v tomto reportu na pozici 0 uložena aktuální cena cesta po tento vrchol, které je dále zvýšená o příslušnou hodnotu, podle toho, jakým směrem je agent natočen a kolik kroků musí vykonat pro dosažení této nové pozice, například se musí otočit o 90 stupňů apod. Poslední hodnota v **report** na pozici 3 je nový směr natočení na této pozici. Tak aby při expanzi této nové pozice bylo možné spočítat opět cenu kroků, jako je otočení apod. Metodou vrácené pole si uložím do proměnné **report_pomocna (ve třídě Astar)**, se kterou dále pracuji. Následně zavolám metodu **HeuristickaFunkce** (**viz. Ad 6**), která spočítá heuristickou funkci pro danou pozici. Na závěr vytvořím novou instanci třídy POZICE, která bude obsahovat všechny parametry popsané výše. Tedy všechny hodnoty z proměnné **report_pomocna** a současně hodnotu celkovou, která obsahuje heuristickou funkci + aktuální cenu cesty. Tuto instanci přidám do kolekce **NEXT,** jako budoucí možnou pozici a kolekci seřadím vzestupně podle celkové ceny cesty. Tento proces opakuji pro všechny 4 směry, tak abych prozkoumal všechny možné

směry z dané pozice. Vytvořím tím maximálně tedy 4 instance třídy POZICE, které nesou kompletní informace o dané pozici. Posledním krokem je uložení aktuální pozice (**pozice v AGENTOVI**) do kolekce **STACK** a uložení nové pozice do agenta, která se nachází na prvním místě kolekce **NEXT.** Tuto pozici samozřejmě poté smažu z next.

4) **ExpandulVrcholy** – vyžaduje na vstup současnou pozici Agenta v hracím poli, tak aby mohl daný vrchol expandovat a prozkoumat čtyři směry. Zda je možné je použít, ať už v závislosti na duplicitě, přítomnosti černého pole. Nejprve spočítám čtyři směry, tedy jejich souřadnice v prostoru a uložím si tyto čtyři souřadnice směrů Sever, jih, východ a západ do proměnné **novaPozice**. Poté procházím tyto čtyři souřadnice a kontroluji u nich zda se jedná o souřadnice kladné. Dále kontroluji duplicitu (viz. **Ad 5**) této pozice s navštívenými v kolekci STACK. Na závěr zkontroluji, zda tato pozice obsahuje 0, tedy mohu jít nebo 1 nelze. Pokud všechny podmínky proběhnou kladně, uložím si do pole **vhodneSmery** 1 pro daný směr. Toto pole poté metoda vrací na svém výstupu a dále se tyto data využívají v **PohybujSe**.

```
150        private int[] ExpandujVrcholy(int X, int Y)
```

5) **Duplicita –** zadané souřadnice na vstup porovnávám se všemi známými, tedy jak kolekci **STACK** tak kolekci **NEXT**, jelikož jsem mohl již tuto pozici objevit v dřívějších expanzích akorát jsem ji prozatím nenavštívil. Pokud pozici nenaleznu ani v jedné kolekci, jedná se o novou, a proto metoda vrátí FALSE. V opačném případě vrací TRUE, jelikož se již pozice shoduje s nalezenou.

```
215        private bool Duplicita(int x, int y)
```

6) **HeuristickaFunkce –** metoda, která spočítá hodnotu heuristické funkce. Pro účely této práce, byla navržena pouze jednoduchá heuristická funkce, avšak pro zlepšení výsledků programu by bylo nutné navrhnout sofistikovanější funkci, tak aby agent si více přibližoval k cíli a zmenšil se počet špatných vrcholů, které v průběhu cesty navštíví. Aktuální podoba je taková, že se spočítá vzdálenost x + vzdálenost y od cílové pozice.

```
48     private int HeuristickaFunkce(int x, int y, int AktualniCenaCesty)
```