Programátorská dokumentace k programu MAPFsimulator

Aplikace je vytvořena v programovacím jazyce C#, přičemž při vývoji byl využit .NET Framework ve verzi 4.6. Pro psaní kódu jsme použili vývojové prostředí *Visual Studio*. Návrh grafického uživatelského rozhraní jsme sestavili pomocí knihovny *Windows Forms*.

Níže se zaměříme na základní princip fungování programu, přehled jeho nejdůležitějších částí a jejich vzájemnou provázanost. Zevrubné komentáře ke všem třídám a metodám lze nalézt ve zdrojových kódech programu a v dokumentaci vygenerované nástrojem *Sandcastle*.

V této dokumentaci využíváme pro popisy oken programu, která tvoří grafické uživatelské rozhraní aplikace, názvy z obrázků nacházejících se v uživatelské dokumentaci. Tato kapitola je součástí textu diplomové práce jako příloha A.1.

Adresářová struktura programu

Celý program se nachází v adresáři *MAPFsimulator*. Kromě zdrojových kódů zde nalezneme další adresáře, obsahující následující součásti programu:

- **Properties** – adresář s uloženým nastavením preferencí uživatele. Jedná se o položky měnitelné v programu, konkrétně v části *Soubor → Nastavení.*
- **bin**
    - **Testy** – adresář obsahující mapy (soubory s příponou *.map*) a seznamy pozic agentů (přípona *.scen*), které byly použité pro testování.
    - **Release** – adresář se spustitelným souborem.
        - **PicatFiles** – v tomto adresáři najdeme zdrojové kódy v jazyce Picat. Jedná se modely hledající klasické, resp. *k*-robustní plány v MAPF problému.
        - **MAPFsimulator.exe** – spustitelný soubor s aplikací

Zdrojové kódy

Grafická okna aplikace

Následující třídy tvoří grafické uživatelské rozhraní aplikace:

- ***AgentLoading*** – okno s výběrem počtu agentů, kteří se nahrají do aktuální MAPF instance. Spuštění proběhne po výběru konkrétního souboru s pozicemi agentů v hlavním okně programu.
- ***BenchmarkRuns*** – okno umožňující provádění testů. Spouští se pomocí *Soubor → Spustit benchmark.*

- ***Form1*** – hlavní okno aplikace, které se zobrazuje po spuštění programu.
- ***PicatTimeWindows*** – okno s časomírou vyvolané při hledání plánu externím Picat řešičem.
- ***PlanTable*** – okno zobrazující barevně zvýrazněné plány, vzniklé exekucí dané MAPF instance. Spouští se z okna *BenchmarkRuns* po poklepání na konkrétní exekuci.
- ***Settings*** – okno s nastavením programu vyvolané po výběru položky *Soubor* → *Nastavení.*

Způsoby hledání řešení MAPF

Algoritmus CBS

Implementace všech řešičů založených na algoritmu CBS se nachází v souboru *CBSsolver.cs.* Obsahuje celkem 4 třídy, které se liší převážně tím, jaké druhy konfliktů řeší.

- **Třída *CBSsolver*** – implementace algoritmu CBS, který vrací optimální řešení neobsahující vrcholové konflikty. Jedná se o základní třídu, od které dědí tři následující třídy.
- **Třída *CBSwithSwapping*** – přidává kontrolu konfliktu vzájemné výměny vrcholů, poskytuje plán, který je validní dle definice použité v textu práce.
- **Třída *I_k_CBSsolver*** – přidává kontrolu *k*-zpožděných konfliktů, poskytuje řešení, které splňuje podmínky *k*-robustnosti. Jedná se o implementaci vylepšeného *k*-robustního CBS algoritmu (Improved *k*R-CBS), o kterém se zmiňujeme ve třetí kapitole práce.
- **Třída *CBSsolverForModifiedMapf*** – hledá alternativní plány pro upravený MAPF problém ve striktní variantě alternativní *k*-robustnosti.

Všechny třídy jsou generické a mohou tak poskytovat optimální řešení dle libovolné účelové funkce, implementující interface *CostFunction.* Jako prohledávácí algoritmus, který používá CBS k hledání plánů na nižší úrovni, využíváme algoritmus A*. Ten dále drobně pozměňujeme pro hledání alternativních plánů a plánů na nižší úrovni upraveného MAPF. Všechny tyto třídy se nacházejí v souboru *Astar.cs*

Picat řešič

V případě volby této možnosti bude vytvořená MAPF instance předána externí aplikaci s Picat řešičem, který začne hledat požadované řešení. V naší aplikaci se současně objeví nové okno definované ve třídě *PicatTimeWindows.* Po nalezení řešení je výsledek předán zpět do aplikace *MAPFsimulator*, kde dojde k jeho zobrazení. Časový limit pro hledání plánu lze nastavit v *Soubor* → *Nastavení*, kromě toho je případně možné i hledání předčasně ukončit příslušným tlačítkem v okně *PicatTimeWindows.*

Picat řešiči se na vstupu předávají dva soubory. První z nich je nutné nově vytvořit, neboť musí obsahovat příslušný graf s pozicemi agentů. Tento dočasný soubor se vytváří s unikátně zvoleným jménem v adresáři, kam operační systém ukládá dočasné soubory (konkrétně se zde volá metoda *Path.GetTempFileName()* z namespace *System.IO*). Druhý soubor obsahuje model s deklarativním popisem k nalezení řešení. Vybrán je jeden ze souborů z adresáře *PicatFiles* a to dle požadavků na robustnost plánu.

O veškerou komunikaci s Picat řešičem se stará třída *PicatSolving*.

Komunikace jednotlivých tříd za běhu programu

Vizualizace plánů v hlavním okně programu

Od spuštění programu až po vizualizaci nalezeného plánu projde aplikace několika stavy.
V každém ze stavů je pak možno vidět a ovládat různé položky v hlavním okně programu. Při standardní práci s aplikací projdeme těmito stavy:

- **Stav *Start*** – ve stavu Start se aplikace nachází ihned po jejím spuštění. Dostupná je pouze volba načtení nového grafu. Je vytvořen nový objekt třídy *MapfModel*, který v sobě uchovává data o konkrétní MAPF instanci a objekt třídy *MapfView*, který se stará o zobrazení dat ze třídy *MapfModel* do hlavního okna aplikace.
- **Stav *MapLoaded*** – v části okna *Graf* je vidět nakreslení grafu, objeví se možnost přidávat agenty. Do objektu *MapfModel* se přidá nový objekt třídy *Graph*.
- **Stav *AgentLoaded*** – do stavu přecházíme, pokud je v části okna *Agenti v grafu* načten alespoň jeden agent. Zobrazí se možnost nastavit parametry řešení aktuální MAPF instance (robustnost, řešič, zpoždění agentů). S každým přidáním agenta se vytvoří nová instance třídy *Agent*, která je přidána do modelu.

  Po stisknutí tlačítka **Najít řešení** se na objektu třídy *MapfModel* zavolá metoda *FindSolution()*, které jsou předány uživatelem zvolené parametry řešení. Metoda *FindSolution()* zajistí vytvoření objektu třídy s příslušným řešičem CBS, nebo použije třídu *PicatSolving*. Pokud chceme alternativně *k*-robustní, nebo semi *k*-robustní řešení, provede se zde nejprve úprava hlavního plánu pomocí *wait* akcí. Ta je zajištěna zavoláním metody *GetRobustPlan()* na objektu třídy *WaitAddingRobustness*. Následně je zavolána metoda *GetRobustPlan()* na objektu třídy zajišťující danou formu robustnosti.

  V případě, kdy řešení existuje, provedeme i jeho exekuci zavoláním metody *ExecuteSolution()* (v rámci třídy MapfModel). V ní je vytvořen objekt třídy Execution (případně *ContingencyExecution*, nebo *MinMaxRobustExecution*), na kterém se volá

metoda *ExecuteSolution()*, vracející list abstraktních pozic. Pomocí objektu *MapfView* pak získáme objekty k vykreslení na obrazovku, které zobrazíme v příslušných částech okna.

- **Stav *ComputedSolution*** – zobrazí se nalezené plány a možnost jejich simulace, včetně grafického řešení. Při zvolení konkrétní hodnoty na posuvníku získáme grafické objekty od třídy *MapfView* a zobrazíme je v části okna *Graf*.
- **Stav *NoSolution*** – v případě, že řešení neexistuje, se tato zpráva zobrazí v hlavním okně programu.

Provádění benchmarků v okně pro provádění testů

Zde nejprve uživatel v levé horní části okna vybere a nastaví všechny parametry testu. Testování začíná stisknutím tlačítka **Spustit test**.

Následně je vytvořen nový objekt třídy *MapfModel*, do kterého je nahrán objekt třídy *Graph* a počáteční počet agentů (objektů třídy *Agent*). Řešení MAPF instance hledáme opět voláním metody *FindSolution()* a *ExecuteSolution()* na objektu třídy *MapfModel*. Při více opakování při stejném počtu agentů dojde vždy k vymazání agentů z instance a přidání nových, náhodně vybraných. Při více exekucích jednoho plánu je pro nalezené řešení vícekrát volána metoda *ExecuteSolution()*. Všechny výsledky (plány a jejich exekuce) během testování ukládáme do vytvořených datových struktur, aby bylo možné je zobrazit po ukončení testování.

Metodu *FindSolution()* voláme asynchronně, abychom neblokovali vlákno zobrazující GUI aplikace. V případě stisknutí tlačítka **Zastavit** proto dojde jen k nastavení upozornění, že další volání této metody už nemá proběhnout.

Generátor náhodných čísel a náhoda v programu

Pro generování náhodných čísel program využívá dvě třídy *IntGenerator* a *DoubleGenerator*, obě navržené dle vzoru Singleton.

- ***IntGenerator*** – má na starost generování náhodných čísel, pomocí kterých vybíráme agenty do konkrétní MAPF instance. Resetování generátoru a nastavení seedu určité hodnoty probíhá:
    - po spuštění programu (vždy na stejnou hodnotu),
    - po spuštění testu v okně s benchmarky (vždy na stejnou hodnotu),
    - při změně počtu agentů v rámci daného testu (vždy na stejnou hodnotu v rámci testu, ale různou hodnotu pro každý z testů).

Budeme-li tedy testovat různé robustní techniky na stejných grafech, dostaneme pro každou techniku stejné náhodné instance MAPF problému, takže je můžeme navzájem porovnávat. Při zvyšování počtu agentů zůstanou ti původní na stejných místech. Pokud ale nastavíme opakované provádění testů jedné z technik, bude v každém opakování vytvořena různá sada instancí.

- ***DoubleGenerator*** – má na generování náhodných čísel, pomocí kterých rozhodujeme, zda došlo při exekuci plánu ke zpoždění agenta, či nikoliv. Resetování generátoru probíhá:
    - po spuštění programu (vždy na stejnou hodnotu),
    - po spuštění testu v okně s benchmarky (vždy na stejnou hodnotu).

Budeme-li testovat různé robustní techniky na stejných grafech, dostaneme výskyt zpoždění vždy po pevně daném počtu zavolání metody *NextDouble()* na tomto generátoru. Vzhledem k rozdílným délkám plánů napříč různými robustními technikami se ale zpoždění může týkat rozdílných agentů v rozdílných časech.