Univerzita Karlova, Matematicko-fyzikální fakulta

Dokumentace k programu Diskrétní simulace zoologické zahrady

Letní semestr 2022/23

Adam Řeřicha, I. ročník bakalářského studia, st. skupina 38
28.6.2023

Obsah

1. Anotace

Tento program simuluje návštěvníky procházející zadanou zoologickou zahradou. Program měří časy strávené návštěvníky v zoologické zahradě a vypočítá aritmetický průměr a počítá, kolik návštěvníků stihlo projít všechny stanoviště, které chtěli navštívit, do konce otevírací doby.

2. Přesné zadání

Tento program simuluje návštěvníky procházející zoologickou zahradou načtenou ze vstupního souboru od uživatele, kde jsou vypsané expozice, obchody se suvenýry, stánky s občerstvením a lanovky. Program vypočítá průměrný čas strávený návštěvníkem v zoo a počet návštěvníků, kteří stihli navštívit všechny stanoviště ze svého listu.

V programu se dají nastavovat parametry v grafickém rozhraní Windows Forms jako je seed pseudo-náhodného generátoru čísel, otevírací doba a různé vlastnosti návštěvníků jako například počet stanovišť a občerstvení, které chce navštívit, za jak dlouho návštěvníkovi dojde trpělivost a opustí frontu, za jak dlouho dostane hlad, čas příchodu do zoo a čas udávající za jak dlouho návštěvník dojí své jídlo. Návštěvník může být jeden ze čtyř typů, které se liší způsobem výběru dalšího stanoviště, které navštíví a výběru stánku s občerstvením, kde bude jíst. Je možné také spustit několik výpočtů s různými počty návštěvníků najednou. Všechny vlastnosti návštěvníků se vybírají pseudo-náhodně v zadaných rozsazích. Celý průběh simulace se dá prohlédnout v záznamu log, který se vypisuje v průběhu simulace.

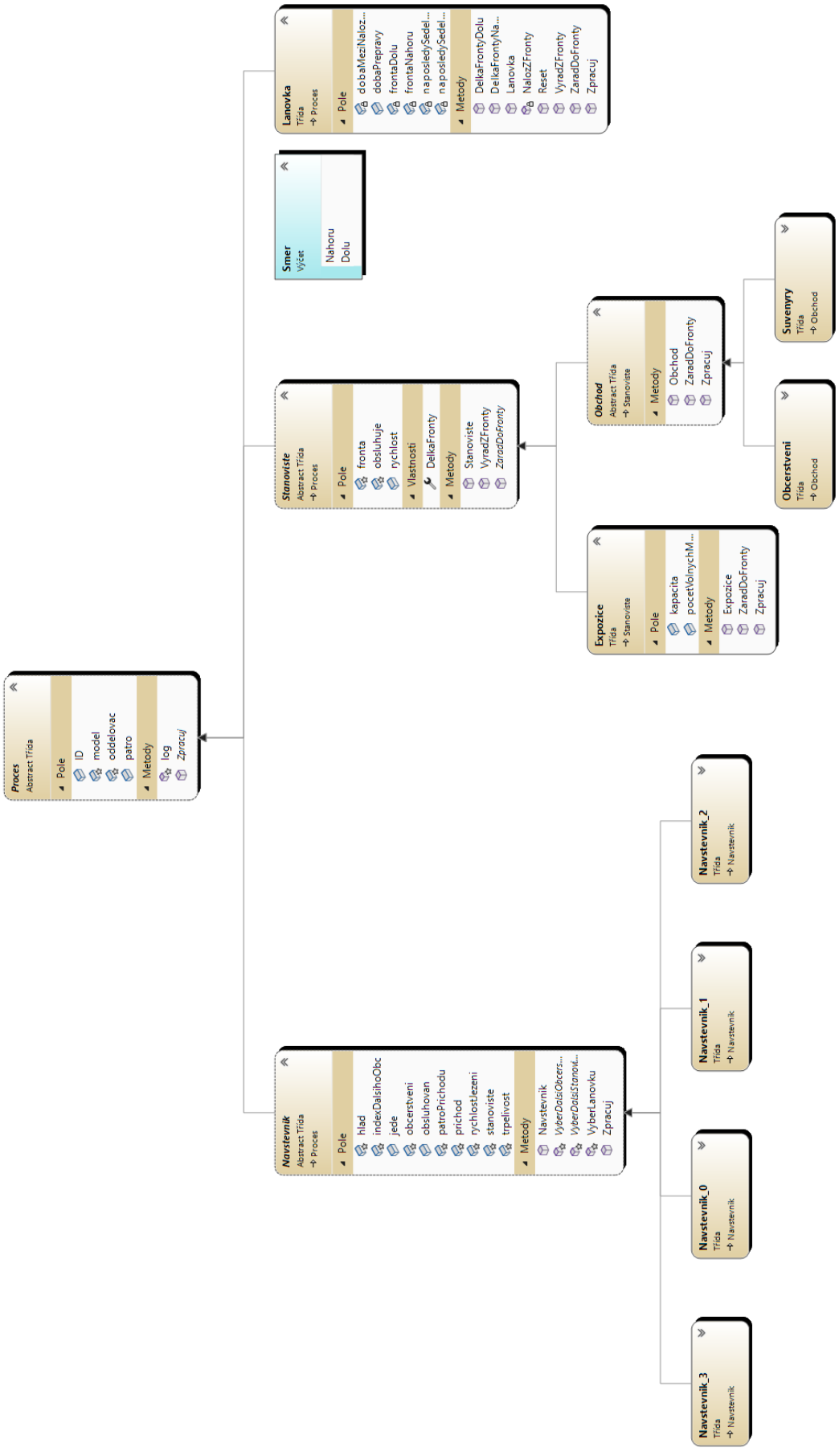
3. Program

3.1. Diskrétní simulace

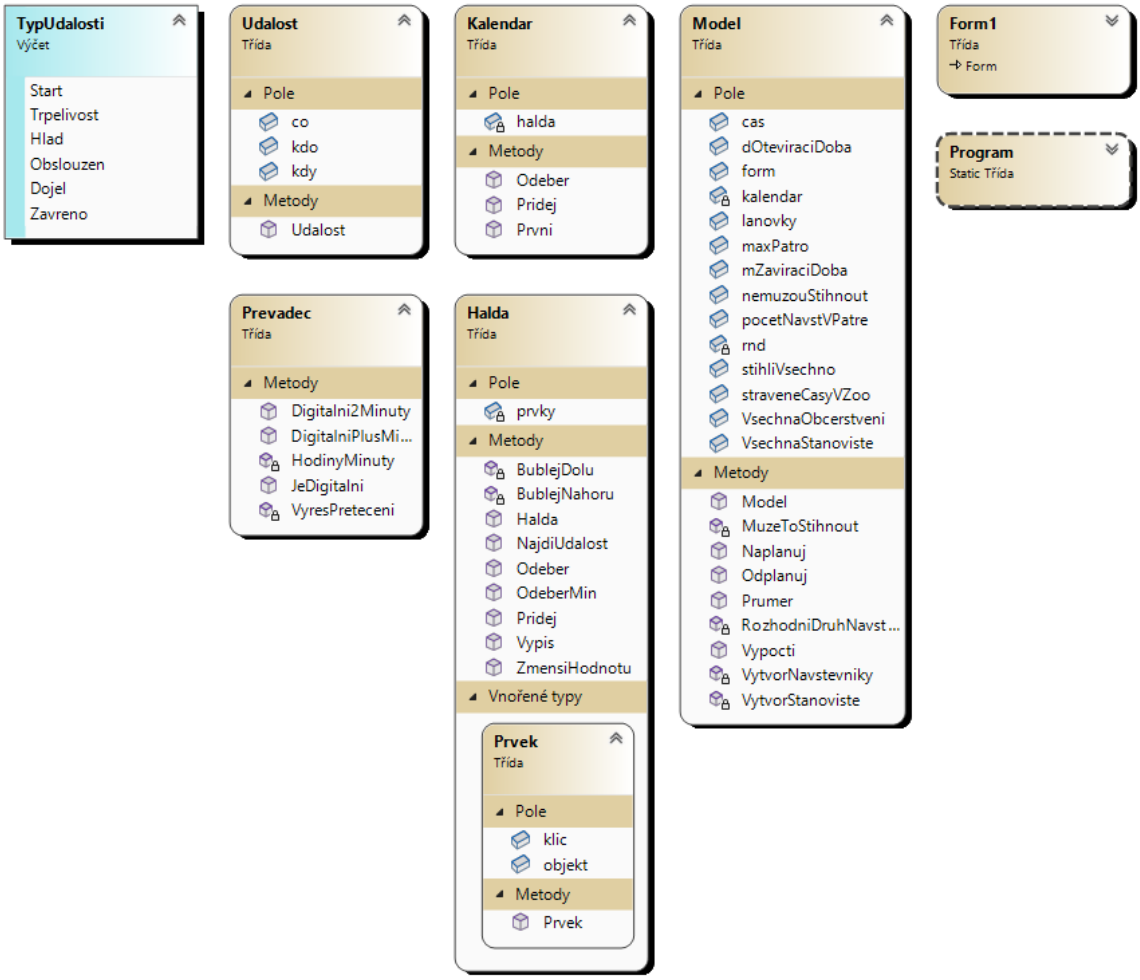
Program funguje na principu diskrétní simulace, takže se vykonávají *události* v pořadí od časově nejbližší, které mají informace *kdy* nastane tato událost, *kdo* ji má zpracovat a *co* má zpracovat. Zpracováním události se může naplánovat další událost a ta se časem také zpracuje, ale musí to někdy skončit. Tento program skončí, pokud už všichni návštěvníci odešli, nebo zoologická zahrada zavřela. Události se plánují do *kalendáře*, který s nimi manipuluje.

Události zpracovávají *procesy*. Každý proces má funkci `Zpracuj(událost)` a každý ji zpracuje tak, jak umí. Zde se hodí využít dědičnost. Když každý proces bude dědit z jedné třídy, která má funkci `Zpracuj(událost)`, tak si tuto metodu přepíše tak, jak má událost zpracovávat a stačí jen iterovat přes události a volat metodu `Zpracuj(událost)` toho procesu, který je v události napsaný.

Model v diskrétní simulaci vytvoří všechny potřebné procesy, spustí simulaci a následně vrátí vypočítané hodnoty. Na dalších dvou stranách je obrázek stromu dědičnosti procesů tohoto programu (Obrázek 1) a ostatních tříd, které vytváří výslednou diskrétní simulaci (Obrázek 2).



Obrázek 1 - Strom dědičnosti procesů



Obrázek 2 - Třídy potřebné pro simulaci a pomocné třídy

3.2. Třídy diskrétní simulace

Každá diskrétní simulace má události, kalendář událostí a model. Následně také mají procesy, ale pro to jsem vymezil samostatnou část. V této části dokumentace popíšu tyto třídy, jak v programu fungují.

3.2.1. Třída Udalost

Všechny procesy umí zpracovat událost typu `Start`. Ostatní události zpracovává jen `Navstevnik`. Událost nosí informaci, kdy nastane, kdo ji má zpracovat a typ události.

Události jsou několika typů. Zde je napsáno, co představují pro procesy:

- `Start` – Obsluhuj/Jdi na další stanoviště
- `Trpelivost` – Návštěvníkovi došla trpělivost, odejde z fronty
- `Hlad` – Návštěvník dostal hlad, musí se jít najíst
- `Obslouzen` – Návštěvník byl obsloužen u stanoviště
- `Dojel` – Návštěvník dojel lanovkou do jiného patra
- `Zavreno` – Nastala zavírací doba zoologické zahrady

3.2.2. Třída Kalendar

Události se ukládají do binární haldy, kterou jsem naimplementoval já. Prvky si ukládá ve struktuře `List<Prvek>`, která je už dostupná s jazykem C#. `Prvek` je objekt, který má dvě proměnné: `int klic`, podle kterého se prvky v haldě řadí, a `Udalost objekt`, kde je samotná událost. Halda se hodí díky tomu, že je potřeba rychle najít událost s nejmenším časem (`klicem`), aby se s tím neztrácel čas. Implementace s `List` není nejoptimálnější, ale přístup k prvkům je pořád asymptoticky čas O(1) a přidání prvku je v průměru O(1) (stránky Microsoft metoda List<T>.Add(T), List<T>.Item[Int32]).

3.2.3. Třída Model

`Model` vytvoří celé zoo podle souboru se vstupními daty (zavolá konstruktory procesů). Pomocí metody `Vypocti(počet_návštěvníků)` se vytvoří všichni návštěvníci s vlastnostmi v rozsazích zadané uživatelem v grafickém rozhraní formuláře a spustí se celá simulace pro daný počet návštěvníků. `Model` sbírá data při simulaci a pak je zpracovává a posílá na výstup formuláři.

Drží aktuální čas, který je roven právě zpracovávané události. Program počítá s časem v minutách od otevírací doby. Pro zobrazení uživateli se tento čas přičítá k otevírací době a zobrazuje se ve tvaru, ve kterém čas zobrazují digitální hodiny. O tento převod se stará `Prevadec`.

Všechny `Expozice` a `Suvenyry` načtené ze souboru se vstupními daty se začlení do slovníku `VsechnaStanoviste`. Všechna `Obcerstveni` budou ve slovníku `VsechnaObcersrveni`. Návštěvníci mají list stanovišť pouze s názvy. Když chtějí interagovat s nějakým stanovištěm, tak se podívají modelu do slovníku.

3.3. Procesy

Abstraktní třída `Proces` je kořenem stromu dědičnosti pro každý proces v programu (`Navstevnik`, `Stanoviste`, `Lanovka`). Má to, co každý proces (`ID`, `patro` a `model`, do kterého se zapíše). Má abstraktní metodu `Zpracuj(událost)` a metodu `log(zprava)`, díky které všechny procesy mohou dát vědět uživateli o tom, co právě dělá.

3.3.1. Třída Navstevnik a jeho druhy

Třída `Navstevník` je abstraktní třída, ze které dědí `Navstevnik_0` až `Navstevnik_3`. V konstruktoru každého návštěvníka se naplánuje do kalendáře událost `Start`, `Hlad` a `Zavreno`, aby se zapsali do pole `straveneCasyVZoo`, jak dlouho byli v zoologické zahradě.

`Navstevnik` má list stanovišť `stanoviste`, které chce ještě navštívit a list občerstvení `obcerstveni`, u kterých bude jíst, když nastane událost `Hlad`. První stanoviště v listu `stanoviste` je považováno za ‚aktivní'. Je to stanoviště, kam se chystá nebo kde právě je obsluhován nebo je zde ve frontě. Odstraní se po obsluze.

Druhy návštěvníků se liší jen výběrem dalšího stanoviště, takže nikdo z nich nemění metodu `Zpracuj(událost)` od `Navstevnik`. To, co se liší jsou metody `VyberDalsiStanoviste()` a `VyberDalsiObcerstveni()`. Pokud je ale aktivní stanoviště `Obcerstveni`, tak si nevybírá a vrátí toto `Obcerstveni`, protože si nemůže naplánovat nic jiného, když je hladový.

`Navstevnik_0` si nijak zvlášť nevybírá. Vezme si to stanoviště, které je v jeho listu stanovišť jako první. Občerstvení si vybírá postupně tak, jak je má v listu a cyklí se pomocí `% obcerstveni.Count`.

`Navstevnik_1` si vybírá stanoviště, které je v patře, ve kterém tento návštěvník právě je. Pokud už prošel všechny stanoviště v tomto patře, tak si vezme to, co má v listu jako první. Občerstvení si vybírá stejným způsobem a pokud nenajde, tak si vybere to, které je na `indexDalsihoObc` v `obcerstveni`.

`Navstevnik_2` si vybírá první stanoviště, které má nejmenší frontu ze všech stanovišť z jeho listu `stanoviste`. Pokud je stanoviště v jiném patře, tak přičte délku fronty u lanovky, ale jen z jeho patra (na ostatní fronty by návštěvník neviděl). Občerstvení si vybírá stejným způsobem.

`Navstevnik_3` si vybírá patro, ve kterém projde všechna svá stanoviště, podle počtu návštěvníků v patře (vybere si to s nejmenším počtem). Počty návštěvníků v patře najde od modelu v poli `pocetNavstVPatre`, které se udržuje v průběhu simulace. S výběrem občerstvení není vybíraví a vybírá ze všech občerstvení z modelu to, které má nejmenší frontu a je v jeho patře.

3.3.2. Třída Stanoviste

Z abstraktní třídy `Stanoviste` dědí `Obchod` a `Expozice`. Jsou to ty třídy, které návštěvníci navštěvují a jsou poté obslouženi. Každé stanoviště má rychlost obsluhy, frontu pro návštěvníky a stav, jestli obsluhuje. Událost Start pro každé `Stanoviste` se vykonává při příchodu návštěvníka do fronty a po obsloužení návštěvníka.

Na rozdíl od Obchodu Expozice má kapacitu. Pokud má volné místo, tak nezařadí do fronty, ale rovnou naplánuje návštěvníkovi událost `Obslouzen`. Pokud je ale kapacita plná, tak se zařadí do fronty. Obchod má jen jednu kasu, u které může být obsloužen jen jeden návštěvník.

3.3.3. Třída Lanovka

`Lanovka` umožňuje návštěvníkům se dostat o jedno patro výše nebo níže. Každé patro může mít jen jednu `Lanovku`. Každá `Lanovka` v `modelu` je ve slovníku `Lanovek` identifikována *nižším patrem* ze dvou, ve kterých se `Lanovka` nachází. Nastoupit se dá vždy po uplynutí doby mezi naloženími, která se píše do souboru se vstupními daty u lanovky. Pokud v čase mezi naloženími někdo nastoupil, tak návštěvník musí počkat ve frontě na další uplynutí doby mezi naloženími. `Lanovka` má frontu návštěvníků čekajících na směr nahoru a dolu. Když se dostane návštěvník na řadu, tak se naplánuje událost `Dojel`. Při zpracování `Dojel` si návštěvník změní patro a adekvátně změní počty návštěvníků v patře.

4. Používání programu

Tato část popisuje, jak se program používá a vysvětluje, jak má vypadat vstupní soubor.

4.1. Formát souboru se vstupními daty

Soubor se vstupními daty obsahuje výpis stanovišť a lanovek, které chcete, aby byly v simulaci. Každý takovýto objekt musí mít u sebe i své specifikace jako například identifikátor. Jeden řádek představuje jeden objekt, pokud první znak na řádku je roven jednomu z těchto:

- `L` – lanovka
- `E` – expozice
- `O` – stánek s občerstvením
- `S` – obchod se suvenýry

Po znaku určující typ objektu následují specifikace. Všechny specifikace i první znak řádku musí být odděleny *tabulátorem* (jedním nebo více). Zde je výpis toho, co jaký objekt potřebuje mít specifikované (specifikace musí být zapsány v tomto pořadí):

- **Lanovka**: ID, nižší ze dvou pater, doba přepravy návštěvníka, doba mezi sedačkami
- **Expozice**: ID, patro, rychlost obsluhy, kapacita
- **Občerstvení**: ID, patro, rychlost (délka objednávání + délka čekání na jídlo)
- **Suvenýry**: ID, patro, rychlost obsluhy

Příklad zápisu expozice Ptačí mokřady:

```
X Expozice | ID                patro        rychlost     kapacita
-------------------------------------------------------------------------
E            Ptaci mokrady     0            15           100
```

4.1.1. Upozornění

Nejnižší patro musí být vždy 0. Soubor se vstupními daty musí být textový (s koncovkou .txt).

4.2. Nastavení parametrů

Můžete si nastavit parametry výpočtu pro zkoumání chování programu.

4.2.1. Otevírací doba

Udává kdy zoologická zahrada otevře a kdy zavře. Otevírací doba může být i přes noc, ale nemůže být otevřeno déle než jeden den. Časy musí být zapsány ve tvaru, který zobrazují běžné digitální hodiny (hh:mm). Nesmí chybět dvojtečka oddělující hodiny a minuty.

4.2.2. Typ návštěvníků

Je na výběr z několika typů návštěvníků, které se liší výběrem stanoviště a občerstvení, na které půjdou jako další:

- **0** – Vybere si to stanoviště, které je další v jeho seznamu.
- **1** – Nejdříve projde stanoviště z jeho seznamu, která jsou v patře, ve kterém se nachází. Potom si vybere patro, které je další v seznamu.
- **2** – Vybere si první stanoviště z jeho seznamu, které má nejmenší frontu.
- **3** – Vybírá si patro, ve kterém projde všechna svá stanoviště ze seznamu, s nejmenším počtem návštěvníků. Občerstvení si vybere to, které je v jeho patře a má nejmenší frontu. (Parametr ‚Počet občerstvení' pro něj nehraje žádnou roli)

Můžete si vybrat, že každý návštěvník bude stejného typu nebo se využijí všechny typy. Všech návštěvníků s jedním typem bude přibližně stejný počet.

4.2.3. Počet návštěvníků

Pro nastavení simulace pro jeden počet návštěvníků nastavte políčka Min a Max na stejné hodnoty. Pro nastavení simulace pro více počtů návštěvníků nastavte nejmenší a největší počet a po jak velkých krocích se má spustit simulace.

4.2.4. Počet stanovišť

Nejmenší a největší počet stanovišť, která návštěvník navštíví.

4.2.5. Počet občerstvení

Nejmenší a největší počet stánků s občerstvením, které bude moct navštívit, když dostane hlad. Určuje, jak budou návštěvníci vybíraví ohledně jídla.

4.2.6. Příchod

Nejpozdější čas příchodu návštěvníka do zoologické zahrady. Musí být také ve tvaru jako otevírací doba.

4.2.7. Trpělivost

Nejkratší a nejdelší čas, který návštěvník vydrží čekat ve frontě (v minutách). Pokud návštěvníkovi dojde trpělivost, tak odejde z fronty a půjde na jiné stanoviště nebo občerstvení. Když návštěvníkovi zbývá poslední stanoviště k navštívení, tak trpělivost neplatí a zůstane ve frontě.

4.2.8. Hlad

Nejkratší a nejdelší čas, který návštěvník vydrží bez jídla (v minutách). Po uplynutí této doby se půjde návštěvník najíst ke stánku s občerstvením. Pokud má návštěvník poslední stanoviště k navštívení, tak nebude odcházet na občerstvení a dokončí poslední stanoviště.

4.2.9. Rychlost jezení

Nejkratší a nejdelší čas, kterým návštěvník stráví jezením (v minutách). Po najezení jde na další stanoviště.

4.2.10. Tlačítko Vybrat soubor se vstupními daty

Po stisknutí se zobrazí průzkumník souborů. Zde je potřeba vybrat textový soubor, kde jsou vypsány stanoviště zoologické zahrady ve správném formátu. Pokud si soubor nevyberete, tak program nemůže spustit simulaci.

4.2.11. RNG seed

Určuje ‚semínko' náhodného generátoru čísel, který náhodně vybírá hodnoty ze zadaných rozsahů parametrů pro vytvoření návštěvníků. Pokud toto semínko necháte stejné, tak se budou generovat stejná náhodná čísla a tím pádem návštěvníci se stejnými vlastnostmi.

4.2.12. Tlačítko START

Smaže log a spustí simulaci podle zadaných parametrů. Po spuštění se objeví nápis ‚POČÍTÁ'.

4.3. Záznam (log)

Při zaškrtnutí políčka **Log** se při výpočtu budou vypisovat zprávy o tom, co se v simulaci děje. Pro filtrování těchto zpráv se dá použít políčko Filtr. Pokud něco napíšete do Filtru, tak se budou zobrazovat jen zprávy, které obsahují to, co jste sem napsali. Tlačítkem ‚Vymazat log' se smaže všechen text ze záznamu.

4.4. Výstup

Ve výstupu se objeví vypočítaná data ze simulace. Po výpočtu se do výstupu vypíše něco takového:

```
4 z 5 stihli vše. 1 mělo moc velký plán.
Průměrný čas návštěvníka v zoo 03:35
```

Počet návštěvníků, kteří měli moc velký plán jsou ti, u kterých jejich plán nebylo možné stihnout i kdyby nebyly v zoo žádné fronty.

Při zaškrtnutí kolonky ‚Čistě data' se nebude výstup vypisovat ve větách, ale jen daty oddělenými čárkou v pořadí:

1. počet návštěvníků, kteří stihli navštívit všechna stanoviště
2. celkový počet návštěvníků
3. počet návštěvníků, kteří měli nesplnitelný plán
4. průměrný čas strávený návštěvníkem v zoologické zahradě v minutách

Tlačítko ‚Vymazat výstup' po stisknutí smaže všechen text ve výstupu.

5. Průběh prací

Na začátku jsem si vytvořil něco jako myšlenkovou mapu nápadů, tříd a metod, které jsem věděl, že budu potřebovat a promyslel rozložení, propojení tříd a strom dědičnosti, abych si vytvořil takovou teoretickou kostru programu. V kostře toho bylo spoustu nedomyšleno, ale to jsem zjistil až při implementaci. Ale i tak to byl dobrý základ, o který se dalo opřít.

Dále jsem si sepsal, co bude muset být v grafickém rozhraní, co bude moct uživatel měnit a co bude vyžadováno na vstupu. Také jsem si nakreslil jednoduchý náčrt. Ale i tak se vzhled na konci lehce změnil a je možné měnit více parametrů, než jsem měl původně v plánu.

Myšlenka diskrétní simulace mi přijde jasná a vcelku jednoduchá. Prostě každý proces dělá to, co umí ve správném pořadí tak, jak je to naplánováno v kalendáři. Tím pádem mě překvapilo, že pro mě bylo vcelku složité a časově náročné to celé zrealizovat. Hlavně událost Hlad byla problematická v tom, že ruší plány a může nastat skoro kdykoliv. Proto je potřeba vyřešit každý případ stavu návštěvníka, ve kterém může dostat hlad.

V plánu jsem měl ještě přidat speciální události expozic, ale ty by zase mohli nastat kdykoliv a rušili by už naplánované události. Hlavně hledání určité události je asymptoticky lineární k počtu událostí, protože nemáme přímý odkaz k události, takže to značně zpomaluje chod programu. Také bych to už časově nestihl. Možná je ale lepší přístup, jak to naimplementovat, který neznám.

6. Závěr

Vyvíjení tohoto projektu pro mě bylo velice přínosné z hlediska navrhování programu, dekompozice a implementace takto silně objektově orientovaného programu. Vyzkoušel jsem si dědičnost v plné síle a teď se toho už tolik nebudu bát ji využívat častěji. Je to můj největší program, který jsem kdy zatím vytvořil. Má to své nedokonalosti, ale posunulo mě to o kus dál v mých programátorských schopnostech a za to jsem rád.