Zadání semestrální práce KIV/UPG 2021/2022

Marek Hravý je geek do počítačů a také do všeho co se týče vesmíru. Má v plánu nastoupit na informatiku na FAV a nejen svoje znalosti standardní galaktické abecedy tam uplatnit. Asi to nepůjde úplně cestou hraní počítačových her typu Space Invaders či Star Trek Online, ale mohl by si kvůli pandemii něco bokem z volné chvíle naimplementit. Rozhodl se tedy že udělá nějakou poggers vizualizaci vesmíru s planetami. Sice si teď myslí že to udělá za hodinu, ale IRL jeho zkušenosti jsou, řekněme si to otevřeně, velmi omezené. Bude to pravděpodobně bastlit jak může (asi bude lepit kód dohromady ze Stack Overflow) a použije na to Brainfuck, protože tento programovací jazyk se používal na SPŠE Dolní Lhota a jiný programovací jazyk zatím nezná. Pojďme mu ukázat, jak by na to měl jít a udělat nějakou základní vizualizaci vesmíru.

Cílem semestrální práce je naimplementovat vizualizaci simulace vesmíru. Práce je rozložena do tří částí, přičemž první dvě jsou povinné (a jejich řešení musí být odevzdáno), třetí je pak volitelná.

V první části bude očekávána dynamická vizualizace vesmíru, ve druhé se rozšíří např. o zobrazení grafů. V rámci volitelných rozšíření bude možno program dále vylepšovat (exportem vizualizace do rozličných formátů, přidáním různých prvků do vizualizace apod.) a také bude možné navrhnout vlastní rozšíření, které bude předem cvičícím schváleno (cvičící navrhne bodové hodnocení). K implementaci je možno užít libovolného programovacího jazyka, ovšem s knihovnami rozsahově nejvýše stejnými standardním knihovnám jazyka Java tak, aby obtížnost úlohy zůstala zachována.

V každé části musejí být splněny základní funkční požadavky, jinak je práce navrácena k opravě bez hodnocení. Každý neúspěšný pokus o odevzdání je penalizován dle tabulky uvedené na Courseware.

Vstupní data

Vstupními daty této semestrální práce jsou CSV (comma-separated values) soubory. Oddělovačem je standardní čárka (`,`), používá se desetinná tečka. Kódování souborů je UTF-8. Výjimkou je první řádek souboru, kde se nachází gravitační konstanta G (určující, jak moc se vesmírné objekty vzájemně přitahují) a také časový krok, který má uběhnout za jednu sekundu reálného času. Sloupce na dalších řádkách jsou v pořadí:

```
Nazev,Typ,Pozice X,Pozice Y,Rychlost X,Rychlost Y,Hmotnost
```

Název je libovolný řetězec, názvy jsou unikátní. Typ je řetězec, možnosti jsou: `Planet`, `Comet`, `Rocket` a případně další Vámi definované typy. Pozice a rychlost jsou rozloženy na X a Y složky. Rychlostí je myšlenou, jakou má objekt rychlost na počátku simulace v době $t = 0$. Všechny veličiny jsou v základních SI jednotkách.

Příklad vstupního souboru:

```
1,0.1
First,Planet,10,10,0,0,1.0
Second,Planet,20,20,0,0,5.0
Third,Planet,30,5,0,0,8.0
```

Část 1: Animace (až 15 bodů)

Základní funkční požadavky (10 bodů): Po spuštění programu pomocí příkazu `Run.cmd` resp. `./run.sh` s parametrem cesty k datovému souboru v textové podobě (jsou poskytnuty k zadání) se spustí simulace pohybu vesmírných objektů (dle parametrů v datovém souboru). Simulace bude probíhat tak, že 1 sekunda ve skutečnosti odpovídá času uvedenému ve vstupním souboru. Simulaci možno pozastavit a opětovně rozběhnout stiskem mezerníku. Způsob simulace je popsán na poslední straně tohoto zadání.

Vesmírné objekty budou pravidelně zobrazovány v okně (po spuštění se zobrazí vizualizace v simulačním čase $t = 0$). Všechny objekty budou v každém čase **zcela** viditelné (s výjimkou vzájemného překryvu objektů) a zároveň tato oblast **maximálně vyplní** dostupný prostor okna s podmínkou zachování poměru stran vizualizace. Střed vesmíru bude ve středu okna. Okno po startu bude mít minimální velikost 800px na 600px. V pravém horním rohu okna aplikace bude vypsán aktuální simulační čas.

Další požadavky: Poloměr objektů bude variabilní a určíte jej na základě hmotností uvedené v datech; předpokládejte shodnou a „jednotkovou“ hustotu všech objektů.

Velikost okna bude moci uživatel po spuštění změnit libovolně. Vizualizace bude na tuto změnu reagovat a přizpůsobovat se jí.

Po kliknutí myši na libovolný objekt se zobrazí základní informace o objektu (název, aktuální pozice a rychlost). Tato interakce bude doprovázena vizuální změnou daného objektu. Přiložena bude dokumentace ve formátu `*.pdf` se strukturou obdobnou vzorové dokumentaci, která je k dispozici na Courseware a s odpovídající informativní hodnotou.

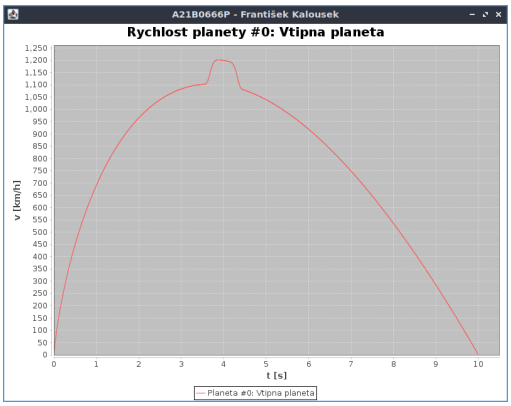
Část 2: Grafy a další rozšíření (až 10 bodů)

Základní funkční požadavky (5 bodů): Program musí splňovat veškeré (nejen základní) požadavky z Části 1. Simulace bude rozšířena o ošetření případu, že dva objekty kolidují. v takovém případě se spojí v jeden (dle zákonů zachování hmoty a hybnosti a budou se dále jevit jako jeden objekt. Připomínáme, že předpokládáme stejnou „jednotkovou“ hustotu.

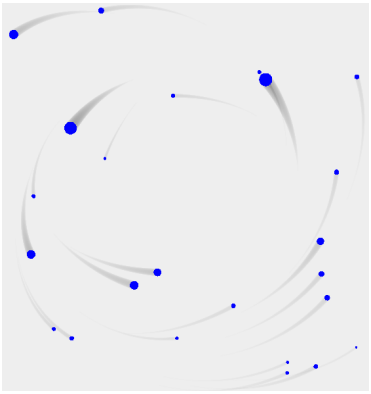
Po kliknutí na objekt bude zobrazen graf rychlosti (za posledních 30 simulačních sekund). Rychlost v grafech bude převedena na km/h. Grafy můžete vykreslit externí knihovnou (v Javě např. JFreeChart).

Přiložená dokumentace ve formátu `*.pdf` bude aktualizována a bude popisovat i problematiku z této části samostatné práce.

Další požadavky: Budou zobrazeny trajektorie pohybu objektů, tyto trajektorie budou postupně mizet, a kromě začátku simulace budou zobrazovat pohyb provedený právě jednu sekundu právě proběhlé simulace.



Obrázek 1: Příklad grafu rychlosti planety 0 ve vesmíru.



Obrázek 2: Příklad jednoduché animace planet s trajektoriemi.

Část 3: Volitelná rozšíření (až 15 bodů)

Základní funkční požadavky (0 bodů): Program musí splňovat veškeré (nejen základní) funkční požadavky z Části 2. Součástí odevzdání je aktualizovaná dokumentace obsahující přehled realizovaných volitelných rozšíření. Rozšíření neuvedená v dokumentaci nebudou hodnocena.

Další funkční požadavky:

Export do PNG (až 2 body): Animaci bude možné v libovolném okamžiku zastavit a stav exportovat do rastrového obrázku PNG.

Export do SVG (až 2 body): Animaci bude možné v libovolném okamžiku zastavit a stav exportovat do vektorového obrázku SVG. Zde je možné užít knihovny pro export (v Javě např. JFreeSVG)

Raketa (až 5 bodů): V simulaci bude raketa, která bude ovládána pomocí WASD. Vztažnou soustavou nebude okno, ale raketa (W pohne raketou dopředu, S dozadu, A otočí raketu vlevo a D vpravo). Raketa bude moci přistát na libovolné planetě přistát, čímž hráč vítězí. Pokud však při bude velká přistávací rychlost, raketa vybuchne. O tom uživatel bude informován animací výbuchu. Raketa bude umístěna libovolně do prostoru, avšak ne příliš blízko k nějakému z objektů (nesmí dojít k okamžitému konci hry).

Vtipná data (až 5 bodů): Vymyslete vlastní datovou sadu tak, aby pohyb prvního objektu tvořil nějaký komplexní tvar a zároveň byl periodický. Tvar trajektorie musí v jedné periodě alespoň 2x měnit konkavitu.

Alternativně Vymyslete vlastní datovou sadu tak, aby graf rychlosti prvního objektu byl periodický a měl nějaký "zajímavý tvar", například sinusoidu. Tvar bude v jedné periodě obsahovat alespoň tři lokální extrémy.

Gravitační pole (až 4 body): Umožněte zakreslení gravitačního pole pomocí šipek ve vizualizaci. Toto rozšíření bude možné vypnout a zapnout kdykoliv během simulace. V každém okamžiku bude vykresleno alespoň 100 šipek.

Couvání (až 4 body): Během simulace bude možno obrátit směr pohybu času a navrátit se i za počátek simulace. Je potřeba si pamatovat, které objekty se v jaký moment sloučily.

Zrychlení/zpomalení simulace (až 2 body): Simulaci bude možné 2x urychlit / zpomalit a vrátit zpět do původní rychlosti pomocí ovládacích prvků GUI.

Zvětšení a posun (až 4 body): Přidejte možnost intuitivního přiblížení, oddálení a pohybu prostorem. Zobrazení bude možné také vrátit do původního stavu tak, aby bylo možno vše opět vidět.

Piktogramy (až 4 body): Ve vizualizaci se nevykreslí objekty pouze jednolité, ale budou vykresleny odpovídajícím obrázkem (planety, raketa apod.). Velikost obrázků bude reagovat na poměr přiblížení (viz rovněž zvětšení a posun).

Optimalizace (až 3 body): Tři práce, které budou mít simulaci provedenou nejrychleji a bude splňovat požadované podmínky budou oceněny 3, 2 a 1 bodem. Na Vás je tedy simulaci co nejvíce urychlit.

Lepší grafy (až 3 body): Umožněte, aby se v grafech zobrazovala data od počátku simulace. Datovou řadu adekvátně redukujte tak, aby nebylo zbytečně zarhnuto mnoho bodů.

Vlastní rozšíření (až ? bodů): Po předchozí domluvě se cvičícím je možné si vymyslet vlastní rozšíření. Počet bodů vyplyne z domluvy.

N-objektů

Pohyb těles, jejich vzájemná gravitační síla není zanedbatelná, lze simulovat pomocí tzv. N-objektů. Základem výpočtu je Newtonův gravitační zákon. V simulaci N-objektů je nejprve potřeba určit zrychlení všech objeků pomocí vzorce:

$$\mathbf{a}_i = G \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} m_j \frac{\mathbf{p}_j - \mathbf{p}_i}{|\mathbf{p}_j - \mathbf{p}_i|^3} \tag{1}$$

kde $\mathbf{a}_i$ je zrychlení objektu s indexem i, $\mathbf{p}_i$ je jeho pozice, m_j jeho hmotnost a G je gravitační konstanta. V každém kroku se přepočítá rychlost objektu $\mathbf{v}_i$ a pozice pomocí výpočtů:

$$\begin{aligned} \mathbf{v}_i &+= \frac{\Delta t}{2} \mathbf{a}_i \\ \mathbf{p}_i &+= \Delta t \mathbf{v}_i \\ \mathbf{v}_i &+= \frac{\Delta t}{2} \mathbf{a}_i \end{aligned} \tag{2}$$

Nejprve se tedy upraví rychlost objektu jako by uběhla polovina časového kroku Δt. V polovině kroku se určí samotná pozice objektu a nakonec se upraví rychlost tak, jak by měla být na konci časového kroku. Určením pozic „uprostřed“ časového kroku získáme přesnější výpočet.

Simulaci lze zapsat následujícím pseudokódem:

```
1:  procedure UpdateSystem(t)        ▷ t je simulační čas, který uplynul od
                                       poslední aktualizace systému
2:      dt_min = 0.02                ▷ elementární časový krok simulace
3:      while t > 0 do
4:          dt = min(t, dt_min)

5:          for all objects i do     ▷ Spočti zrychlení objektů podle vzorce 1
6:              a_i = computeAcceleration(i, dt)
7:          end for

8:          for all objects i do     ▷ Spočti nové rychlosti a pozice objektů
                                       podle vzorce 2
9:              v_i = v_i + 0.5 * dt * a_i
10:             p_i = p_i + dt * v_i
11:             v_i = v_i + 0.5 * dt * a_i
12:         end for

13:         t = t - dt
14:     end while
15: end procedure
```