# Úvod

Jako zadání semestrální práce mám naprogramovat vizualizaci simulace vesmíru s planetami. Práce je rozdělena na tři části, které zdaleka nejsou hotovy. Postupně budu části plnit a také doplňovat tuto dokumentaci. Toto není finální verze mé práce.

# Práce v první části

Na první pohled program obsahuje více souborů. Snažil jsem se od začátku dělit program tak, aby v každé třídě byly ty části kódu, které spolu souvisí. Nemělo by se tedy stát, že by se v třídě, která má na starosti vykreslovat plátno, počítaly fyzikální zákony pro pohyb planet. V této dokumentaci chci popsat, jak jednotlivé soubory pracují a tvoří dohromady simulaci vesmíru.

## DoubleVector2D.java

Jelikož budu pracovat s hmotnými body, udělal jsem si třídu, která mi uloží pozici x a y bodu. Třída je jednoduchá přepravka s getry (bez setrů) a metodou toString(). Body v přepravce jsem neměl potřebu měnit a když jsem chtěl změnit pozici planety, jednoduše jsem si vytvořil novou instanci DoubleVector2D. Specialita co tato třída má, je metoda computeDistance(). Ta vypočítá vzdálenost od jiného bodu třídy DoubleVector2D.

## Planeta.java

Název je poněkud zavádějící, nejedná se nutně o planetu ale o vesmírné těleso. Možná později z této třídy udělám abstraktní třídu a vytvořím jí potomky Planeta, Kometa a Raketa. Zatím v tom nevidím smysl a budu všechno nazývat planetou.

Každá planeta je charakteristická svým názvem, typem (Planeta, Raketa, Kometa), pozicí středu, rychlostí a hmotností. Ve třídě jsou getry, setry a toString(). Dále bude potřeba atribut zrychlení a poloměru. Poloměr se vypočítá z hmotnosti a hustoty, kterou mají všechny planety 1kg/m3. Zrychlení bude nastavovat simulace. Jelikož pozice x, y představuje pozici středu planety, udělal jsem si speciální getry, které vrací střed změněný o poloměr planety. To mi usnadňuje práci při kreslení, jelikož elipsy se kreslí ne od středu, ale od levého horního rohu.

## VstupDat.java

Na vytvoření této třídy jsem použil znalosti z předmětu OOP. Udělal jsem ze třídy jedináčka, který umí číst soubor. Předpokládám, že data v souboru mají strukturu stejnou jako popisuje zadání, tj. na první řádce dvě čísla a poté n řádků s daty planet, vše odděleno čárkou. Jinou strukturu by třída nepřečetla. Metoda nactiData() nejprve hledá dvě čísla, gravitační konstantu G a časový skok. Až poté načítá data planet. Konstanty a planety si k sobě uloží do atributů, aby k datům měli přístup ostatní třídy.

## Galaxy\_SP2022.java

Hlavní třída programu. Načte data, spustí simulaci, vizualizaci a časovač. Zároveň se stará o vstup z klávesnice (pauza mezerníkem) a obsluhuje kliknutí myši na planetu.

## SimulationTimer.java

Časovač celého vesmíru. Předává simulaci čas a překresluje plátno. Důležitým parametrem pro simulaci je casovySkok ze vstupního souboru. Ten říká, kolik sekund simulace uběhne za jednu sekundu reálného času. Třída také umí simulaci pozastavit a poté na ni správně navázat. Časovač zařizuje, že se každých 20 ms počítají nové pozice planet a zároveň se vykreslí.

## Simulace.java

Třída pracuje s konstantou G a časovým skokem ze souboru. Simuluje pohyb planet pomocí N-objektů. Metoda updateAllAcceleration() nastaví všem planetám najednou zrychlení podle vzorce. Samotný pohyb planet zařizuje metoda updateSystem(). Její výpočet je náročný a provádí se často. Pokud je mnoho planet, počítač nemusí stihnout dokončit počítání do nového volání metody a tím zpomalí celou simulaci. Z tohoto důvodu jsem se rozhodl, že přesnost simulace se bude odvíjet od počtu planet. Čím méně bude planet, tím více se provede výpočtů a tím přesnější simulace bude. Naopak čím více bude planet, tím se provede méně výpočtů a simulace bude nepřesná.

## Vizualizace.java

Správce plátna, vykresluje vše co vidíme. Aby byla co nejvíce využita daná plocha, při každém volání repaint() se musí vypočítat okraje vesmíru. Všechny planety tak mohou být vidět najednou a maximálně využijí plochu okna. Okraj vesmíru je vždy ta nejmenší nebo největší pozice hrany elipsy. Když jsem měl výpočet vesmíru hotový, musel jsem ho ještě aplikovat podle tohoto postupu:

1. Vycentrovat vizualizaci na střed okna
2. Aplikovat scale na osu x a y
3. Vycentrovat vizualizaci na střed vesmíru
4. Odebrat minima vesmíru v osách x, y

Po této aplikaci se všechny nakreslené tvary budou řídit těmito pravidly. Zezačátku jsem pracoval s obdélníkem, abych viděl, jak škálování přesně funguje. Nakonec jsem ho tam nechal, protože proč ne. Po obdélníku jsem nakreslil planety a rozhýbal simulaci. Jelikož chci ještě na obrazovku vypisovat simulační čas a informace o vybrané planetě, uložil jsem si transformaci původního okna. Aby byl text na popředí, zavolám jeho vykreslení jako úplně poslední.

Dalším úkolem vizualizace je vyhodnocení HitTestu myši. Vyhodnocení provádí metoda getHitPlanet(). K tomu si uložím výše vypsaná pravidla a udělám jejich inverzi. Když kliknu myší do okna, souřadnice kliku se převede na skutečnou velikost planet. V této soustavě vytvářím nové planety a zjišťuji, jestli se v nějaké náhodou nenachází souřadnice kliknutí. S touto metodou jsou ale problémy. Zjistil jsem, že pokud je operační systém uživatele nastavený tak, aby zvětšoval velikost okna, transformace se posouvají a HitTest je prakticky nepoužitelný. Zatím mě nenapadá jiný způsob, jak bych mohl HitTest vyřešit. Pokud jsme klikli na planetu, zobrazí se nám její informace a planeta se překreslí šedým kruhem. Zatím nastavuji atribut selectedPlanet na instanci kliknuté planety a na černý kruh kreslím další šedý, ale tuto metodu chci předělat, aby se planeta vykreslila jinou barvou.

# Vyhodnocení první části

Na práci jsem využil znalosti nejen ze cvičení, ale také z ostatních předmětů, za což jsem rád. Rozhodně vidím místa, kde bych mohl program vylepšit. Ať už optimalizovat výkon, tak vylepšit čitelnost kódu. Přesto jsem s prací v první části spokojen a nezbývá mi než doufat, že ji nebudu muset předělávat.