Programová dokumentace PlanetSystem

Marek Bečvář
12.2.2021

Obsah

I Úvod

PlanetSystem je programem pro Windows/Linux umožňující uživateli ve 2D vytvářet vlastní simulované planetární systémy. Simulace pracují se skutečnými fyzikálními vlastnostmi, které mohou být pro jednotlivá tělesa ve fázi editování upravována.

Program je pro Windows i Linux. Zdrojový kód celého programu je napsaný v programovacím jazyce Python 3 s důrazem na využíte OOP. Hlavní část programu - grafické rozhraní aplikace, je zajištěné knihovnou Pygame, která se stará také o příjem uživatelských vstupů. Další využitou nestandardní knihovnou je knihovna Numpy. Dále jsou k programu přidány standardní knihovny Pythonu (celý seznam níže).

Tento projekt byl vytvořený jako zápočtový program pro předmět Programování 1, ZS 2020/21 MFF UK.
Github stránka projektu: https://github.com/Marculonis21/PlanetSystem.

II Kompatibilita

Python Projekt byl vytvořen na verzi Pythonu 3.8.5.

Knihovny Seznam všech knihoven, které byly použité ve finální verzi projektu.

Jméno knihovny	Dokumentace	Standardní knihovna
Enum	https://docs.python.org/3/library/enum.html	✓
Copy	https://docs.python.org/3/library/copy.html	✓
Os	https://docs.python.org/3/library/os.html	✓
Pickle	https://docs.python.org/3/library/pickle.html	✓
Random	https://docs.python.org/3/library/random.html	✓
Sys	https://docs.python.org/3/library/sys.html	✓
Numpy - v1.19.4	https://numpy.org/doc/	✗
Pygame - v2.0.1	https://www.pygame.org/docs/	✗

Postup pro doinstalování potřebných knihoven je jednoduchý, přesněji popsaný v těchto zdrojích:
Numpy: https://numpy.org/install/
Pygame: https://www.pygame.org/wiki/GettingStarted

III Rozdělení projektu

Program je rozdělený do tří částí (modulů): **PlanetSystem.py**, **SpaceObject.py** a **UIObject.py**. **PlanetSystem.py** je centrální část propojující vnitřní logiku všech objektů s vykreslováním pomocí Pygame. **SpaceObject.py** řeší veškerou problematiku planetárních objektů v simulaci a pomáhá s vykreslováním a lehkou logikou práce s těmito objekty před fází simulace. **UIObject.py** slouží k definování tříd potřebných pro přijatelné uživatelské prostředí (Pygame samotné neobsahuje žádné UI) - např. tlačítka, textové vstupy, atd.

Dále podrobněji o zdrojových kódech, nejprve o pomocných modulech.

i. SpaceObject

Třídy Tento modul se dále dělí na dvě spolupracující třídy **Object** a **simVars**. Třída **simVars** je pomocná, vytváří strukturu pro ukládání a snazší práci se simulačními daty (poloha, rychlost) všech planetárních objektů.

- **Třída Object**

Tato třída slouží ke zpracování většiny logiky a vykreslování spojené s planetárními objekty. Rovnou v konstruktoru třídy jsou předávány defaultní fyzikální hodnoty pro daný objekt (poloha, rychlost, hmotnost).
Barva objektu je při vytváření zvolena náhodně.

Resetování Možnost resetovat záznamy ze simulací dává funkce *ResetSteps*, která maže všechna simulační data a navrátí těleso do původní polohy.

Set/Get Další je skupina typických set/get funkcí. Setovací funkce, umožňují měnit počáteční vlastnosti tělesa (využívány při editování simulace). Funkce

GetStepPos je návratová funkce, zprostředkovávající pozici tělesa v určitém časovém kroku. Ta je nejčastěji použita při vykreslování nebo kontrole kolizí.

Zoom funkce Skupina funkcí *ChangeCoordinates* je řešením pro umožnění přibližování/oddalování kamery v hlavním okně. Využívají výpočtů z lineární algebry - přepočítávání souřadnic mezi dvěma různými bázemi. Tato stejná změna souřadnic pro všechna vykreslená tělesa, spojená se změnou jejich vykreslovací velikosti, tvoří zoom efekt.

Vykreslení trajektorie Simulace umožňuje vykreslovat jak předpokládanou tak uplynulou trajektorii, kterou objekt letěl. To je zajištěno funkcí *DrawSimPath*, která na základě daných parametrů vykresluje buď přesně určitý počet kroků, kterým těleso poletí, nebo určitou hodnotu kroků v trajektorii, kterou těleso již letělo. Zároveň vykresluje varovné značení, kdyby mezi objekty mělo dojít ke kolizi.

Contains/Collides Poslední dvě funkce slouží ke kontrole kolizí. *Contains* specifikuje, jestli je daný pod na obrazovce uvnitř plochy tělesa (klik kurzorem). *Collides* pak řeší, jestli v daném časovém kroce nedošlo mezi vlastním tělesem a libovolným dalším ke srážce.

- **Třída simVars**

Důležitou proměnnou všech objektů v simulace je *simSteps*. Jedná se o list postupně naplňovaný objekty třídy **simVars** pomocí simulačních výpočtů. Jednotlivé objekty této třídy jsou vždy dvě uložené vektorové hodnoty (poloha, rychlost), ke kterým se může přistupovat pomocí jednoduchého zápisu *obj.pos* nebo *obj.vel* (position, velocity).

ii. UIObject

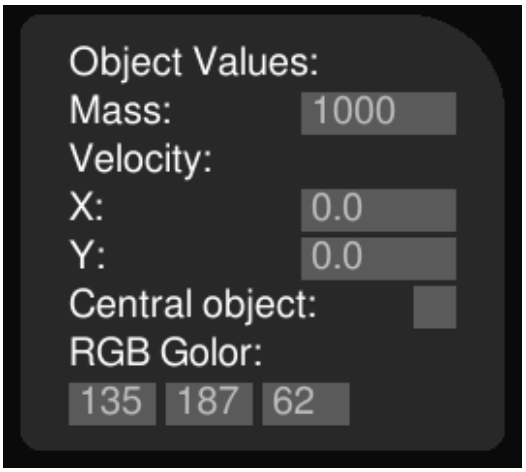
Úvod Vlastní modul *UIObject* se stará o celou logiku a vzhled UI prvků, se kterými může uživatel interagovat. Centrální je třída *UI*, která slouží jako šablona pro další ovládací prvky (tyto prvky třídu dědí).

- **Třída UI**

Tato třída definuje jakousi šablonu, kterou dědí ostatní uživatelské ovládací prvky. Pro všechny definuje základní potřebné proměnné a důležitou funkci *Draw*, která zvolený prvek a jeho další prvky (popisky, tlačítka, textové vstupy) vykresluje. I tak si ale další třídy dodefinovávají určité funkce do *Draw* samostatně.

- **Třída ObjPopup**

V této třídě se tvoří hlavní popup/widget obsahující menu s vlastnostmi těles v simulaci, které takto může uživatel snadno měnit. Pro tyto potřeby bylo dále nutno vytvořit samostatné třídy *InputField* a *RadioButton*, pro textové vstupy a pro funkci tlačítek (Pygame žádné UI elementy nenabízí).



ObjPopup opět dědí z třídy *UI*. V konstruktoru se předem definují všechny popisky a jejich počáteční pozice vůči levému hornímu rohu widgetu, samotné vstupy se musí generovat vždy při první vykreslení, protože počáteční hodnoty musí vždy odpovídat libovolnému zvolenému objektu.

Draw Funkce *Draw* této třídy přijímá jako další argument objekt, se kterým uživatel zrovna pracuje. Ten si udržuje po celou dobu, kdy je widget otevřený. Dále se zde předdefinuje poloha na obrazovce, kde se bude vykreslovat. Jak již bylo řečeno při prvním vykreslení, se musí založit všechny vstupy se správnymi hodnotami. O to se stará následující funkce *SetupInput*, kde je pevně vytvořené nastavení všech vstupů, pouze se doplní potřebné vlastnosti zpracovávaného objektu. Tato třída nevyužívá jako pozadí svého widgetu obrázek, proto ve své vykreslovací funkci ještě vytváří vlastní vzhled pozadí, vykreslené pod vstupy. Dále se potřebné proměnné předávají do původní zděděné *Draw* funkce pro vykreslování popisků a vstupů.

Event_handler Toto je nejrozsáhlejší metoda této třídy. Pracuje s logikou všech vstupních prvků, kontroluje správnost vstupů a zpracovává jejich propojení s upravovaným objektem. Podrobněji ve vlastních komentářích v kódu.

- **Třída InputField**

Tato třída je pomocná hlavnímu widgetu. Vytváří textové pole (preferovaný je číselný vstup), s pomocnou kontrolou správnosti vložených hodnot. Důležitou proměnnou v konstruktoru je *pointer*, specifikující kterou hodnotu vstup upravuje.

Draw Vykreslovací metoda této třídy je poměrně jednoduchá. Nejprve dle podmínek volí barvu textu, poté vytváří a vykresluje vlastní textový řetězec.

SetText Funkce *SetText* je klasický hodnotový setter s vlastní kontrolou zapisovaných dat. Pro toto využití je nejdůležitější možnost vytvořit limity, kterými se výstupy daného pole musí řídit.

GetDragValue Tato funkce tvoří ovládání pro změnu hodnot textového pole, pomocí tažení kurzoru. Uživatel může kliknout a držet textové pole a pohybem myši do libovolné strany, hodnotu zvyšovat/zmenšovat.

- **Třída RadioButton**

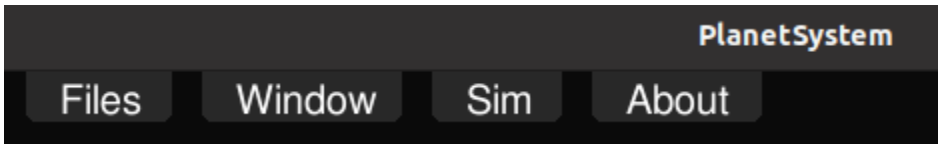
V této třídě se definuje zaskřtávací tlačítko, které umožňuje předávat hodnoty true/false.

Draw Metoda vykreslující tento vstup je také poměrně jednoduchá. Nejprve se zvolí text (výplň) tlačítka, podle bool proměnné *toggle*. Text je potom vykreslován ve správné barvě na předem vytvořené pozadí.

Event_handler Třída má vlastní *Event_handler*, protože jednotlivé vstupy je potřeba řešit blíže na straně samotného tlačítka. Pro možný event (stisknutí tlačítka) se zde řeší patřičné vizuální a logické změny.

- **Třída TopMenu**

TopMenu je další třída, která přímo dědí z hlavní šablony *UI*. Využívá se pro vytváření nabídky panelu nástrojů, který je uživateli vždy přístupný v horní části obrazovky.



Toto menu má formu záložek, které se při rozkliknutí mohou otevřít do více tlačítek, nebo do dalšího samotného okna.

Konstruktor Konstruktor této třídy dědí z původní třídy *UI*, ale je navíc obohacený o proměnné ukládající možný obsah tlačítek spadající pod danou záložku, barevný design záložky a rozvržení velikost prostoru, ve kterém bude každá záložka pracovat.

Tyto hodnoty jsou všechny určeny k úpravě a přesnému nadefinování při vytváření samotného ovládacího panelu v hlavní části programu.

Draw Vykreslovací funkce zde vyžaduje více proměnných, protože je potřeba zajistit správné zarovnání záložek vedle sebe i ve chvílích, kdy můžeme v programu zvolit libovolný rozměr tlačítka. Dále zde vykresluje jednotlivá tlačítka uvnitř záložky, pokud je daná záložka rozkliknuta.

Event_handler Vlastní *Event_handler* pak zajišťuje správné změny barev při interakci, kontroluje jestli je uživatel kurzorem stále v oblastni záložek (pro automatické zavírání), kontroluje potřebu otevřít záložku (kliknutí), nebo posunutí řešení eventů hlouběji do jednotlivých tlačítek záložky, pokud jsou otevřená.

Funkce taky pro různé výsledky vrací hodnoty, které jsou potřebné do logiky práce hlavního programu (př. záložka sama nedokáže otevřít připojené informační menu).

- **Třída MenuItem**

Tato třída je blízce spojená s předchozí třídou. Objekty třídy *MenuItem* tvoří vnitřní části záložek, zobrazené pouze po rozkliknutí dané záložky.

Konstruktor Pro jednotlivé objekty můžeme definovat jejich vnitřní text, pointer funkce na kterou ukazují, text možné klávesové zkratky a její pozici v tlačítku a samotnou šířku tlačítka.

Draw Pro vykreslování funkce nejprve dopočítává správnou polohu řádku, na kterém tlačítko bude. Poté již standardně vykreslí pozadí tlačítka a hlavní text (s možnou zkratkou).

Event_handler Typický vlastní *Event_handler*, nejprve zpracovává vizuální stránku tlačítka (změny barev při přejetí kurzorem přes tlačítko), poté možné kliknutí, po kterém by vracel hodnotu svého *pointeru*.

iii. PlanetSystem

Úvod Toto je centrální část celého program, která propojuje všechny moduly a sama provádí hlavní část výpočtů simulace prostředí.

Výčet proměnných V úvodu programu se nachází výčet všech proměnných (magických konstant), které jsou využívány napříč programem. Velmi důležité jsou zejména konstanty ovlivňující fyzikální simulaci. Důležitou proměnnou je *STEPTIME* ovlivňující kolik bude času mezi faktickými fyzikálními kroky (hodilo by se 0 vteřin, ale to pro nás není možné), menší hodnota vrací přesnější simulaci za ceny nárůstu výpočetní síly.

Hlavní cyklus aplikace Celý běhový cyklus aplikace se nachází v poslední části zdrojového kódu uzavřený do těla while cyklu. Ačkoli se jedná na první pohled o nekonečný cyklus, k zacyklení nedojde kvůli ošetření výstupů Pygame eventů, umožňující aplikaci řádně ukončit.

Další popis bude postupovat podle pořadí průběhu typického aplikačního cyklu.

- V první fázi dochází k vyčištění obrazovky z předchozího cyklu do předem nastavené barvy.
- Poté dochází ke kontrole, zda v seznamu simulovaných objektů není takový, ve kterém došlo ke změně (daný objekt nastaví svoji proměnnou *simUpdate* na True) a tudíž by bylo potřeba simulaci přepočítat. Podmínka *PAUSE and TIMESTEP == 0* ve své podstatě značí, že se stále jedná o fázi, kdy uživatel edituje objekty. Samotný výpočet simulace probíhá ve funkci *sim*, která pro určený počet kroků dopředu vypočítá polohu a rychlost každého z objektů v daném kroku simulace.
- Pokračuje se vykreslením všech planetárních objektů a jejich trajektorií.
- Dále se pro potřeby ukládání souborů volá funkce, která může zaznamenat snímek obrazovky (důležité ještě před vykreslením UI elementů).
- Další je vykreslování UI - horní panel záložek a widget objetku (pokud je nějaký zvolen).
- Pokud máme z předchozích cyklů vybraný nějaký objekt a uživatel stisknul šipku (posouvání objektu), voláme v této chvíli metodu pro posun na zvolený objekt.
- V tomto bodě se cesta dělí na tři stavy. SIM stav pracuje v době vlastní simulace a editování, stav LOAD a ABOUT zastavují simulační cyklus a vykreslují vlastní menu.

  Stav SIM:

  - Podobně jako u posunu objektu, posun kamery se řeší stejným způsobem. Program si pamatuje, které směrové klávesy má uživatel stisknuté a pro ně v dalším cyklu udělá požadovaný posun.
  - Další je řešení a předávání možných eventů (stisknutí tlačítka myši, klávesnice, pohyb myši, ukončení aplikace, atd.).
  - Pokud jsme ve fázi editování a uživatel vybral nějaké těleso, eventy hlavního okna se dále posílají ke zpracování do widgetu (textové vstupy, tlačítka).

- Stisknutí dalších kláves je posíáno do funkce *shortcutEvents* pro zpracování vybraných podporovaných klávesových zkratek, které uživatel mohl využít.
- Pak se také všechny možné eventy posílají jednotlivým záložkám horního *TopMenu* pro vlastní zpracování.
- Další jsou samostatně zpracovávané eventy spojené s kliknutím tlačítka myši, scrollování kolečka myši, pohyb pomocí kláves WSAD (kamera) a pohyb pomocí šipek (tělesa).
- Poslední část v SIM stavu zajišťuje, že spuštěná simulace vždy poběží co nejpřesněji ve zvoleném FPS (rychlosti). Vlastní Pygame cyklus je nepřesný a při vyšší zátěži je jeho délka proměnlivá. Proto je v tomto programu vedlejší hodnota počítající simulační cykly, která je nezávislá na rychlosti Pygame (pouze na reálném čase).

Stav LOAD:

- Při tomto stavu se zastaví všechny fyzikální výpočty a vykreslování a logika spojená s Pygame eventy se přesune do funkce *loadDrawing*. Zde je uživatel schopen načíst zpět své dříve uložené simulace.

Stav ABOUT:

- Podobně jako u LOAD, všechno vykreslování se přesouvá do funkce *aboutDrawing*. Ta vykresluje hlavní informační text v aplikaci. Text samotný nemá žádnou přidanou logiku, která by řešila Pygame eventy, tudíž pro možné ukončení aplikace je přímo do tělesa tohoto stavu přidáno možnost ukončení aplikace (LOAD tento stav řeší sám).

- Poslední částí jsou funkce Pygame, vykreslující připravenou grafiku do okna pro uživatele a opakování cyklu v čase zvoleném ve funkci *tick* (60 = FPS).

IV Ukládání souborů

LOAD menu nabízí uživateli možnost své simulace ukládat a znovu načítat. Tato funkce pracuje s externí složkou "*.saves*" ukrytou ve složce s programem. Pro správnou funkčnost se musí při ukládání zachovat určitá struktura.

Při ukládání se vytvoří pomocí knihovny **Pickle** soubor se všemi daty simulace a dalšího nastavení a pomocí vlastní screenshot funkce se zaznamená jak vypadá aktuální obrazovka. Oba dva soubory se uloží do složky "*.saves*".

Data simulace jsou uložené v souboru typu "save_#.sim" a snímek obrazovky "sImg_#.png". Z programátorského hlediska, ve fázi vývoje mohou být jména souborů externě změněna do tvaru "preset_#.sim" a "pImg_#.png". Tyto soubory pak *LOAD* funkce vnímá jako defaultní příklady a pro tyto uložené simulace pak nebude nabízena možnost soubor vymazat.

V Možnosti dalšího rozvoje

Ačkoli se již jedná o plnou, funkční verzi programu, jsou jistě věci, které bych osobně rád dodělával a vylepšoval i nadále. Vedle celkové optimalizace (rychlejší funkčnost pro uživatele, jiný platforma než Python) by se řadila ještě třeba lokalizace jazyků, úprava celkového UI designu a zajištění co možná nejširší kompatibility bez ztráty funkčnosti.