Členění tříd

Hlavní třídy

Do hlavních tříd patří třída s metodou main() Galaxy_SP2022 a třída Space, která reprezentuje vesmír jako celek.

Třídy vesmírných objektů

Obsahují abstraktní třídu SpaceObject, od které dědí specifické vesmírné objekty. Ty jsou také v této kategorii zahrnuty (zatím hotová pouze třída Planet).

Grafické třídy

Do grafických tříd patří třída DrawingPanel, na kterou je vykreslovaná veškerá grafika, třída WindowInitializer, která má na starosti inicializaci okna a třída ColorPicker, jež poskytuje barvy pro objekty.

Utility třídy

Obsahují knihovní třídu FileLoader, která slouží pro načtení obsahu souboru a třídu Vectors, jež poskytuje statické metody pro práci s vektory.

Inicializace

Hlavní třída Galaxy_SP2022

Tato třída pomocí třídy FileLoader nejdříve načte první řádku souboru (obsahuje gravitační konstantu a časový krok) a poté všechny ostatní řádky s vesmírnými objekty.

Pomocí získaných hodnot je vytvořena instance třídy Space a je předána metodě init() třídy WindowInitializer.

Třída WindowInitializer

Třída pomocí metody init() vytvoří instanci JFrame, která představuje okno, a vloží do ní instanci DrawingPanel, což je panel pro vykreslování grafiky.

Instance třídy Timer je zde využívána pro překreslení panelu (každých 17 ms – 60 snímků za sekundu). Panel má také nastavený MouseListener, kde překrytá metoda mousePressed() kontroluje, zda uživatel kliknul na některý z vykreslovaných objektů a pokud ano, zobrazí o něm informace. Další posluchač pro panel je MouseWheelListener, který při scrollování modifikuje přiblížení vesmíru.

Pro celé okno je nastaven KeyEventDispatcher (není použit KeyListener, aby nebyl ztrácen focus) – překrytá metoda dispatchKeyEvent() reaguje na stisknutí mezerníku a kláves W, S, A, D / šipek. Při stisknutí mezerníku se mění status simulace na pozastavená / aktivní. Klávesami W, S, A, D a šipkami lze posouvat vesmír.

Dole v okně se nacházejí tlačítka pro export do PNG a SVG, zrychlení / zpomalení simulace a reset časového kroku, a vycentrování vesmíru zpět do původní pozice.

Třída Space

Třída reprezentuje celý vesmír, obsahuje tedy důležité konstanty G_CONST (gravitační konstanta) a T_CONST (časový krok simulace – kolik sekund v simulaci odpovídá naší jedné sekundě). Také obsahuje kolekci typovanou na SpaceObject, ve které se nachází všechny vykreslované objekty.

Abstraktní Třída SpaceObject a její potomci

SpaceObject

Abstraktní třída SpaceObject definuje předpřipravený vesmírný objekt. Byla zvolena abstraktní třída, protože některé metody nebudou společné pro všechny typy objektů (např. planeta se bude vykreslovat jako jiný tvar než kometa).

Třída pro všechny vesmírné objekty společně definuje:

- název,
- typ,
- pozici X a Y,
- pozici X a Y přizpůsobenou oknu,
- celkovou rychlost, složky X a Y rychlosti,
- celkové zrychlení, složky X a Y zrychlení,
- váhu,
- poloměr,
- přizpůsobený poloměr oknu,
- barvu,
- kolekci neškálovaných trajektorií X a Y,
- kolekci škálovaných trajektorií X a Y,
- kolekci nasbíraných hodnot rychlostí pro graf.

Statické proměnné MIN_RADIUS a MAX_RADIUS představují určité omezení poloměru tak, aby nedosahoval příliš malých rozměrů (planeta by nebyla vůbec viditelná), nebo příliš velkých rozměrů (stejně by byl poloměr pomocí škálování zmenšen). Práce s těmito hodnotami je vidět v setteru pro poloměr přizpůsobený oknu setScaledRadius(). Barva je přiřazena jednou v konstruktoru metodou randomColor() třídy ColorPicker.

Abstraktní metody, které lze implementovat až v konkrétním potomkovi, jsou:

- draw() – vykreslí objekt na jeho souřadnicích přizpůsobených oknu,
- drawHighlight() – vykreslí zvýraznění objektu po kliknutí na objekt uživatelem,
- approximateHitTest() – provede přibližný hit-test pro objekt při kliknutí uživatelem,
- findRadius() – vypočítá poloměr objektu podle jeho hmotnosti (hustota je jednotková).

Všechny vesmírné objekty mají společné metody:

- computeAcceleration() – pomocí Newtonovy pohybové rovnice vypočítá zrychlení objektu, které je ovlivněné všemi ostatními objekty,
- checkForCollision() – zkontroluje, zda objekt nekoliduje s některým z ostatních a pokud ano, provede kolizi a přepočítá vlastnosti nového objektu.

Planet

Tento potomek třídy SpaceObject je reprezentován tvarem kruhu. Obsahuje implementované abstraktní metody ze třídy SpaceObject:

- draw() – vykreslí kruh na souřadnicích planety přizpůsobených oknu,
- drawHighlight() – vykreslí barevný obrys kruhu reprezentujícího planetu po kliknutí na planetu uživatelem,
- approximateHitTest() – provede přibližný hit-test pro planetu při kliknutí uživatelem – je vytvořen nový kruh o něco větší než kruh reprezentující planetu a pro tento větší je proveden hit-test, tolerance je tím menší, čím je planeta větší,
- findRadius() – vypočítá poloměr planety podle její hmotnosti pomocí vzorce pro hustotu a objem koule, kdy hustota je jednotková:

$$\rho = \frac{m}{V}$$
$$V = m$$
$$\frac{4}{3}\pi r^3 = m$$
$$r = \sqrt[3]{\frac{3m}{4\pi}}$$

Grafika

Třída DrawingPanel

Tato třída představuje plátno, a tedy slouží k vykreslování objektů a informací o nich. Obsahuje metodu paint(), ve které se periodicky spouští hlavní metody programu. Třída definuje mnoho důležitých proměnných, které ovládají simulaci:

- Space space – instance celého vesmíru,
- long startTime – čas vytvoření instance DrawingPanel v nanosekundách,
- double simulationTimeS– čas simulace v sekundách,
- double actualTimeS – opravdový čas v sekundách,
- final int UPDATE_TIME – jak často se má aktualizovat simulace v ms – nastaveno na 17 ms, což je 60 snímků za sekundu,
- boolean simulationActive – simulace aktivní / pozastavená,
- double UPDATE_CONST – kolikrát se v jedné aktualizaci přepočítají hodnoty zrychlení, rychlosti, pozice a překreslí se planety,
- final double UPDATE_CONST_ORIGINAL – původní hodnota update konstanty,
- List<SpaceObject> spaceObjects – kolekce všech vesmírných objektů,
- double GConstant – gravitační konstanta,
- double TStep – časový krok simulace,
- final double T_STEP_ORIGINAL – původní hodnota proměnné TStep,
- double x_min, x_max, y_min, y_max – extrémy pozic objektů,
- double world_width, world_height – rozměry simulačního světa,
- double scale – číslo, kterým násobíme souřadnice, aby se objekty vešly do okna,
- SpaceObject currentToggled – uživatelem právě vybraná planeta,
- boolean showingInfo – říká, jestli jsou právě v pravém horním rohu vypisovány informace o planetě,
- boolean collisionOn – kolize zapnutá / vypnutá,
- int trajectoryLength – velikost kolekcí pro trajektorie – nastaveno na 60 (trajektorie program ukládá každých 17 ms, tedy dohromady je uložena 1 sekunda),
- ChartWindow chartWindow – okno s grafem,
- int MAX_CHART_VALUES – kolik maximálně bodů může mít graf,
- double zoom – velikost přiblížení,
- double up – velikost posunutí nahoru / dolů,
- double addUp – číslo, které se má postupně přičíst k proměnné up,
- double right – velikost posunutí doprava / doleva,
- double addRight – číslo, které se má postupně přičíst k proměnné right.

Přizpůsobování se velikosti okna – škálování

Škálování je realizováno v metodě updateDrawing(), která nejdříve spočte hodnotu scale podle extrémních souřadnic objektů, poté touto hodnotou vynásobí souřadnice a rozměry objektů a všechny objekty vykreslí. S tímto přístupem je zároveň souřadný systém vždy vycentrován na střed okna.

Hodnota scale je vypočítána v osách X i Y a jako finální scale je vybrána ta menší z nich. Extrémní souřadnice objektů jsou hledány pomocí pomocných metod findXMinSpaceObject(), findXMaxSpaceObject(), findYMinSpaceObject(), findYMaxSpaceObject().

V této metodě je také vypočítán maximální rozměr vesmírného objektu jako polovina rozměru okna.

```
procedure updateDrawing()
	world_width = abs(x_max – x_min)
	world_height = abs(y_max – y_min)

	scale_x = window_width / world_width
	scale_y = window_height / world_height
	scale = min(scale_x, scale_y)
	for all objects do
		x = (object.x – x_min)*scale + window_width/2 – (world_width*scale)/2
		y = (object.y – y_min)*scale + window_width/2 – (world_height*scale)/2
		radius = object.radius*scale

		object.radius = radius
		object.x = x
		object.y = y

		draw(object)
	end for
end procedure
```

Poznámka: V kódu se ještě od proměnných world_width a world_height odečítá 4x konstanta minimálního poloměru objektu, což zajistí, že se v okně budou zobrazovat celé i nejmenší objekty.

Třída ColorPicker

Knihovní třída, která slouží pro uchovávání pole vhodných barev pro objekty. Barvu pro planetu získáme voláním metody randomColor(), jež vrátí náhodnou barvu z pole.

Vykreslení objektu

Realizováno ve třídě specifického objektu metodou draw(), která je definovaná v abstraktní třídě SpaceObject a implementována až ve třídě potomka. Nejdříve je vhodně nastaven atribut shape na souřadnice a rozměry objektu přizpůsobené oknu a poté je vykreslen určený tvar objektu.

Vykreslení zvýraznění objektu

Realizováno ve třídě specifického objektu metodou drawHighlight(), která je definovaná v abstraktní třídě SpaceObject a implementována až ve třídě potomka. Nejdříve je vhodně nastaven atribut shape na souřadnice a rozměry objektu přizpůsobené oknu a poté je vykreslen určený obrys objektu.

Vypsání informací o objektu

Pokud je proměnná showingInfo třídy DrawingPanel nastavena na true (mění se podle toho, jak uživatel kliká na objekty / mimo ně), zavolá se v metodě paint() metoda drawInfo(). Tato metoda zjistí, zda je uživatelem vybraný objekt opravdu stále v kolekci spaceObjects (mohl být odstraněn kolizí) a pokud ano, zavolá metodu

drawHighlight() pro zvýraznění objektu. Poté uloží vybrané informace (název, pozice X a Y, rychlost, zrychlení, rozměr, váha) do proměnných a ve vhodné podobě (zaokrouhlení, zápis pomocí mocnin deseti) je zobrazí na plátno metodou drawString().

Vypsání aktuálního času

Metoda paint() při každém svém průběhu volá nejdříve metodu computeTime(), jež přepočítá čas simulace a opravdový čas. Proměnné simulationTimeS a actualTimeS se ale změní jen pokud je simulace aktivní (tedy ubíhá v ní čas).

Status simulace se mění při volání metody changeSimulationStatus(), jež se volá tehdy, když uživatel zmáčkl mezerník. Tato metoda změní proměnnou simulationActive na opačnou hodnotu.

Po volání metody computeTime() zavolá metoda paint() metodu drawTime(). Zde jsou proměnné simulationTimeS a actualTimeS převedeny na vhodný řetězec a vykresleny metodou drawString().

Hlavní smyčka

Metoda paint()

Metoda paint() se díky Timeru ve třídě WindowInitializer volá každých 17 ms a obstarává volání metod hlavní smyčky:

- updateSystem() – aktualizace pozic, zrychlení, rychlostí a překreslení panelu – každých 17 ms (obstaráno dělením modulo),
- computeTime() – vypočítání aktuálního času,
- drawTime() – vykreslení aktuálního času,
- drawInfo() – vykreslení informací, pokud je uživatelem vybraný nějaký objekt.

Metoda updateSystem()

Metoda zjistí, zda je simulace aktivní a pokud ano, přepočítá nové zrychlení, rychlosti a pozice. Nakonec zavolá metodu updateDrawing() – ta je volána i pokud simulace není aktivní. Čas t je na začátku nutné převést na sekundy a poté na čas simulace násobením proměnnou TStep (kolik našich sekund odpovídá kolika sekundám v simulaci).

```
procedure updateSystem(t)
	t = t/1000 * TStep
	if (simulationActive)
		for(i = 0; i < UPDATE_CONST; i++)
			for all objects do
				object.computeAcceleration()
			end for
			for all objects do
				object.speedX += 0.5(t/UPDATE_CONST) * object.accelerationX
				object.speedY += 0.5(t/UPDATE_CONST) * object.accelerationY

				object.x += 0.5(t/UPDATE_CONST) * object.speedX
				object.y += 0.5(t/UPDATE_CONST) * object.speedY

				object.speedX += 0.5(t/UPDATE_CONST) * object.accelerationX
```

```
                        object.speedY += 0.5(t/UPDATE_CONST) * object.accelerationY
                end for
                if (collisionOn)
                        for all objects do
                                object.checkForCollision()
                        end for
        end for
    updateDrawing()
end procedure
```

Metoda computeAcceleration()

Tato metoda je volána pro všechny vesmírné objekty metodou updateSystem(). Každému objektu nastaví novou hodnotu atributu accelerationX a accelerationY. Metoda má vždy k dispozici kolekci všech existujících objektů.

ObjectI zde představuje právě zkoumaný objekt. ObjectJ je vždy ten druhý objekt, přičemž je nutné projít celou kolekci objektů – tedy objectI je pořád ten samý, objectJ se mění.

```
procedure computeAcceleration()
    forceXSum = 0
    forceYSum = 0
    for all objects do
        if (objectI != objectJ)
            distanceX = objectJ.x – objectI.x
            distanceY = objectJ.y – objectI.y
            distance = sqrt(distanceX * distanceX + distanceY * distanceY)

            force = (GConst * objectI.weight * objectJ.weight)/(distance * distance)
            forceX = force * (distanceX / distance)
            forceY = force * (distanceY / distance)

            forceXSum += forceX
            forceYSum += forceY
    end for
    accelerationX = forceXSum / objectI.weight
    accelerationY = forceYSum / objectI.weight
end procedure
```

Poznámka: Kód navíc obsahuje ošetření případu, kdy je vzdálenost mezi objekty menší než součet jejich poloměrů, k čemuž dochází, pokud je kolize vypnutá. V tomto případě je jejich vzdálenost uměle nastavena na právě tento součet, aby nedocházelo k „vystřelování“ planet kvůli dělení příliš malou hodnotou distance.

Metoda checkForCollision()

Tato metoda je volána metodou updateSystem(), jestliže je zapnutá kolize. ObjectI znovu představuje právě zkoumaný objekt, ObjectJ jsou ostatní objekty. Metoda má přístup ke kolekci všech existujících objektů.

Ke kolizi dochází, pokud je vzdálenost objektů menší nebo rovna jak součet jejich poloměrů. Poté program rozhodne, který z objektů je větší a menší podle jejich váhy – ten větší se upraví a v kolekci se ponechá, ten menší je z kolekce vymazán.

Ratio představuje poměr obou objektů, aby ten menší neměl takový vliv na rychlost a zrychlení většího objektu. Nový poloměr výsledného objektu je vypočítán metodou findRadius(), kterou implementuje každý potomek SpaceObject. Nové pozice X i Y jsou aritmetický průměr obou pozic X a Y.

```
procedure checkForCollision()
	for all objects do
		if (objectI != objectJ)
			distanceX = objectJ.x – objectI.x
			distanceY = objectJ.y – objectI.y
			distance = sqrt(distanceX * distanceX + distanceY * distanceY)
			if(distance <= objectI.radius + objectJ.radius)
				ratio = smaller.weight / bigger.weight

				newWeight = bigger.weight + smaller.weight
				newRadius = bigger.findRadius()
				newX = (bigger.x + smaller.x) / 2
				newY = (bigger.y + smaller.y) / 2
				newSpeedX = bigger.speedX + (smaller.speedX * ratio)
				newSpeedY = bigger.speedY + (smaller.speedY * ratio)
				newAccelerX = bigger.accelerX + (smaller.accelerX * ratio)
				newAccelerY = bigger.accelerY + (smaller.accelerY * ratio)

				bigger.set()
				spaceObjects.remove(smaller)
	end for
end procedure
```

Poznámka: Nyní metoda kontroluje kolizi i u objektů, které jsou od sebe velmi daleko, což určitě není optimální stav a pravděpodobně by se kód dal v budoucnu vylepšit.

Utility třídy

Třída FileLoader

Knihovní třída, která obsahuje statické metody pro načítání ze souboru. Vstupní soubor je typu CSV (Comma-Separated Values), tudíž načtené řádky dělíme podle čárek.

Metoda loadFirstLine() načte první řádek souboru, jenž obsahuje gravitační konstantu a časový krok. Druhá metoda, loadSpaceObjects(), přeskočí první řádku a načte ty ostatní, které definují vesmírné objekty (název, typ, pozice X a Y, rychlost X a Y a váha). Podle typu bude v budoucnu zvolena vytvořená instance, nyní jsou vytvářeny defaultně jen planety. Načtené objekty jsou uloženy do kolekce, kde si je vyzvedne metoda main().

Při načítání váhy objektu je použita absolutní hodnota, protože váha nemůže nikdy být záporná. Gravitační konstanta a časový krok byl ponechán, protože by se s takovými zápornými hodnotami daly dělat zajímavé experimenty.

Třída Vectors

Knihovní třída pro práci s vektory. Obsahuje dvě metody – vectorSize(), která spočítá velikost vektoru, a vectorAddition(), která ze dvou vektorů vytvoří složený vektor a vrátí jeho velikost.

Rozšíření z 2. části

Kolize

Již implementovány v 1. části – viz kapitola Metoda checkForCollision().

Navíc byly do této metody přidány trajektorie. Při kolizi jsou tedy vytvořeny aritmetické průměry všech nasbíraných bodů trajektorií a uloženy jako nová trajektorie vzniklého objektu.

Graf rychlosti (neaktuální, viz kapitola Rozšíření z 3. části – Lepší grafy)

Pro tvorbu a vykreslení grafu je využita knihovna JFreeChart. Po kliknutí na objekt se zobrazí graf jeho rychlosti za posledních 30 reálných sekund. Nebyl použit simulační čas, protože některé scénáře mají časový krok mnohem vyšší než 30 sekund a hodnoty by musely být zpětně dopočítány.

Každý SpaceObject obsahuje kolekci určenou pro ukládání rychlostí, z níž graf získává data k zobrazení. Hodnoty jsou před vložením do grafu vyděleny konstantou 3,6, abychom získali z jednotek m/s jednotky km/h.

Ve třídě WindowInitializer je nastaven Timer, který volá metodu updateChart() a collectData() ze třídy DrawingPanel každých 96 ms, tedy přibližně desetkrát za sekundu (je nastaven na o něco menší dobu kvůli zpoždění). Třída WindowInitializer také obsahuje posluchač, který při kliknutí na planetu zavolá metodu showChart() ze třídy DrawingPanel, jež vytvoří instanci třídy ChartWindow pro zvolený SpaceObject.

Třída ChartWindow dědí od třídy JFrame, jedná se tedy o samostatné okno. Obsahuje atributy:

- clickedObject – uživatelem zvolený objekt,
- speedData – kolekce hodnot rychlostí objektu,
- chart – graf samotný,
- chartPanel – panel s grafem,
- DATA_COLLECTED_PER_S – kolikrát za sekundu se ukládá rychlost objektu.

Konstruktor této třídy volá metodu initWindow(), jež vytvoří okno a také zavolá metodu createChart(), která vytvoří samotný graf a přidá ho do panelu pro graf. Panel je pak přidán do okna.

V metodě createChart() je volána metoda createDataset(). Dataset je typovaný na XYSeriesCollection. Pokud již v simulaci proběhlo více než 30 sekund, je posunutý počáteční bod grafu tak, aby v grafu vždy bylo vidět právě 30 posledních sekund. To znamená, že z kolekce hodnot rychlostí objektu nejsou brána všechna data, ale je upraven počáteční index, od kterého jsou data vybírána.

Použitý graf je typu XYLine (spojnicový graf). Pomocí metod knihovny JFreeChart byly změněny barvy pozadí grafu i spojnic.

Pokaždé, když je volána metoda updateChart() ze třídy DrawingPanel, je znovu vytvořen graf metodou createChart().

Trajektorie pohybu

Každý SpaceObject má atributy představující neškálovanou i škálovanou trajektorii X a Y (celkem tedy 4 kolekce).

Metoda updateDrawing() ze třídy DrawingPanel byla upravena tak, aby mohla s trajektorií pracovat. Pokud je simulace aktivní a objekt nemá rychlost 0, je do kolekce neškálovaných trajektorií objektu přidán další bod. Pokud je velikost kolekcí větší než atribut trajectoryLength třídy DrawingPanel (nastaven na 60), je první bod v kolekcích odstraněn.

Poté jsou vytvořeny nové kolekce pro škálované trajektorie X a Y. Do těchto kolekcí jsou vloženy body z kolekcí pro neškálované trajektorie po tom, co je na všechny tyto body aplikováno škálování. Tyto kolekce jsou poté nastaveny jako atributy vesmírného objektu.

Nakonec je zavolána metoda drawTrajectory() nad objektem. Tuto metodu musí implementovat každý objekt dědící od SpaceObject, ale zatím je hotová pouze implementace pro Planet. Metoda pomocí cyklu prochází celé kolekce škálovaných trajektorií X a Y. Pro každý bod trajektorie je určena barva tak, že složky R, G a B odpovídají přímo planetě a složka A se mění tak, aby se trajektorie postupně zprůhledňovala. Bod trajektorie je realizován jako elipsa, jejíž velikost se mění tak, aby body trajektorie byly čím dál tím menší.

```
procedure drawTrajectory()
      for all coordinates in planetScaledTrajectory do
            color = findColor(i)
            trajectoryRadius = (planetDiameter / trajectoryLength) * i

            ellipseX = planetScaledTrajectoryX(i) + planetScaledRadius
            - trajectoryRadius / 2

            ellipseY = planetScaledTrajectoryY(i) + planetScaledRadius
            - trajectoryRadius / 2

            trajectoryPoint = new ellipse(ellipseX, ellipseY, trajectoryRadius)
            fill(trajectoryPoint)
      end for
end procedure
```

Rozšíření z 3. části

Export do PNG

Po kliknutí na tlačítko pro export je zavolána metoda exportPNG() ze třídy DrawingPanel. V této metodě je vytvořena nová instance BufferedImage pro obrázek a nový grafický kontext Graphics2D náležící obrázku. Na grafický kontext je vykreslen metodou paint() aktuální obsah okna. Pomocí komponenty JFileChooser je zvolena cesta k vyexportovanému obrázku. Obrázek je vyexportován metodou write() třídy ImageIO.

Export do SVG

Pro export do SVG je využita knihovna JFreeSVG. Po kliknutí na tlačítko pro export je zavolána metoda exportSVG() ze třídy DrawingPanel. V této metodě je vytvořena nová instance SVGGraphics2D pro obrázek. Na grafický kontext je vykreslen metodou paint() aktuální obsah okna. Pomocí komponenty JFileChooser je

zvolena cesta k vyexportovanému obrázku. Instance BufferedWriter vypíše do souboru veškerý obsah grafického kontextu. Obsah je získaný metodou getSVGElement() volanou nad grafickým kontextem.

Zrychlení / zpomalení simulace

Po kliknutí na tlačítko pro zrychlení simulace se zavolá metoda faster() třídy DrawingPanel. Ta (pokud je simulace aktivní) dvakrát zvětší časový krok simulace a update konstantu (kolikrát za jedno překreslení se mají přepočítat pozice, zrychlení a rychlosti). Při kliknutí na tlačítko pro zpomalení simulace se zavolá metoda slower(), která odpovídá metodě faster(), ale naopak hodnoty dvěma dělí.

Tlačítko Reset zavolá metodu resetTimeStep(), která změní časový krok a update konstantu na původní hodnotu ze začátku simulace.

Zvětšení a posun

Při události scrollování je volána metoda zoom(). Z události je zjištěno, zda šlo o rotaci kolečka myši nahoru nebo dolů a podle toho se zavolá metoda changeZoom() třídy DrawingPanel s kladnou nebo zápornou hodnotou. Tato metoda pouze přičte novou hodnotu k hodnotě atributu zoom.

Podle mačkání kláves W, S, A, D / šipek se mohou volat metody up() a right():

- W / ↑ - up(50)
- S / ↓ - up(-50)
- A / ← - right(50)
- D / → - right(-50)

Metody up() a right() nastaví atributy addUp nebo addRight na parametr, který jim byl předaný (zde 50 nebo - 50). Při každém volání metody updateDrawing() je potom kontrolováno, zda atributy addUp a addRight mají jinou hodnotu než 0. Pokud ano, je postupně jejich obsah přičítán do atributů up a right – to způsobí hladší posunutí vesmíru.

Při vykreslování objektů v metodě updateDrawing() jsou škálované souřadnice x a y vynásobeny atributem zoom. K souřadnici x je přičten atribut right, k souřadnici y je přičten atribut up. Škálovaný poloměr je vynásoben atributem zoom.

Při kliknutí na tlačítko Center jsou atributy up, addUp, right, addRight nastaveny na 0 a zoom je nastaven na 1, což způsobí vycentrování vesmíru zpět na střed.

Lepší grafy

Pro tvorbu a vykreslení grafu je využita knihovna JFreeChart. Po kliknutí na objekt se zobrazí graf jeho rychlosti za celou dobu simulace. Je použit simulační čas kvůli možnosti zrychlení a zpomalení simulace – v takových případech by použití reálného času vytvořilo nesmyslný graf.

Při každém volání metody updateSystem (každých 17 milisekund) jsou sesbírána data o rychlostech všech objektů a aktuálním času simulace. Rychlosti a časy se nachází v oddělených kolekcích jako atributy každého vesmírného objektu. V metodě collectData() je rozhodnuto, zda datová řada má více bodů, než je povoleno atributem MAX_CHART_VALUES třídy DrawingPanel. Pokud ano, je datová řada každého objektu redukována na polovinu metodou removeHalfData(), která se volá nad vesmírným objektem.

Metoda removeHalfData() vytvoří nové kolekce, do kterých nakopíruje hodnoty všech rychlostí a časů na sudém indexu. Pokud je hodnota rychlosti lokální extrém, je také ponechána, spolu s odpovídající hodnotou času. Takto upravená datová řada je poté vykreslena ve třídě ChartWindow stejně jako ve 2. části SP.