VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PLUGIN PRO ZÁZNAM A REKONSTRUKCI KRESBY V GRAFICKÉM EDITORU

PLUGIN FOR DRAWING RECORDING AND RECONSTRUCTION IN GRAPHICS EDITOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN HINER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ CHLUBNA**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Hiner Martin**
Program: Informační technologie
Název: **Plugin pro záznam a rekonstrukci kresby v grafickém editoru**
Plugin for Drawing Recording and Reconstruction in Graphics Editor
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s uživatelským prostředím a API vybraného grafického editoru (Krita, MyPaint, Blender...).
2. Navrhněte plugin, který rozšíří editor o možnost záznamu tahů štětce a možnou rekonstrukci kresby s upravenými parametry.
3. Nastudujte možná řešení dané problematiky.
4. Plugin implementujte a demonstrujte jeho použití na různých typech dat.
5. Zdokumentujte a zveřejněte výsledný plugin pro použití dalšími uživateli.
6. Vytvořte video reprezentující výsledky vaší práce.

Literatura:

- Blender Python API Documentation https://docs.blender.org/api/current/index.html
- Krita API https://docs.krita.org/
- Xu, S., Lau, F.C., Tang, F. and Pan, Y. (2003), Advanced Design for a Realistic Virtual Brush. Computer Graphics Forum, 22: 533-542. https://doi.org/10.1111/1467-8659.t01-2-00701

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty vedoucí k bodu 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cieľom tejto práce je preskúmať možnosť sledovania ťahov štetca v grafickom editore, ukladať ich a následne ponúknuť možnosť tieto ťahy individuálne upravovať alebo prehrať ako sekvenciu. Tieto vlastnosti sú následne implementované formou zásuvného modulu pre vybraný vhodný grafický editor.

Abstract

The goal of this thesis is to explore the possibilities of observing brush strokes inside a graphics editor, recording them, and finally, allowing the user to edit each individual stroke or play them out in a sequence. These features are then implemented in a form of a plugin module for a selected suitable graphics editor.

Kľúčové slová

grafický editor, plugin, digitálna kresba, záznam

Keywords

graphics editor, plugin, digital drawing, logging

Citácia

HINER, Martin. *Plugin pro záznam a rekonstrukci kresby v grafickém editoru.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Chlubna

Plugin pro záznam a rekonstrukci kresby v grafickém editoru

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána profesora Tomáša Chlubnu, Ing. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Hiner
18. mája 2022

Poďakovanie

Týmto sa chcem poďakovať pánovi Ing. Tomášovi Chlubnovi za jeho vedenie, čas, odbornú pomoc, rady a ochotný prístup. Ďalej sa chcem poďakovať vývojárom a komunite grafického editoru Blender za informácie z ktoré som pri práci využíval.

Obsah

Kapitola 1

Úvod

Grafické editory typicky ponúkajú len základné nástroje a funkcie na tvorbu a úpravu grafických prvkov. Táto sada nástrojov je síce dostačujúca na bežnú tvorbu ale pre užívateľa - umelca môže byť nevhodná, či už z hľadiska komplexných nastavení alebo potreby využitia špecifických vlastností, ktoré nie sú k dispozícii. Tento problém riešia zásuvné moduly (ďalej ako „pluginy“), ktoré slúžia ako voliteľné rozšírenia funkcií grafických editorov. Pluginy sú programové balíčky, ktoré je možné pridávať a odoberať za behu aplikácie. Ponúkajú takto užívateľsky prívetivú možnosť prispôsobiť si editor k vlastným potrebám.

Väčšina pluginov sa typicky zaoberá pridávaním nových vlastností spojených s tvorbou diela, napríklad nové vzory štetcov. Cieľom tejto práce je práve naopak zachytiť proces tvorby a ponúknuť možnosť tento proces retrospektívne upravovať a vizuálne ho prezentovať formou videa. Hlavnou výzvou pri implementácii je potreba využitia aplikačného rozhrania poskytovaného hostiteľskou aplikáciou pre tvorbu pluginov. Je nutné dodržať princípy pluginov a preto je zásah do zdrojového kódu aplikácie neprípustný, nakoľko by to znamenalo, že vyvinutý nástroj nie je pluginom ale rozšírením aplikácie.

Z tohoto dôvodu, prvým krokom tejto práce bolo nájsť vhodný grafický editor, ktorý v rámci svojho aplikačného rozhrania ponúka najlepší prístup k objektom reprezentujúcich grafické prvky kresby v editore. Rozhranie tiež musí obsahovať spôsob sledovania zmien týchto objektov v reálnom čase, aby bolo možné na ne reagovať a spracovať ich. Kapitola 2 priblíži vybraný editor a popíše proces prieskumu, spolu s nevyhovujúcimi alternatívami. Nasledujúca kapitola 3 sa zaoberá návrhom štruktúry pluginu pre vybraný grafický editor Blender. Kapitola 4 sa následovne venuje implementačným detailom výsledného pluginu, ktorý poskytuje možnosť zachytávať novovzniknuté ťahy štetca a následne ich prehrať ako animáciu.

Kapitola 2

Blender

S ohľadom na ciele tejto práce boli preskúmané nižšie popísané grafické editory. Z nich bol vybraný editor **Blender**, ktorý ako jediný splňoval všetky nutné podmienky, ktoré sú:

- neplatený prístup k aplikácii
- aplikačné rozhranie pre tvorbu pluginov s grafickým rozhraním
- prístup k objektom reprezentujúcich grafické prvky
- vstavaná možnosť sledovať zmeny v objektoch
- kompletná dokumentácia aplikačného rozhrania

Nižšie sú popísané relevantné základné pojmy, spojené s náplňou tejto práce a užívateľským prostredím Blenderu. Ďalej je priblížený proces prieskumu, motivácia pre výber Blenderu a ďalšie preskúmané alternatívy grafických editorov, spolu s ich nedostatkami pre účel tejto práce.

2.1 Prehľad

Blender je neplatený grafický editor distribuovaný pod licenciou *GNU General Public License*. Jeho aplikačné rozhranie, implementované v jazyku Python, sprístupňuje väčšinu funkcií a objektov. Zároveň je detailne dokumentované v oficiálnej dokumentácii[5], ako aj v rámci svojej komunity.

Aj keď Blender primárne slúži na animáciu a modelovanie v 3D prostredí, podporuje aj tvorbu 2D obrázkov. GUI pre túto tvorbu na prvý pohľad nie je intuitívne no i tak je v ňom možné plnohodnotne tvoriť (viď obrázok 2.1). 2D kresba je realizovaná v prostredí nástroja *Grease Pencil*, ktorý funguje ako dvoj-dimenzionálne plátno v rámci troj-dimenzionálneho prostredia. Štandardne ponúka všetky typické vlastnosti grafických editorov na tvorbu 2D grafiky a zároveň je možné pomocou neho tvoriť aj animácie.

2.2 Základné pojmy

Táto sekcia poskytuje vysvetlenia všeobecných pojmov ktoré sú použité v texte tejto práce.

Grafický editor Aplikácia slúžiaca na tvorbu a úpravu digitálnej grafiky.

Plugin Softvérový komponent, ktorý pridáva špecifickú funkcionalitu do existujúcej aplikácie.

API Aplikačné programové rozhranie, slúžiace na komunikáciu medzi programami.

GUI Grafické užívateľské rozhranie. Vizuálna časť aplikácie sprostredkujúca interakciu s užívateľom.

Plátno Oblasť v rámci užívateľského rozhrania grafického editoru, v ktorej sa nachádza grafika. V kontexte softvéru a tejto práce sa jedná o objekt v rámci kódu editoru, ktorý chceme sledovať a ktorý v sebe vo všeobecnosti obsahuje ďalšie objekty tvoriace výslednú grafiku.

Vrstva Abstraktná podjednotka plátna. Zjednocuje grafické objekty do skupiny a slúži k hromadnej úprave vlastností objektov, ktoré zastrešuje. Typicky sa jedná o ich hĺbku a viditeľnosť v rámci plátna.

Ťah štetca Základná grafická jednotka kresby ako celku. Parametrami ako tvar, dĺžka, hrúbka farba a priesvitnosť tvorí výslednú vizuálnu časť grafiky.

Rendering Proces spracovania grafických objektov na výsledný dvoj-dimenzionálny obrázok.[2]

2.3 Prostredie Blenderu

Pre lepšie pochopenie termínov použitých v tejto práci náleží, aby sa čitateľ zoznámil s užívateľským rozhraním aplikácie Blender, presnejšie, s jeho časťou použitou v texte tejto práce. Na názornú ukážku poslúži obrázok 2.1, na ktorom sú pomocou písmen **a** až **d** ilustrované jednotlivé termíny.

Scéna 3D priestor, v ktorom sa nachádzajú jednotlivé objekty.

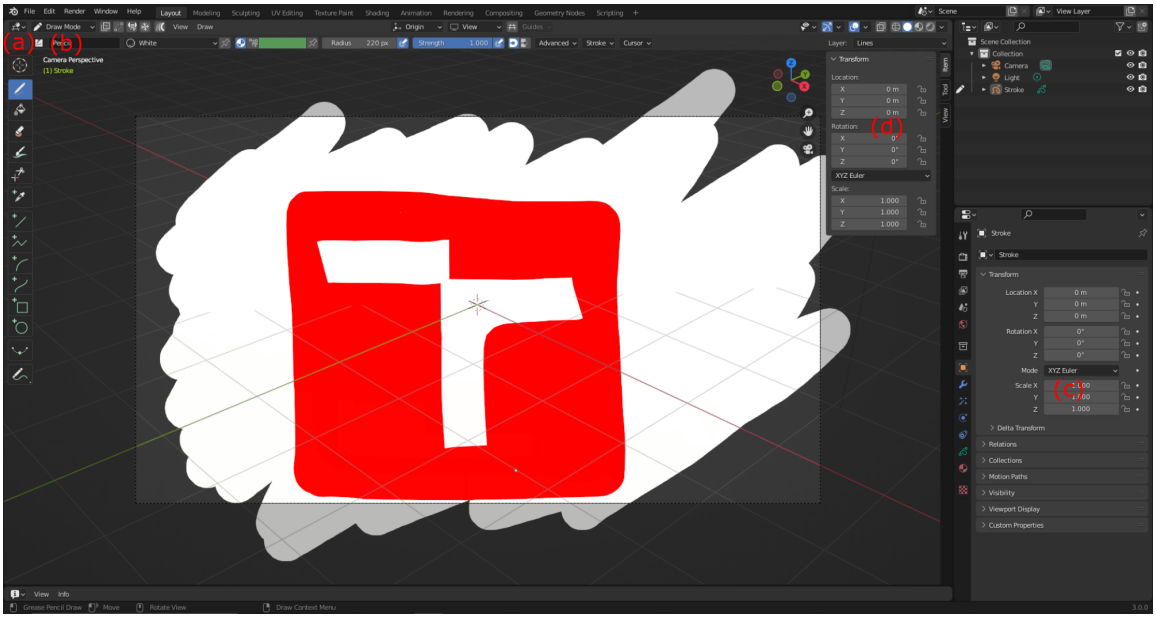
3D Viewport Editačné prostredie slúžiace na interakciu s 3D scénou. Spolu s ostatnými editačnými prostrediami ho možno vybrať z rozbaľovacej ponuky v ľavom hornom rohu okna (Obrázok 2.1 **(a)**).

Editačné módy Každé editačné prostredie ponúka sadu editačných módov slúžiacich na úpravu dát v prostredí. Pre túto prácu je dôležitý mód *Draw Mode* prostredia *3D Viewport*, ktorý slúži na kreslenie ťahov v nástroji *Grease Pencil*. Módy je možné vyberať z rozbaľovacej ponuky v ľavom hornom rohu (Obrázok 2.1 **(b)**).

Kamera Objekt nachádzajúci sa v priestore scény, slúžiaci ako bod pohľadu pre výsledný renderign scény. Na obrázku 2.1 je scéna s objektmi zobrazená z pohľadu kamery, do ktorého sa prepína stlačením numerickej klávesy *0*.

Panel s vlastnosťami Panel nachádzajúci sa na pravej strane okna (Obrázok 2.1 **(c)**) obsahuje vlastnosti a nastavenia objektov v scéne, ako aj samotnej scény. Záložky s nastaveniami sú dostupné podľa vybraného editačného prostredia [3].

Bočný panel s nástrojmi Vedľa panelu s vlastnosťami (Obrázok 2.1 **(d)**) je umiestnený vysúvací panel s nástrojmi. Panel je možné vysunúť stlačením klávesy *N* a nachádzajú sa v ňom rôzne nástroje v závislosti od vybraného editačného prostredia.



Obr. 2.1: Užívateľské rozhranie editoru Blender

2.4 Nástroj *Grease Pencil*

Ako základné prostredie na sledovanie bol vybraný nástroj *Grease Pencil*, slúžiaci na 2D vektorovú kresbu a animáciu v 3D prostredí. Tento objekt funguje ako kontajner ťahov nakreslených na vrstvách [1]. Na kreslenie ťahov slúži mód *Draw Mode* a detailná úprava ťahov je možná v módoch *Edit Mode* a *Sculpt Mode*.

Ťahy štetca sú implementované štýlom lomených čiar (anglicky polyline). Jedná sa o reťaz segmentov rovných čiar spájajúcich každý bod s jeho bezprostrednými susednými bodmi. Každý ťah je uložený ako troj-dimenzionálny vrchol (z-súradnica je nepoužitá/ignorovaná). Pre moduláciu hrúbky ťahu v každom jeho bode sú použité hodnoty tlaku pera grafického tabletu [6].

V kóde aplikácie Blender sa tento nástroj skladá z nasledujúcich, pre túto prácu dôležitých tried:

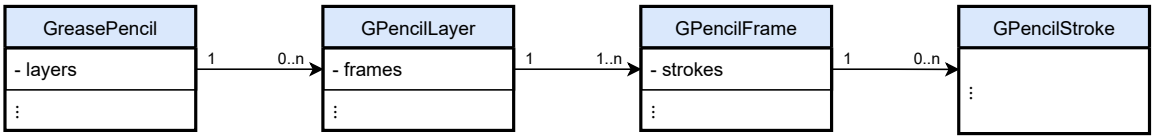
GreasePencil Hlavná trieda nástroju obsahujúca všetky dáta o kreslení a relevantné nastavenia. Táto trieda bude slúžiť ako vstupný bod pre sledovanie procesu kresby.

GPencilLayer Trieda reprezentujúca vrstvu kresby ako 2D plochu.

GPencilFrame Keďže Blender je editor podporujúci animáciu, je táto možnosť sprístupnená aj nástroju *Grease Pencil*. Táto trieda preto reprezentuje jednu snímku z animačnej sekvencie, ktorá obsahuje aktuálne ťahy štetca.

GPencilStroke Trieda obsahujúca dáta ťahu štetcom. Na rozdiel od predchádzajúcich tried, táto je vizuálne reprezentovaná v prostredí editoru. Vznik a zánik objektov tejto triedy a zmena dát budú generovať značnú väčšinu zachytených záznamov.

Na ilustráciu vzťahov medzi týmito triedami slúži diagram tried na obrázku 2.2:



Obr. 2.2: Hierarchia tried grafických prvkov v programovom prostredí Blenderu

2.5 Formálne požiadavky na tvorbu pluginu

Plugin pre aplikáciu Blender je implementovaný buď jedným súborom alebo formou modulu v jazyku Python a musí spĺňať formálne pravidlá aby bol aplikáciou akceptovaný. Základné pravidlá pre formu implementácie pomocou modulu sú nasledovné [4]:

- v koreňovom priečinku modulu sa musí nachádzať súbor `__init__.py` slúžiaci ako vstupný bod pre načítanie a prácu s modulom
- na začiatku súboru `__init__.py` musí byť definovaný slovník s názvom `bl_info` obsahujúci metadáta o tomto module[1]:
  - ➢ **name** názov modulu, ktorý bude zobrazený v grafickom rozhraní editoru
  - ➢ **author** meno autora alebo autorov modulu
  - ➢ **version** verzia modulu
  - ➢ **blender** verzia aplikácie Blender vo formáte trojice čísel[2]
  - ➢ **location** popis umiestnenia modulu v rámci grafického rozhrania
  - ➢ **category** názov kategórie pod ktorú modul spadá
- v súbore `__init__.py` musia byť implementované funkcie `register` a `unregister`, ktoré slúžia na registráciu použitých tried a definíciu dodatočných vlastností vstavaných tried aplikačného rozhrania.

Minimálna forma pluginu s použitím vyššie spomenutých pravidiel je naznačená vo výpise 2.1.

```
bl_info = {
    "name": "Minimal Plugin",
    "blender": (3, 0, 0),
    "category": "User"
}

import bpy

def register():
    bpy.utils.register_class(...)

def unregister():
    bpy.utils.unregister_class(...)
```

[1] kompletný popis formátu slovníka je dostupný na adrese: https://wiki.blender.org/wiki/Process/Addons/Guidelines/metainfo

[2] dátový typ jazyka Python `tuple`

Výpis 2.1: Minimálna forma pluginu pre Blender

Komunitné pluginy majú možnosť byť oficiálne prijaté do katalógu modulov aplikácie Blender, no musia spĺňať ešte ďalšie dodatočné podmienky:

- modul musí byť byť plne dokumentovaný
- na oficiálnej Wiki stránke musí mať modul vytvorenú vlastnú stránku s dokumentáciou
- kód modulu musí spĺňať požiadavky štýlovej konvencie *PEP 8* jazyka Python
- modul musí byť posúdený a schválený vývojárom z komunity Blenderu
- modul musí byť kompatibilný s najnovšou verziou aplikácie Blender
- modul nesmie byť závislý na binárnych súboroch[3]

2.6 Alternatívy

Pred samotným návrhom pluginu bolo nutné porovnať jednotlivé editory a vybrať z nich jeden, ktorý poskytuje najlepšie podmienky pre vytvorenie pluginu, tak, aby sa čo najviac priblížil funkcionalite opísanej v zadaní práce.

Prieskum aplikácie sa zvyčajne začínal študovaním dokumentácie, s cieľom zistiť na koľko aplikačné rozhranie spĺňa nutné podmienky. Základom je prístup k objektom reprezentujúcich grafické prvky na plátne za behu programu. Nemožnosť prístupu by znamenala že plugin by nebol schopný identifikovať sledovaný objekt, manipulovať s ním a zbierať o ňom dáta. Bez tejto vlastnosti by vývoj pluginu pre daný editor nebol možný. Ďalšou nutnou vlastnosťou, ktorú bolo potrebné overiť, bol spôsob sledovania zmien na objekte prístupný z API pre pluginy. Tento prirodzený spôsob sledovania má jasnú výhodu v tom, že je spravovaný priamo editorom a tak predchádza potrebe implementovať vlastný spôsob sledovania, ktorý by potencionálne mohol byť menej efektívnym a viac výkonne náročným v porovnaní so vstavaným spôsobom.

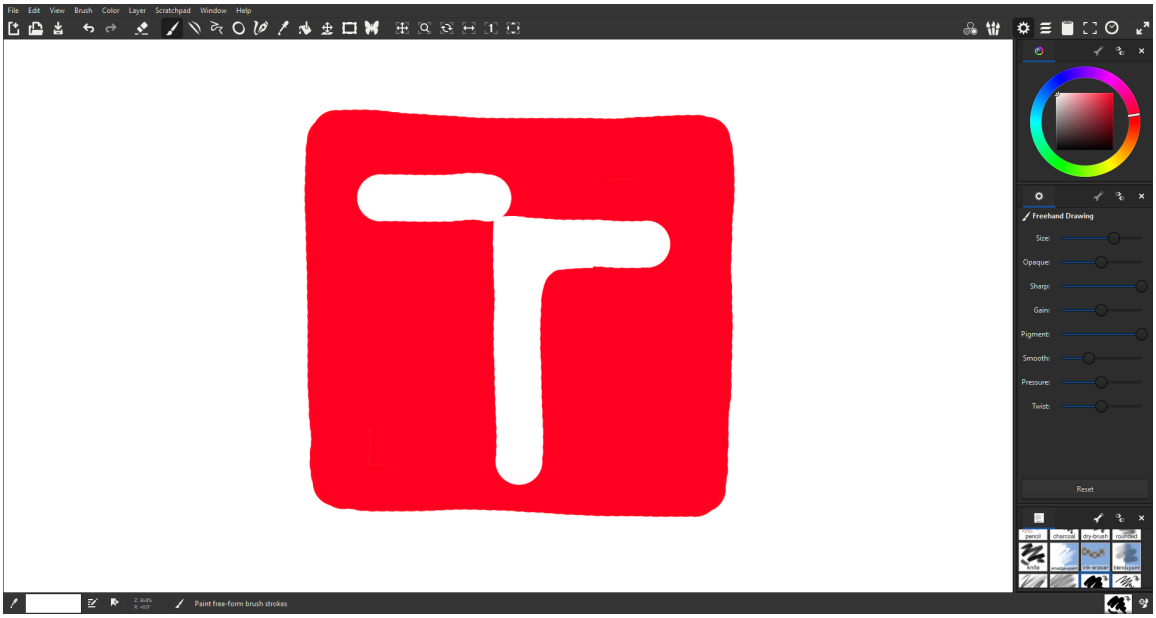
Po overení prítomnosti vyššie spomínaných vlastností nasledovala experimentácia s API formou implementácie minimálnych pluginov. Cieľom bolo v praxi overiť fungovanie spomínaných vlastností a zoznámiť sa s aplikačným rozhraním a jeho fungovaním.

MyPaint

MyPaint je neplatený grafický editor na tvorbu rastrovej grafiky. Z princípu minimalistická aplikácia (viď obrázok 2.3) sa viac sústredí na tvorbu kresieb než ich úpravu a spracovanie s výraznou podporou pre grafické tablety.

Po vyhodnotení výsledkov prieskumu bol najmenej vyhovujúcim kandidátom editor MyPaint. Hlavným dôvodom je slabá podpora vývoja pluginov - podporované je len pridávanie jednoduchých vlastností ako sú napríklad nové vzory štetcov, nejasné API a jeho dokumentácia.

[3] s výnimkou vlastných ikon v prípade nutnosti



Obr. 2.3: Užívateľské rozhranie editoru MyPaint

Krita

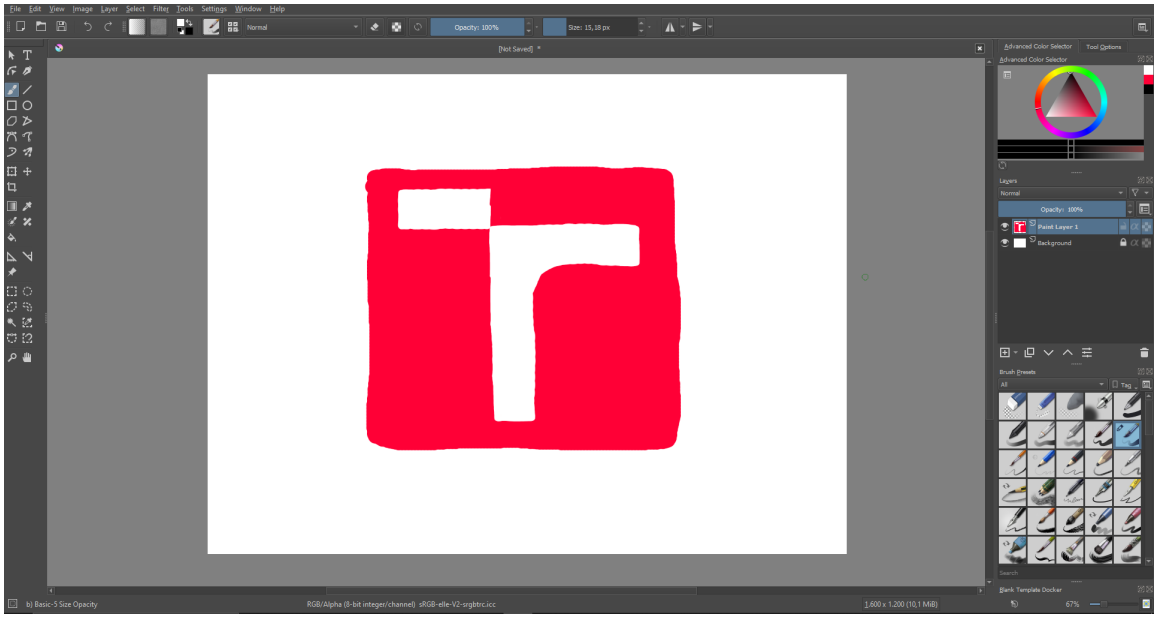
Krita je neplatený a open-source grafický editor navrhnutý pre digitálnych umelcov. Na obrázku 2.4 možno vidieť že v porovnaní s editorom MyPaint má Krita viac nástrojov pre tvorbu kresieb, čím sa Krita štylizuje ako kompletné riešenie pre tvorbu digitálnych kresieb, ilustrácií a animácií [7].

Vo vývoji od roku 1999 s pôvodným názvom *KImageShop* sa tento editor z počiatku zameriaval okrem kresby aj na manipuláciu s obrázkami na štýl editoru *Photoshop*. V súčasnosti sa editor sústredí len na aspekt kresby a je vyvíjaný s pomocou svojej komunity užívateľov - umelcov.

Editor Krita spĺňal požiadavky na API pre tvorbu pluginov ale chýbal spôsob sledovania zmien na objektom grafických prvkov prístupný pre pluginy. Ten bol prístupný len zo zdrojového kódu aplikácie, čo je ciele tejto práce nemožné, nakoľko by sa už nejednalo o plugin ale o priame rozšírenie aplikácie.

Motivácia

Editor Blender splnil všetky požiadavky spomenuté v kapitole 2, a preto bol v konečnom dôsledku vybraný na implementáciu cieľov tejto práce. Dôležitým faktorom bolo rozsiahle API sprístupňujúce väčšinu aplikácie pre tvorbu pluginov aj s grafickým rozhraním. Napriek tomu, že zamýšľaným účelom pluginov pre tento editor je úprava objektov tvorivého charakteru (napríklad procedurálna generácia objektov a textúr) sú v rámci API prístupné všetky objekty grafických prvkov a funkcií aplikácie. Je tak možné k nim pristupovať a sledovať vznikajúce zmeny v reálnom čase. API tiež obsahuje vstavané spôsoby, ako tieto zmeny sledovať. V neposlednom rade je toto API podrobne opísané v oficiálnej dokumentácii a zároveň má širokú komunitnú podporu. Vedľajšou výhodou je tiež že aplikácia Blender obsahuje vstavanú konzolu a textový editor pre jazyk Python, ktorú možno využiť pre prístup k API za behu aplikácie a zároveň testovať prvky vyvíjaného pluginu.



Obr. 2.4: Užívateľské rozhranie editoru Krita

Kapitola 3

Návrh

Ako vyplýva z predchádzajúcej kapitoly, pre implementáciu cieľového pluginu tejto práce bol vybraný editor **Blender**. Táto kapitola sa venuje návrhu štruktúry pluginu, menovite:

- návrhu formy a štruktúry pluginu
- návrhom vzhľadu a umiestnenia grafického užívateľského rozhrania pluginu
- spôsobu sledovania a zaznamenávania zmien
- obmedzeniam spojených s implementáciou pluginu

3.1 Forma pluginu

Z dôvodu rozsiahlosti a komplexnosti tejto práce je plugin implementovaný formou modulu rozdeleného do ďalších podmodulov a súborov, podľa ich funkcie v rámci modulu. Dôraz je kladený na dodržiavanie pravidiel spomenutých v kapitole 2.5 s cieľom vytvoriť kvalitný modul, vhodný na prípadné budúce rozšírenie. Konkrétne, okrem povinných požiadavkov na formu, je počas vývoja napísaný kód kontrolovaný pomocou nástroja *flake8* starajúceho sa o dodržiavanie konvencie jazyka Python *PEP 8*. Ďalej, vzniknutý kód je štandardne komentovaný a pre lepšiu pochopiteľnosť je tiež použitý type hinting[1]. Výsledkom je teda konzistentný a prehľadný kód.

Pre lepšiu prehľadnosť kódu je každý podmodul zodpovedný za registráciu a odregistráciu trieda vlastností vstavaných tried, ktoré implementuje. Realizované to je implementáciou funkcií `register` a `unregister` v súbore `__init__.py`, pre každý podmodul. Tieto funkcie budú následne volané zo súboru `__init__.py` v hlavnom module. Dodatočne je potrebné, ochrániť tieto operácie pred výnimkami a tak v prípade chyby zabezpečiť korektné ukončenie.

V poslednom rade, výsledný modul je licencovaný pod licenciou *GNU General Public License v3.0*[2]. Súbor s licenciou je umiestnený v koreňovom priečinku modulu a skrátená verzia licencie je umiestnená v hlavičke každého súboru.

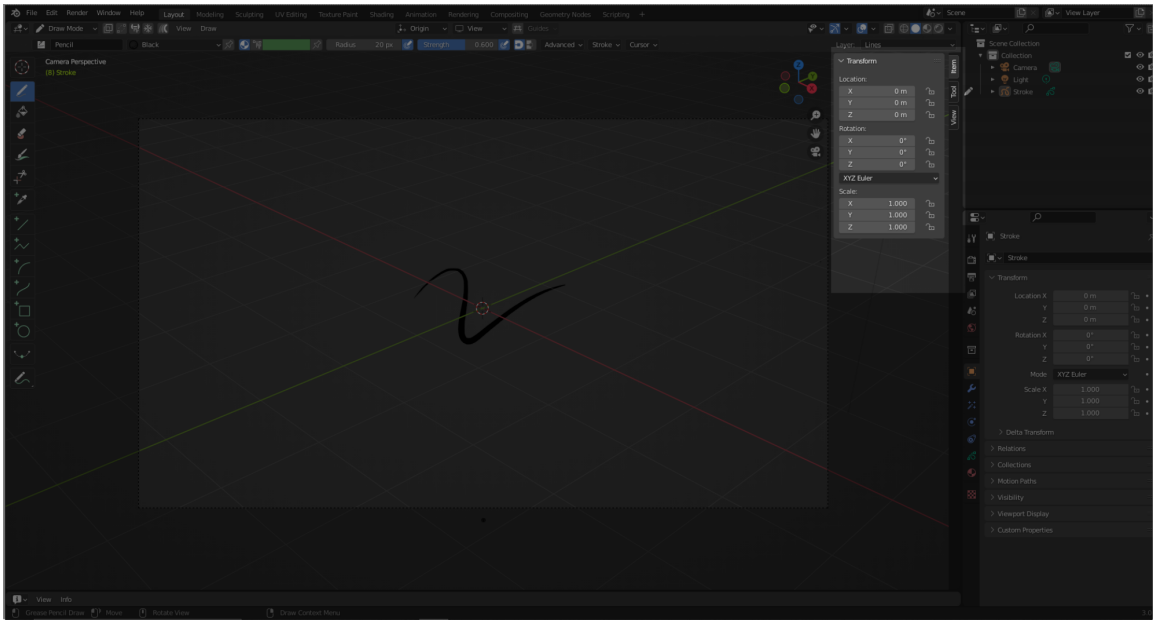
[1] programovací jazyk Python je dynamicky typovaný, čo znamená že typy vstupných parametrov a výstupnej hodnoty funkcií nie sú pevne dané a môžu sa za behu meniť; type hinting je dobrovoľné statické otypovanie týchto parametrov s cieľom naznačiť očakávaný vstup a výstup

[2] plné znenie licencie dostupné na stránke https://www.gnu.org/licenses/gpl-3.0.html

3.2 Užívateľské rozhranie

Grafické užívateľské rozhranie (ďalej len ako GUI, vysvetlené v kapitole 2.2) je dôležité pre zobrazovanie dát a získavanie vstupu v užívateľsky prijateľnom formáte. Keďže cieľom tohoto pluginu je poskytovať umelcovi informácie a umožniť mu s nimi pracovať, bolo nutné navrhnúť vhodné rozhranie a umiestniť ho do prostredia editoru.

GUI pluginu bude umiestnené v rámci okna Blenderu v bočnom paneli prostredia *3D Viewport* v karte *Layout* (viď obr. 3.1). Táto lokácia bola vybraná z dôvodu že plugin na svoj beh nepotrebuje stálu pozornosť užívateľa a tak je vhodné GUI vložiť do spomínaného bočného panelu, ktorý sa dá podľa potreby skryť. Výsledkom je že GUI svojou prítomnosťou nerozptyľuje užívateľa a nezatieňuje ostatné prvky grafického rozhrania samotnej aplikácie.



Obr. 3.1: Umiestnenie užívateľského rozhrania pluginu

Rozhranie pluginu má dva stavy zobrazenia (viď obr. 3.2).

- neaktívny stav
- aktívny stav

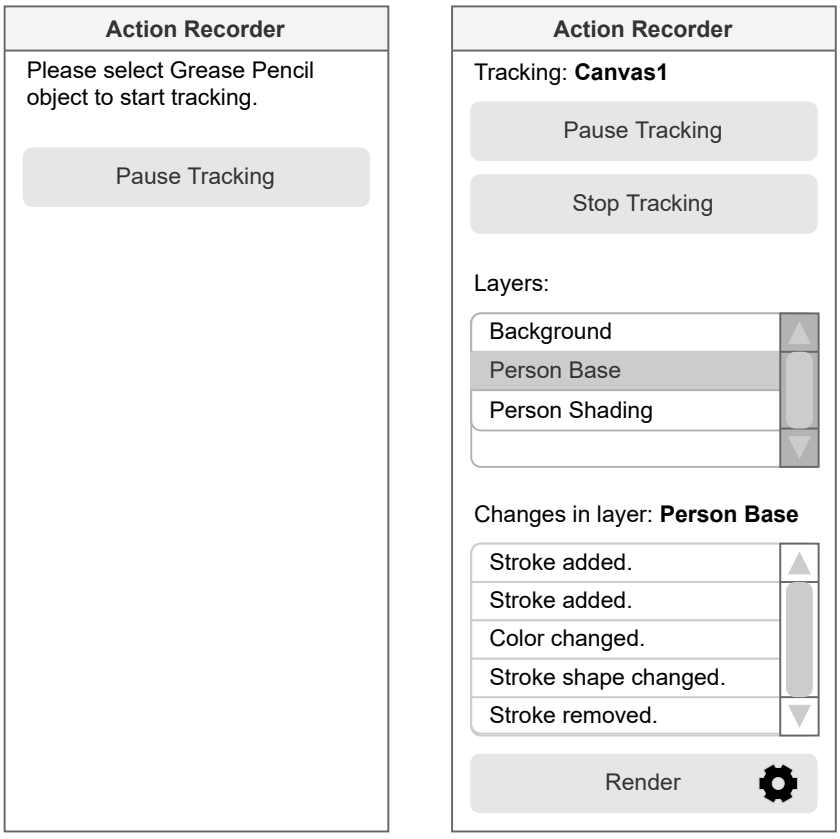
Neaktívny stav je východiskový stav v ktorom sa rozhranie nachádza pri spustení pluginu a medzi sledovaniami objektov. Prechod do aktívneho stavu nastáva vybratím objektu nástroja *Grease Pencil* v editačnom prostredí a kliknutím na tlačidlo *Start tracking* v paneli pluginu, čím sa GUI presunie do aktívneho stavu. V prvých verziách návrhu GUI sa v neaktívnom stave nachádzala aj možnosť vybrať objekt na sledovanie pomocou jeho mena. Táto možnosť bola neskôr zavrhnutá a to z dôvodu že je v porovnaní s jednoduchým výberom objektu z pohľadu užívateľa neatraktívna.

V aktívnom stave rozhranie v prvej sekcii zobrazuje názov sledovaného objektu a tlačidlá na riadenie stavu sledovania. Dvoj-stavové tlačidlo *Pause tracking*/*Resume tracking* slúži na pozastavenie a následné obnovenie sledovania zmien na vybranom objekte. Motivácia za vznikom tohoto tlačidla je možnosť pozastaviť sledovanie a vykonať viacero zmien na plátne. Následným obnovením sledovania sa nezachytené zmeny na plátne budú brať ako

jedna ucelená zmena. Výsledkom tak bude skoková sekvencia skutočných zmien na plátne ako alternatíva k plynulej sekvencii jednotlivých zmien. Tlačidlo *Stop tracking*, slúži na ukončenie sledovania objektu a rozhranie prejde naspäť do neaktívneho stavu. Primárne využitie tohoto tlačidla je zmena sledovaného objektu a uloženie nazbieraných dát z predchádzajúceho objektu do perzistentnej pamäti.

Druhá sekcia obsahuje zoznam vrstiev v objekte slúžiaci na výber vrstvy, pre ktorú sa budú v rozhraní zobrazovať zmeny. V sekcii sa ďalej sa nachádza štítok s názvom vybranej vrstvy a zoznam zachytených zmien zobrazujúci zmeny vo vybranej vrstve v poradí, v ktorom sa vyskytli. Kliknutím na položku tohoto zoznamu sa v aplikačnom rozhraní označí objekt ťahu, ktorého sa zmena týka a týmto sa objekt sprístupní pre ďalšie úpravy. Oba spomínané zoznamy budú aktualizované v reálnom čase, v závislosti od vzniknutých zmien.

Nakoniec, v spodnej časti rozhrania, sa nachádza tlačidlo *Render*, ktorým sa spustí proces renderovania zachytenej sekvencie. Vedľa sa ešte nachádza tlačidlo so symbolom ozubeného kolesa, ktoré otvorí kontextové okno s nastaveniami spojenými s renderovaním, menovite: cesta k priečinku, do ktorého sa uloží vyrenderovaná sekvencia spolu s animáciou, počet snímkov za sekundu a začiarkavacie políčko značiace, či sa má renderovať z pohľadu kamery.



(a) Neaktívny stav (b) Aktívny stav

Obr. 3.2: Stavy grafického rozhrania pluginu

3.3 Sledovanie zmien

Sledovanie nastávajúcich zmien v objektoch spomínaných tried (sekcia 2.4) bude realizované pomocou implementácie sledovacích tried pre každú spomínanú triedu. Tieto sledovacie triedy budú kopírovať hierarchiu tried reprezentujúcich grafické prvky (ilustrované na obrázku 2.2) a ich úlohou bude zbierať dáta o zmenách na objektoch o jeden stupeň nižšie v hierarchii a informovať o nich sledovaciu triedu najvyššie v hierarchii, ktorá je zodpovedná za uchovávanie dát. V praxi teda, pre príklad, sledovacia trieda pre objekt triedy `GPencilFrame` má za účel sledovať vznik nových objektov triedy `GPencilStroke` a reagovať naň odoslaním správy o zmene do objektu sledovacej triedy pre triedu `GreasePencil`.

Rovnako ako trieda `GreasePencil`, ktorá obsahuje všetky dáta o kreslení, bude aj jej sledovacia trieda slúžiť ako koreňový prvok procesu sledovania. Za úlohu bude mať spracovanie zachytených zmien pre použitie v grafickom rozhraní pluginu a tiež proces tvorby sekvencie výslednej animácie. Samotná tvorba výslednej animácie, ako videa, však bude prenechaná inej časti pluginu.

Pre sprístupnenie získaných dát pre zobrazenie v GUI a celkovo riadenie chodu pluginu je dôležité mať hlavný prístupový bod. Pre tento účel bude implementovaná trieda podľa návrhového vzoru Jedináčik (z anglického Singleton). Znamená to, že trieda bude v rámci kódu inštanciovaná len jeden krát a rovnako, v rámci kódu, sa bude narábať len touto jednou inštanciou.

Úlohou tejto triedy bude iniciácia a ukončenie procesu sledovania pomocou inštanciácie hlavnej sledovacej triedy. Ďalej táto databáza bude sprostredkovávať prístup k objektom hlavnej sledovacej triedy pre použitie v iných častiach kódu (napríklad v grafickom rozhraní). V poslednom rade sa bude starať o perzistentné ukladanie a načítanie zachytených dát obsiahnutých v objektoch hlavnej sledovacej triedy.

3.4 Tvorba animácie

Na tvorbu výslednej animácie bude priamo využitá možnosť editoru Blender na tvorbu animácií. Pri každej zachytenej zmene bude vytvorená nová snímka v animačnej sekvencii. Po ukončení práce teda bude mať užívateľ možnosť vyrenderovať zachytenú sekvenciu snímkov vo forme videa.

Ako bolo spomenuté v sekcii 3.2, bude mať možnosť upraviť nastavenia spojené s renderingom. Jednať sa bude o výber priečinku v ktorom bude výsledná animácia uložená, ďalej počet snímkov za sekundu a či má rendering prebiehať z pohľadu kamery nachádzajúcej sa v scéne.

Na základe týchto nastavení bude vytvorená výsledná animácia obsahujúca celý zachytený proces kresby.

3.5 Obmedzenia

Počas procesu návrhu a experimentovania s editorom a vývojom modulu vyšli najavo rôzne obmedzenia sťažujúce implementáciu modulu.

Prvé hlavné obmedzenie vyplýva z podstaty zásuvného modulu. Zásuvný modul musí byť samostatný komponent, ktorý pripojením do aplikácie rozšíri jej funkcionalitu, preto nieje prípustné modifikovať zdrojový kód aplikácie. V kontexte tejto práce to znamená, že nieje možné rozšíriť triedy sledovaných objektov, čo by poskytlo možnosť implementovať

sledovanie pomocou návrhového vzoru Pozorovateľ (z anglického Observer). Tento vzor v skratke funguje na princípe, že sledovaný objekt pri zmene svojich vlastností sám informuje svojich registrovaných pozorovateľov o tejto konkrétnej zmene a tak nieje nutné túto zmenu manuálne hľadať. Rozšírenie tried a implementovanie tohoto návrhového vzoru by pri vývoji modulu poskytlo plnú kontrolu nad spôsobom, akým sa vyhľadávajú zmeny ale bohužiaľ v prostredí aplikácie Blender je táto možnosť pre zásuvné moduly neprístupná je preto nutné sa spoľahnúť na vstavané spôsoby sledovania alebo v horšom prípade implementovať vlastné.

Druhé obmedzenie sa týka zobrazenia dát v grafickom rozhraní aplikácie. Blender umožňuje zobrazenie dát vo svojom grafickom rozhraní len svojim vstavaným triedam, ktoré obaľujú základné dátové typy jazyka Python. Na tieto vstavané triedy sa tiež pozerá ako na vlastnosti objektu a musia byť preto registrované pod konkrétnou podtriedou rozhraní `ID`, `Bone` alebo `PoseBone`. Toto mierne komplikuje prácu s dátami, ktoré majú byť zobrazené pretože je ich nutné definovať ako samostatnú podtriedu dátového typu, registrovať v rámci aplikácie a nakoniec aj odregistrovať pri vypnutí aplikácie alebo vysunutí modulu.

Posledné obmedzenie je spojené s úpravou jednotlivých ťahov, ktoré boli zachytené za behu pluginu. Ako bolo vyššie spomenuté, pri zaznamenaní novej zmeny sa vytvorí nová snímka v animácii, presnejšie, vytvorí sa nový objekt triedy `GPencilFrame` do ktorého sa nakopírujú všetky objekty ťahov z predchádzajúceho snímku. Úprava ťahu by teda žiadala preniesť túto zmenu na všetky predchádzajúce objekty tohoto konkrétneho ťahu, čo je samo o seba výkonne náročná operácia hlavne z dôvodu identifikácie zmeneného ťahu v každej snímke. Ďalej ešte API Blenderu nepodporuje odkazovanie na svoje objekty, čiže by bolo nutné odkazovanie manuálne implementovať. Z týchto dôvodov teda úprava ťahov v rámci pluginu nebude implementovaná.

Kapitola 4

Implementácia

Na základe návrhu v kapitole 3 bol implementovaný funkčný plugin pre grafický editor **Blender**. Výsledný plugin je schopný zaznamenávať vznik a zánik vrstiev a ťahov plátna pre užívateľom zvolené prostredie nástroja *Grease Pencil*. Zachytené zmeny sú v reálnom čase zobrazené v grafickom užívateľskom rozhraní pluginu. V poslednom rade je zachytenú sekvenciu možné exportovať ako animáciu. Užívateľ má možnosť zvoliť si lokáciu, počet snímkov za sekundu a uhol pohľadu pri tvorbe tejto animácie.

Cieľom tejto kapitoly je ozrejmiť spôsob fungovania pluginu a priblížiť implementačné detaily použité pri jeho vývoji.

4.1 Štruktúra modulu

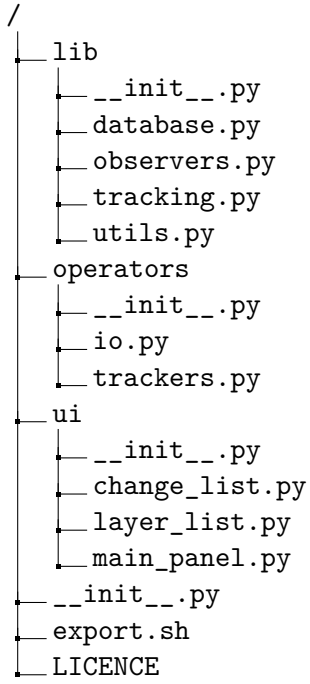
Ako bolo už spomenuté v sekcii 2.5, z dôvodu rozsiahlosti a komplexnosti je táto práca implementovaná formou modulu jazyka Python. Tento modul je ďalej členený do podmodulov podľa svojej funkcionality. Pre zjednodušenie réžie sú v každom podmodule implementované funkcie `register` a `unregister`, v ktorý sa podmodul sám stará o registráciu tried a vlastností a ktoré sú volané v rámci súboru `__init__.py` v koreňovom súbore modulu.

Súčasná štruktúra priečinka pluginu je vo výpise 4.1:

Podmodul `lib` obsahuje implementácie funkčných častí pluginu. Jedná sa o triedy a funkcie podieľajúce sa na procese sledovania zmien. V súbore `utils.py` sú dodatočne implementované podporné funkcie pre logovanie, registráciu a odregistráciu tried a vlastností pre použitie v rámci API Blenderu.

Podmodul `operators` je spojením funkcionality a grafického rozhrania pluginu. Sú v ňom implementované operátory, starajúce sa o riadenie chodu pluginu. V rámci GUI sú reprezentované a tlačidlá a prípadne vyskakovacie dialógové okná slúžiace na komunikáciu a získanie hodnôt od užívateľa.

Grafické užívateľské rozhranie pluginu je implementované v podmodule `ui`. Nachádzajú sa tu implementácie panelu a zoznamov rozhrania spolu s definíciou rozloženia jednotlivých prvkov.



Obr. 4.1: Štruktúra priečinku pluginu

4.2 Užívateľské rozhranie

Na základe návrhu vzhľadu a funkcionality spomenutej v sekcii 3.2 bolo vytvorené grafické rozhranie spĺňajúce všetky stanovené ciele. Jednotlivé prvky rozhrania sú implementované v dvoch podmoduloch. Podmodul `operators` obsahuje implementáciu tlačidiel a dialógových okien, spolu s ich funkcionalitou. Hlavný panel rozhrania definujúci rozmiestnenie prvkov a ich správanie, spolu so zoznamami sú implementované v podmodule `ui`.

Tlačidlá

Tlačidlá použité v rozhraní sú implementované ako podtriedy vstavanej triedy API Blenderu `Operator`, ktorá slúži na implementáciu funkcionality tlačidiel [5]. Konvencia vývoja pluginov pre Blender žiada, aby triedy implementovaných operátorov boli nazvané podľa regulárneho výrazu `[A-Z][A-Z0-9_]*_OT_[A-Za-z0-9_]+` [4]. Prvá časť výrazu by mala byť názov pluginu, pod ktorý operátor spadá, napísaný veľkým písmom. Nasleduje reťazec `_OT_`, značiaci že daná trieda je implementáciou operátoru. Názov je zakončený stručným popisom funkcie operátoru napísaný v štýle *snake case*[1].

Ďalej je nutné, aby implementovaná trieda operátoru obsahovala triedny atribút s názvom `bl_idname`. Jedná sa o reťazec, podľa ktorého je možné operátor identifikovať v mennom priestore aplikácie Blender. Operátory implementované v rámci tejto práce obsahujú spomínaný reťazec vo forme `action_recorder.` plus stručný popis funkcie operátoru, rovnako ako je napísaný v názve triedy. Takže pre operátor triedy `RECORDER_OT_pause_tracking` je jej atribút `bl_idname` napísaný ako reťazec `action_recorder.pause_tracking`.

[1] štýl písania, v ktorom je každá medzera nahradená znakom podčiarknutia

Ďalšie, síce nepovinné ale dôležité, triedne atribúty sú `bl_label` obsahujúci názov tlačidla zobrazený v rámci grafického rozhrania a `bl_description` obsahujúci opis funkcie tlačidla, ktorý sa zobrazí pri nabehnutí kurzoru myši na tlačidlo.

Funkcionalita tlačidla je podľa požiadavkov API implementovaná v metóde `execute` [5]. Prípadné rozšírenie tlačidla o dialógové okno sa dá dosiahnuť implementáciou metód `draw` a `invoke`, ktoré definujú obsah okna a vykreslia ho na grafickom rozhraní aplikácie Blender.

Dialógové okná sa dajú rozšíriť o užívateľské vstupy deklaráciou triednych atribútov s použitím vstavaných tried rozhrania `bpy.props`. Po získaní vstupu a ukončení okna sú získané hodnoty dostupné na spracovanie v rámci metódy `execute`. Príklad deklarácie vstupnej hodnoty operátoru je ilustrovaný vo výpise 4.1.

```
fps: IntProperty(
    name='Framerate',
    description='Number of frames per second in animation.',
    default=24
)
```

Výpis 4.1: Príklad deklarácie vstupnej hodnoty dialógového okna vybraný z implementácie triedy `RECORDER_OT_render_settings`

S použitím vyššie spomenutých pravidiel boli implementované nasledujúce triedy operátorov:

`RECORDER_OT_start_track_active` spustenie procesu sledovania vybraného objektu nástroja *Grease Pencil*

`RECORDER_OT_stop_track_active` ukončenie procesu sledovania

`RECORDER_OT_pause_tracking` dočasné pozastavenie procesu sledovania

`RECORDER_OT_resume_tracking` obnovenie procesu sledovania

`RECORDER_OT_render` konverzia zachytenej sekvencie na výslednú animáciu

`RECORDER_OT_render_settings` spustenie dialógového okna pre upravenie nastavení spojených s renderingom

Panel a zoznamy

Hlavný panel grafického rozhrania pluginu je implementovaný v súbore `ui/main_panel.py` ako trieda `RECORDER_PT_main_panel`.

Lokácia panelu je uložená v jeho triednych atribútoch. Menovite:

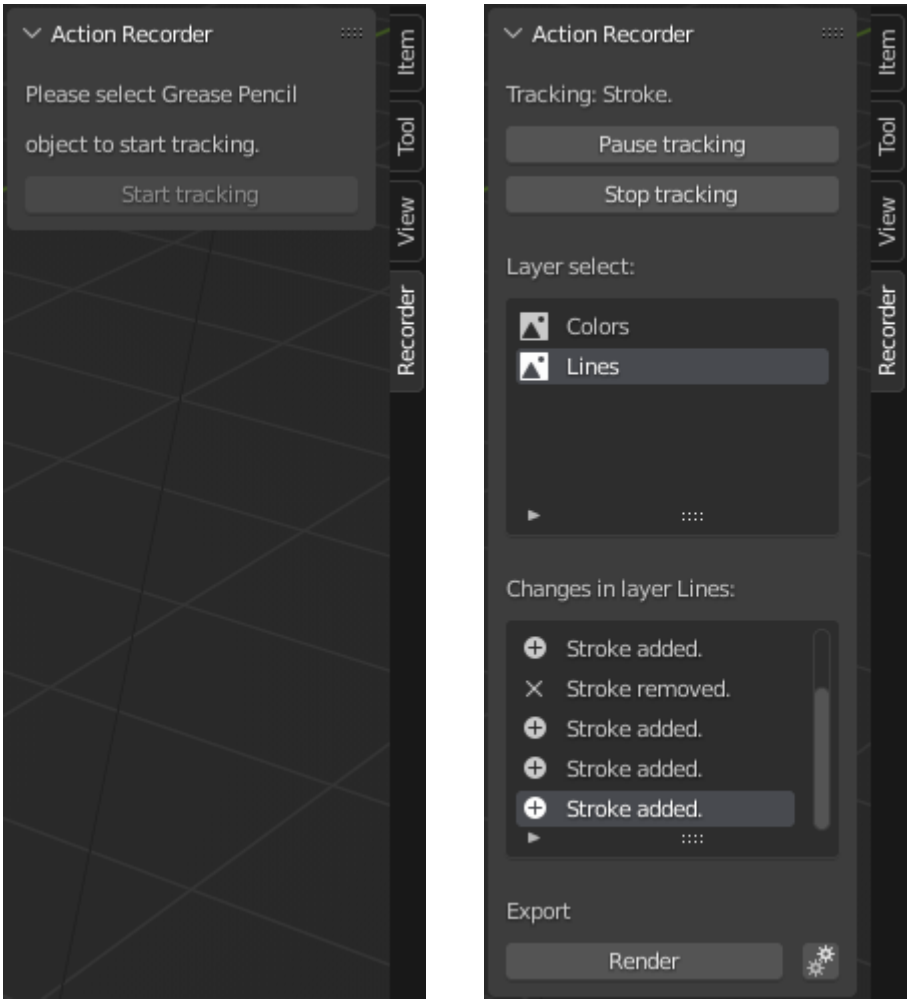
`bl_category = "Recorder"` názov karty panelu GUI

`bl_space_type = "VIEW_3D"` umiestnenie panelu v GUI Blenderu, presnejšie umiestnenie do editačného prostredia *3D Viewport*

`bl_region_type = "UI"` umiestnenie panelu v oblasti editačného prostredia, reťazec `UI` reprezentuje bočný panel s nástrojmi

Týmto spôsobom bola naplnená navrhnutá lokácia z kapitoly 3.

Rozloženie jednotlivých prvkov grafického rozhrania pluginu je definované v metóde `draw`. Zároveň je tu implementovaná funkcionalita dvoch stavov rozhrania, spomenutá v sekcii 3.2. Použitá na to je metóda `is_active` triedy `ObserverDatabase` opísanej v sekcii 4.3. Pred samotnou definíciou rozloženia prvkov je touto metódou kontrolované, či práve prebieha sledovanie nejakého objektu nástroja *Grease Pencil*. Na základe toho sú následne definované prvky príslušného stavu. Výsledný vzhľad týchto dvoch stavov je zobrazený na obrázkoch 4.2.



(a) Neaktívny stav (b) Aktívny stav

Obr. 4.2: Implementované stavy grafického rozhrania pluginu

Zoznamy použité v rozhraní sú implementované v triedach `RECORDER_UL_layer_list` a `RECORDER_UL_change_list`. O ich vzhľad a správanie sa z veľkej časti stará aplikačné rozhranie Blenderu. Dôležitou poznámkou však je, že na vytvorenie položiek zoznam akceptuje len objekty vstavaných tried rozhrania, ktoré obaľujú základné dátové typy jazyka Python.

Pre toto použitie musia byť obalené do vstavaných typov aplikačného rozhrania Blenderu a registrované ako vlastnosti určitej triedy. Pre tento účel boli v súbore `lib/tracking.py`

implementované dve skupiny vlastností, do ktorých sú uložené zachytené zmeny. Implementované sú ako podtriedy triedy `PropertyGroup`, ktorá je vstavaný typ aplikačného rozhrania Blenderu a slúži na zastrešenie viacerých vlastností do jednej položky [5]. Samotné vlastnosti sú obalené vo vstavaných triedach, podľa ich dátového typu.

Funkciu záznamu pre konkrétnu vrstvu plní trieda `LayerChangesGroup`. Jej implementácia sa nachádza vo výpise 4.2. Trieda zastrešuje skupinu dát reprezentujúcu zmeny zachytené pre konkrétnu vrstvu. Atribút triedy `layer_name` obsahuje názov vrstvy ktorej záznam patrí. Okrem zobrazenia v grafickom rozhraní je použitý pre spárovanie záznamu a objektu vrstvy v ktorom bola zachytená zmena. Atribút `change_index` slúži na uloženie indexu tabuľky zachytených zmien vrstvy v grafickom rozhraní pluginu. Posledný atribút `changes` je zoznam záznamov zachytených zmien v danej vrstve.

Táto trieda vlastností je priradená pod triedu `GreasePencil` vo forme zoznamu pod názvom `layer_records` (viď výpis 4.4). Na indexovanie tohoto zoznamu v rámci tabuľky vrstiev je ešte priradená vlastnosť `layer_index`, rovnako pod triedu *GreasePencil*.

```
class LayerChangesGroup(PropertyGroup):
    layer_name: StringProperty()
    change_index: IntProperty(default=0)
    changes: CollectionProperty(type=ChangeGroup)
```

Výpis 4.2: Trieda reprezentujúca skupinu zachytených zmien pre konkrétnu vrstvu.

Jednotlivé záznamy zachytených zmien sú reprezentované triedou `ChangeGroup`. Ako vidno z výpisu 4.3, trieda obsahuje položku `obj`, ktorá mala byť použitá na odkazovanie na objekt na ktorom vznikla zmena. Táto možnosť kvôli obmedzeniam na aplikačnom rozhraní nebola vo finálnej verzii pluginu použitá. Zvyšné položky triedy reprezentujú text popisujúci charakter zachytenej zmeny a reťazec názvu ikony, ktorá má byť pri zobrazení záznamu v grafickom rozhraní použitá.

```
class ChangeGroup(PropertyGroup):
    obj: PointerProperty(type=Object)
    text: StringProperty()
    icon: StringProperty(default='')
```

Výpis 4.3: Trieda reprezentujúca jednu položku zachytenej zmeny.

Spomínané triedy vlastností musia byť pri načítaní pluginu registrované pre použitie v rámci API Blenderu pomocou vstavanej funkcie `bpy.utils.register_class`. Po registrácii je ich ďalej nutné priradiť pod určitú triedu, ktorá je podtriedou tried `ID`, `Bone` alebo `PoseBone`. Toto je riešené na úrovni podmodulu `lib`, v ktorom sú implementované, v rámci funkcie `register`. Na ilustráciu priradenia pod určitú triedu slúži výpis 4.4. Za zmienku stojí možnosť `HIDDEN`, ktorá signalizuje že daná vlastnosť sa nemá zobraziť v grafickom rozhraní aplikácie a teda je dostupná len z aplikačného rozhrania. Analogicky, pri vypnutí aplikácie alebo vysunutí modulu musia byť triedy odregistrované pomocou funkcie `bpy.utils.unregister_class`.

```
GreasePencil.layer_records = CollectionProperty(
    type=LayerChangesGroup,
    options={'HIDDEN'}
)
```

Výpis 4.4: Príklad priradenia implementovanej vlastnosti pod existujúcu vstavanú triedu

4.3 Spôsob sledovania zmien

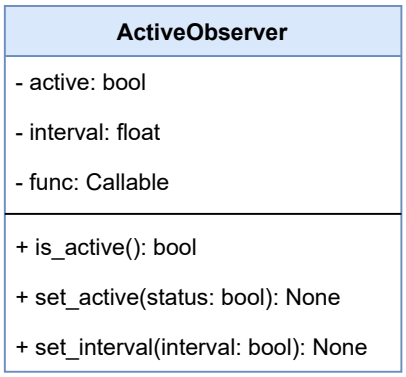
Sledovanie nastávajúcich zmien, preto je jeho implementácia rozdelená do viacerých tried v závislosti od sledovaných objektov. Ako bolo spomenuté v sekcii 3.3, sledovacej triede je pridelený objekt, na ktorom sleduje zmeny v objektoch nižšie v hierarchii. Zaznamenané zmeny sú potom reportované hlavnej triede, ktorá ich patrične obslúži.

Táto sekcia rozoberá jednotlivé triedy podieľajúce sa na procese sledovania, ako aj samotný spôsob sledovania zmien.

ActiveObserver

Pôvodným zámerom bolo využiť vstavané funkcie aplikačného rozhrania `bpy.msgbus` na sledovanie zmien v objektoch. Na základe experimentov s rozhraním ale vyplýva, že bohužiaľ v súčasnej verzii aplikácie[2] toto rozhranie dokáže sledovať zmeny len jednoduchých vlastností objektov. Znamená to, že vie zachytiť napríklad zmenu polohy objektu v rámci 3D scény ale nedokáže zachytiť zmenu počtu ťahov patriacich pod nástroj *Grease Pencil*. Správy od vývojárov a komunity Blenderu naznačujú, že v budúcnosti môže byť toto rozhranie rozšírené ale v súčasnom stave sa nehodí pre použitie v rámci tejto práce.

Z tohoto dôvodu bol zvolený spôsob sledovania zmien pomocou periodických kontrol sledovaných vlastností. Na tento účel je implementovaná rodičovská trieda `ActiveObserver`, z ktorej následne dedia triedy zodpovedné za sledovanie konkrétnych objektov. Diagram triedy je opísaný na obrázku 4.3. Keďže je tento spôsob sledovania výkonnostne náročný, je implementované len sledovanie vzniku a zániku vrstiev a ťahov na plátne. Zmena farby a tvaru už vytvoreného ťahu teda nie sú sledované.



Obr. 4.3: Diagram triedy `ActiveObserver`

Triedu `ActiveObserver` je nutné inicializovať vhodnou funkciou ktorej úlohou je sledovať konkrétne zmeny na objekte a informovať o nich príslušnú triedu, ktorá dedí z triedy

[2] verzia 3.0.0

`ActiveObserver`. Za pomoci rozhrania `bpy.app.timers` je táto funkcia periodicky volaná podľa intervalu stanoveného v triede `ActiveObserver`.

Na ilustráciu požiadaviek na spomínanú funkciu poslúži funkcia vo výpise 4.5 `observe_layers` implementovaná v súbore `lib/tracking.py`. Úlohou tejto konkrétnej funkcie je sledovanie počtu vrstiev pomocou porovnávania posledného počtu a súčasného počtu vrstiev. Použitá je v rámci triedy `GPenObserver`, ktorá bude prebraná nižšie. Sledovanie zmien ťahov je analogicky implementované vo svojej vlastnej funkcii.

```
def observe_layers(observer) -> float:
    new_count = observer.get_layer_count()

    if(observer.last_count < new_count):
        observer.on_add()
    elif(observer.last_count > new_count):
        observer.on_remove()
    observer.last_count = new_count

    return observer.interval
```

Výpis 4.5: Funkcia `observe_layers` zodpovedná za periodické sledovanie počtu vrstiev

Dôležité je, aby funkcia vracala interval, reprezentovaný dátovým typom `float` v sekundách, ktorý bude použitý rozhraním `bpy.apps.timers` na naplánovanie ďalšieho volania tejto funkcie. Keďže návratová hodnota funkcie je zabratá, na prístup a komunikáciu s objektom triedy `GPenObserver` slúži parameter `observer`[3]. Keďže funkcia nie je volaná z mnou vytvoreného kódu ale z plánovača v rámci Blenderu, nie je možné jej predať argumenty klasickým spôsobom. Tento problém je riešený použitím knižnice jazyka Python `functools`, ktorej trieda `partial` „zmrazí“ argumenty funkcie a spojí ich spolu s funkciou do jedného objektu [5]. Výpis 4.6 obsahuje výber z implementácie triedy `ActiveObserver`, ktorý rieši tento problém.

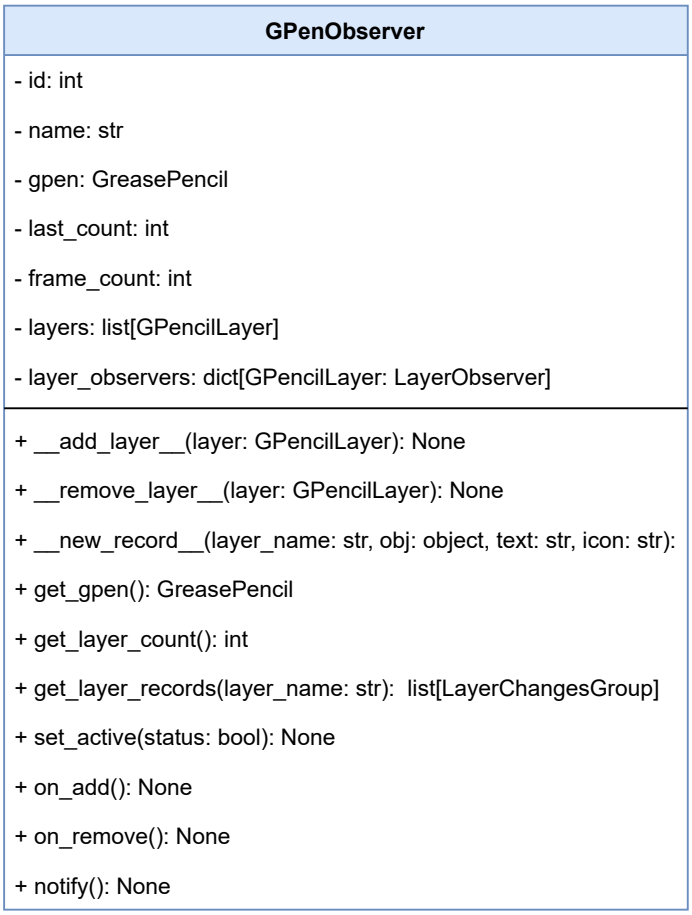
```
self.func = functools.partial(observing_func, self)

if status and (not registered):
    timers.register(self.func)
elif (not status) and registered:
    timers.unregister(self.func)
```

Výpis 4.6: Práca s periodickou funkciou pri použití rozhrania `bpy.apps.timers`

GPenObserver

Proces sledovania zmien na plátne ako celku zastrešuje trieda `GPenObserver`. Táto trieda dedí z vyššie spomínanej triedy `ActiveObserver`, čím je sprostredkované sledovanie vzniku a zániku vrstiev plátna. Okrem sledovania zmien je táto trieda ďalej zodpovedná za spracovanie zachytených dát, ktoré sa zobrazia v užívateľskom rozhraní pluginu a na základe ktorých vznikne výsledná animácia. Diagram triedy je naznačený na obrázku 4.4.

[3] parameter nie je rozšírený o type hinting z dôvodu prevencie cyklickej závislosti v module



Obr. 4.4: Diagram triedy `GPenObserver`

Triedu `GPenObserver` je nutné inicializovať objektom triedy *GreasePencil*, ktorý sa bude sledovať. Tento objekt je pri inicializácii uložený v rámci triedy *GPenObserver* a je z neho vybraný jeho názov a zoznam jeho vrstiev. Pre každú jeho vrstvu je potom vytvorený objekt triedy `LayerObserver`, čím sa ďalej propaguje proces sledovania na ďalšiu úroveň hierarchie objektov. Vrstvy a ich príslušné objekty triedy `LayerObserver` sú uložené v položke `layer_observers` v rámci triedy `GPenObserver`. Jedná sa o dátový typ slovník, kde kľúčom záznamu je objekt sledovanej vrstvy a hodnotou záznamu je sledovací objekt príslušnej vrstvy. Tým je zaistené správne párovanie spomínaných objektov.

Okrem vytvorenia objektov zodpovedných za sledovanie vrstvy je ďalej vytvorený záznam o danej vrstve pre použitie v grafickom rozhraní pluginu. Ako bolo spomenuté v sekcii 3.5, aplikačné rozhranie nepodporuje zobrazenie základných dátových typov vo svojom grafickom rozhraní. Pre tento účel boli implementované dve nové triedy, presnejšie opísané v sekcii 4.2.

Nové záznamy sa vytvárajú pomocou metódy `__new_record__` triedy `GPenObserver`. Jej parametre sú identické s položkami vyššie spomenutej triedy `ChangeGroup`. Pridaný je ešte parameter `layer_name` reprezentujúci názov triedy v ktorej sa zmena vyskytla. Slúži na identifikáciu záznamu patriacemu danej triede. Na základe týchto parametrov je v metóde vytvorený nový záznam zmeny pre danú vrstvu.

Okrem pridávania záznamov je metóda `__new_record__` zodpovedná aj za tvorbu nových snímkov výslednej animácie. Vznik a posun snímkov je v rámci aplikačného rozhrania riešený na úrovni vrstiev. V metóde sú preto všetky objekty vrstiev inštruované posunúť aktuálnu snímku prostredníctvom ich sledovacích objektov triedy `LayerObserver` a jej metódy `advance_frame`. Následne je inkrementovaný počet snímkov triedy a zároveň počítač snímkov triedy `GPenObserver` nachádzajúci sa v položke `frame_count`.

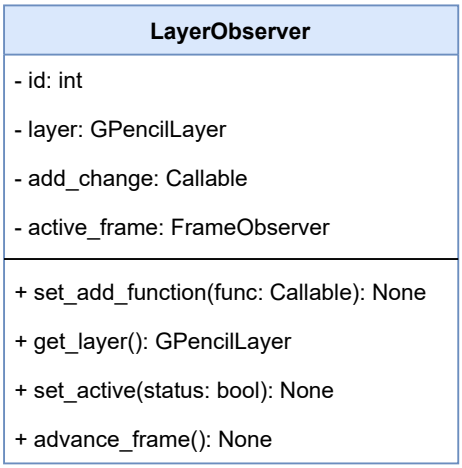
Samotný proces sledovania prebieha pomocou zdedeného rozhrania vyššie spomenutej triedy `ActiveObserver`. Použitá na to je sledovacia funkcia `observe_layers` taktiež už spomenutá vo výpise 4.5. Jej úlohou je periodicky kontrolovať počet vrstiev v sledovanom objekte triedy `GreasePencil`.

Pri vzniku novej vrstvy funkcia volá metódu triedy `GPenObserver` `on_add`, ktorá nájde novo vzniknutý objekt vrstvy a predá ho ďalej metóde `__add_layer__`. Táto metóda sa následne postará o vytvorenie nového záznamu a sledovacieho objektu triedy `LayerObserver`, ktorý bude novo vytvorený objekt vrstvy sledovať.

Pri zániku vrstvy funkcia volá metódu `on_remove`. Metóda upraví index tabuľky vrstiev a pomocou rozdielu medzi zoznamami aktuálnych a sledovaných vrstiev nájde objekt zaniknutej vrstvy, ktorý je následne predaný metóde `__remove_layer__`. Na základe spomínaného objektu je potom zrušený sledovací objekt zaniknutej vrstvy a zároveň je vymazaný záznam pre danú vrstvu.

LayerObserver

Na rozdiel od ostatných tried sa trieda `LayerObserver` aktívne nepodieľa na procese sledovania a preto nefiguruje ako podtrieda triedy `ActiveObserver`. Úlohou tejto triedy je réžia posunu snímkov pri zachytení novej zmeny v rámci jej priradenej vrstvy. Diagram tejto triedy je naznačený na obrázku 4.5.



Obr. 4.5: Diagram triedy `LayerObserver`

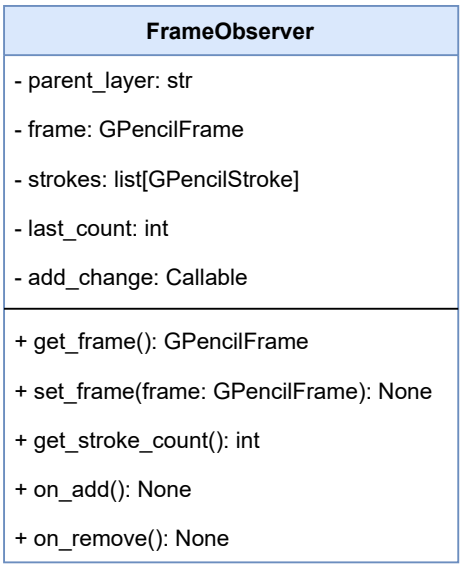
Trieda `LayerObserver` je inicializovaná pomocou objektu triedy `GPencilLayer` a funkcie slúžiacej na vytvorenie nového záznamu. Konkrétne sa jedná o metódu `__new_record__` triedy `GPenObserver`. Z prijatého objektu vrstvy je vybratá posledná snímka, ktorou je inicializovaný objekt triedy `FrameObserver`, spolu s názvom vrstvy a spomínanou funkciou na vytvorenie nového záznamu. Novo vzniknutý objekt je následne uložený v položke `active_frame` v rámci triedy `LayerObserver`.

Pri zachytení novej zmeny je z triedy `GPenObserver` volaná metóda `advance_frame`. V rámci tejto metódy je aktuálny snímok vrstvy skopírovaný a zaradený na koniec sekvencie snímkov. Následne je objekt triedy `FrameObserver` aktualizovaný novo vzniknutým objektom snímky, ktorý bude teraz sledovať.

Spomínané kopírovanie snímkov je príčinou chýbajúcej možnosti upravovať zachytené zmeny. Pri kopírovaní snímkov sa skopírujú aj všetky objekty ťahov. Ak by užívať zadal zmenu v strede sekvencie bolo by nutné aktualizovať všetky nasledujúce snímky. Musel by byť presne identifikovaný objekt ťahu, ktorý bol takto dodatočne zmenený, a to pre každú nasledujúcu snímku. Takáto operácia by bola výkonnostne aj implementačne náročná.

FrameObserver

Úlohou triedy `FrameObserver` je sledovanie vzniku a zániku ťahov na plátne, v rámci aktuálnej snímky animácie. Pri zachytení novej zmeny je vytvorený nový záznam popisujúci charakter zmeny. Záznam je následne zaradený do zoznamu príslušnej vrstvy, pod ktorú sledovaná snímka patrí. Diagram triedy je dostupný na obrázku 4.6.



Obr. 4.6: Diagram triedy `FrameObserver`

Na inicializáciu triedy je potrebný objekt triedy `GPencilFrame`, reprezentujúci počiatočnú snímku, na ktorej budú sledované zmeny. Ďalej je potrebný názov vrstvy, ktorej snímky budú sledované a funkcia pomocou ktorej sa vytvorí nový záznam o zachytenej zmene.

Keďže táto trieda je podtriedou triedy `ActiveObserver`, spomenutej v sekcii 4.3, zmeny sa sledujú za pomoci periodického volania funkcie. Použitá na to je funkcia `observe_strokes` implementovaná v súbore `lib/tracking.py`. Ako vyplýva z výpisu 4.7, funkcia funguje na princípe porovnávania aktuálneho počtu ťahov s naposledy zaznamenaným počtom ťahom.

Pri vzniku nového ťahu je volaná metóda `on_add`, ktorá vytvorí nový záznam s textom „Stroke added.“ a s ikonou „plus“. Pri zániku je volaná metóda `on_remove`, ktorá rovnako vytvorí nový záznam s textom „Stroke removed.“ a ikonou „X“.

Vytvorenie nového snímku animácie, ako reakcie na novú zmenu, je obslúžené metódou `set_frame`, ktorá nastaví v parametri obdržaný objekt novej snímky pre ďalšie sledovanie.

Metóda je volaná na úrovni vrstvy v rámci triedy `LayerObserver`. Použitím tohoto spôsobu aktualizácie teda nieje potrebné inštanciovať ďalší objekt triedy `FrameObserver` pre novo vzniknutú snímku.

```
def observe_strokes(observer) -> float:
    new_count = observer.get_stroke_count()

    if(observer.last_count < new_count):
        observer.on_add()
    elif(observer.last_count > new_count):
        observer.on_remove()
    observer.last_count = new_count

    return observer.interval
```

Výpis 4.7: Funkcia `observe_strokes` používaná na periodické sledovanie počtu ťahov[4]

ObserverDatabase

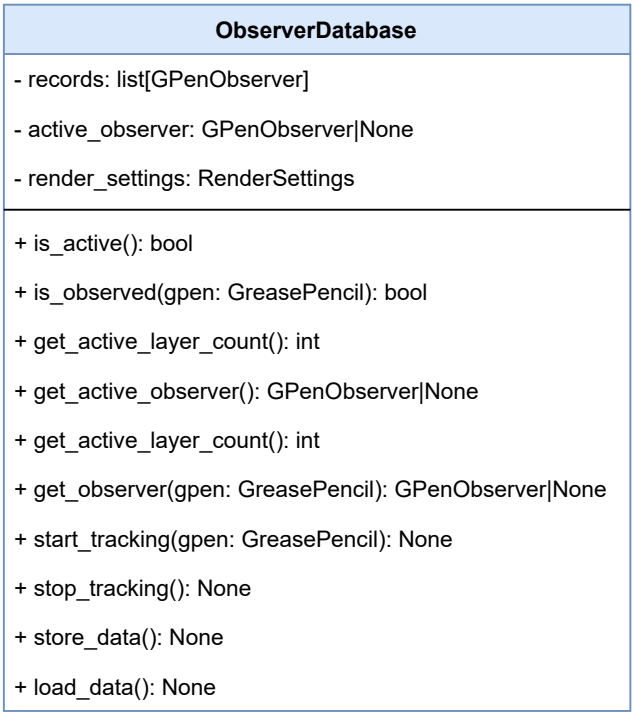
Napriek tomu, že trieda `ObserverDatabase` sa priamo nepodieľa na procese sledovania je dôležitá z hľadiska réžie sledovania a práce s dátami. Použitie tejto triedy je navrhnuté na základe návrhového vzore Jedináčik (z anglického Singleton), čo znamená, že v rámci kódu pluginu existuje len jedna inštancia tejto triedy. Tento prístup bol zvolený z dôvodu potreby mať jednotné miesto prístupu k riadeniu procesu sledovania a príslušným dátam. Celé rozhranie tejto triedy je zobrazené na obrázku 4.7.

Väčšina metód triedy je určených na réžiu sledovania. Je nimi riešené spustenie sledovanie pomocou inštanciácie triedy `GPenObserver` a rovnako aj ukončenie sledovania. Ďalej je nimi sprostredkované zisťovanie statusu sledovania, teda, či v súčasnej dobe prebieha nejaké sledovanie a či je konkrétny objekt triedy `GreasePencil` už sledovaný pomocou triedy `GPenObserver`.

Okrem réžie sledovanie táto trieda ďalej slúži na uchovávanie nastavení spojených s renderingom výslednej animácie. Uložené sú v položke `render_settings`, ktorá je inštanciou triedy `RenderSettings`.

V poslednom rade je trieda `ObserverDatabase` zodpovedná za perzistentné ukladanie nazbieraných dát sledovania. Slúžia na to metódy `store_data` a `load_data`. V súčasnej verzii pluginu však tieto metódy nie sú implementované. Odôvodnenie a možné budúce riešenie sú rozobraté v sekcii 4.5.

[4] parameter nie je rozšírený o type hinting z dôvodu prevencie cyklickej závislosti v module



Obr. 4.7: Diagram triedy `ObserverDatabase`

4.4 Animácia

Ako bolo spomenuté v predchádzajúcej sekcii 4.3, pri každom zachytenom vzniku a zániku ťahu na plátne je vytvorený nový snímok v rámci sledovaného objektu triedy *GreasePencil*.

Za konverziu snímkov na výslednú animáciu zodpovedá trieda `RECORDER_OT_render` implementovaná v súbore `operators/io.py`. Táto trieda zároveň implementuje tlačidlo *Render* v rámci grafického rozhrania pluginu. Použitie tlačidla je možné len v prípade, že v súčasnosti prebieha sledovanie. Na to je použitá metóda `is_active` databázy sledovania (spomenutá v sekcii 4.3). Po stlačení tlačidla sa začína proces renderingu.

Na začiatku prebehne kontrola, či zvolené nastavenia sú v poriadku a teda je na ich základe možné úspešne vykonať rendering. Kontroluje sa tiež, či vôbec existuje aspoň jedna snímka, z ktorej má vzniknúť výsledná animácia. Dôvod prečo tieto kontroly nie sú súčasťou kontrol blokujúcich použitie tlačidla *Render* je, že v takom prípade by nebolo možné užívateľa informovať o príčine chyby. Takýto prístup je prirodzene žiaduci, pretože užívateľ je takto presne informovaný a môže tak vzniknutú chybu jednoduchý napraviť.

Po úspešnom vyhodnotení spomenutých kontrol sa začína skutočný proces renderingu. V prvej etape sú jednotlivé snímky vyrenderované do priečinku `render`, ktorý sa nachádza v užívateľom vybranom priečinku. Snímky sú renderované pomocou vstavanej funkcie `bpy.ops.render.render` a podľa voľby užívateľa je na to použitá kamera scény. Výsledné obrázky sú vo formáte *PNG*.

Druhá etapa renderingu spojí vzniknuté obrázky do výslednej animácie. Jej formát je *AVI* a jej názov je spojením reťazca `animation` a reťazca reprezentujúceho rozsah použitých obrázkov. Teda pri renderingu 42 pôvodných snímkov je vysledný názov animácie `animation0001-0042.avi`. Použitie rozsahu snímkov v názve animácie je vstavané do rozhrania Blenderu a dá sa riešiť premenovaním súboru. To je už však ponechané

na voľbu užívateľa. Predvolený počet snímkov za sekundu je 24, prípadne iná, užívateľom zadaná hodnota. Na tvorbu výslednej animácie je tiež použitá vstavaná funkcia `bpy.ops.render.render`.

Na rekapituláciu, po úspešnom vykonaní procesu renderingu sa v užívateľom zadanom priečinku nachádza výsledná animácia vo formáte *AVI* a priečinok `render` obsahujúci obrázky použité v animácii.

4.5 Ukladanie dát

V súčasnej verzii nie je implementované perzistentné ukladanie dát a teda modul si neuchováva nazbierané dáta po vypnutí aplikácie editoru. Keďže sa ale jedná o dôležitú súčasť funkčnosti aplikácie bude túto funkciu nutné implementovať. Pravdepodobný spôsob ukladania bude pomocou textových súborov vo formáte XML alebo JSON. Tieto formáty sú vhodné z hľadiska možnosti definovať presnú štruktúru uložených dát a zároveň možnosť s ňou jednoducho pracovať v rámci kódu modulu. Z týchto dvoch formátov sa perspektívnejší javí formát JSON a to z dôvodu že je ho v jazyku Python jednoduché konvertovať do vstavaného dátového typu slovník, čo by uľahčilo réžiu práce pri ukladaní dát a tiež znížilo množstvo kódu na to potrebného.

Názov a lokáciu súboru s uloženými dátami by bolo vhodné odvodiť od súboru `*.blend` obsahujúceho dáta práce v rámci editoru a na ktorý je takto naviazané prebiehajúce sledovanie.

Jednou z možných komplikácií ukladania sa môže stať kompatibilita medzi rôznymi verziami aplikácie Blender. Je potrebné brať do úvahy, že ak by medzi verziami nastala významná zmena rozhrania bude tiež nutné prispôsobiť spôsob a formát ukladania. V praxi sa teda bude mimo samotných dát musieť ukladať aj verzia aplikácie v ktorej boli zachytené, preskúmať kompatibilitu medzi verziami a pri zistení nezhôd ideálne implementovať riešenia problémov alebo informovať užívateľa o možných problémoch.

Kapitola 5

Záver

Cieľom tejto práce bolo preskúmať možnosť sledovať ťahy na plátne grafického editoru, zaznamenať ich a následne ponúknuť možnosť ich prehrať a upravovať. Táto funkcionalita mala byť následne implementovaná formou zásuvného modulu pre vhodný grafický editor.

Prvým krokom k dosiahnutiu stanovených cieľov bolo preskúmať grafické editory a vybrať z nich jeden, ktorý poskytoval najlepšie podmienky pre ich implementáciu. Kandidáti na prieskum boli grafické editory MyPaint, Krita a Blender. Vybrané z pomedzi iných editorov boli na základe nasledovných kritérií: ich popularita v umeleckej komunite, neplatený prístup k aplikácii a ich rozšíriteľnosti pomocou zásuvných modulov. Proces prieskumu začínal študovaním dokumentácie a iných dostupných zdrojov. Cieľom bolo zistiť, či editor poskytuje dostatočné prostriedky na vývoj zásuvných modulov, má zo svojho aplikačného rozhrania prístup k objektom reprezentujúcich grafické prvky, má vstavanú možnosť sledovať zmeny na spomínaných objektoch a v poslednom rade či je proces vývoja zásuvných modulov a na to použité aplikačné rozhranie dobre dokumentované. Následne, po vyhodnotení dostačujúcich výsledkov tohoto formálneho prieskumu, sa prešlo k experimentovaniu s aplikačným rozhraním formou implementácie minimálnych zásuvných modulov a s cieľom overiť fungovanie a možnosti sledovania zmien na objektoch grafických prvkov.

Na základe výsledkov prieskumu bol vybraný editor Blender. V porovnaní s ostatnými editormi má dobre navrhnuté aplikačné rozhranie, ktoré spĺňa vyššie spomenuté podmienky. Zároveň je kvalitne opísané vo svojej dokumentácii, ktorá obsahuje presné návody na vývoj zásuvných modulov a detailné popisy celého rozhrania. Pridanou hodnotou bola tiež vstavaná konzola v jazyku Python s prístupom do aplikačného rozhrania za behu aplikácie a tiež široká komunitná podpora pre vývoj zásuvných modulov.

Značná časť textu tejto práce sa venuje vysvetleniam základných pojmov použitých v texte s cieľom uviesť čitateľa do kontextu problematiky spojenej s prostredím grafických editorov a vývoja zásuvných modulov. Ďalej je priblížená pre prácu dôležitá časť prostredia editoru Blender, relevantné triedy grafických prvkov a v poslednom rade formálne požiadavky na zásuvné moduly pre tento editor.

Zvyšok textu sa následne sústredí na návrh a implementačné detaily vývoja výsledného zásuvného modulu.

Výsledkom tejto práce je plne funkčný zásuvný modul s vlastným grafickým rozhraním použiteľný v rámci editoru Blender. Na základe formálnych požiadavkov je implementovaný formou modulu jazyka Python za dodržiavania štýlovej konvencie PEP 8.

Zo zadanej funkcionality sa podarilo splniť zaznamenávanie vznikajúcich a zanikajúcich ťahov a ich následné prehranie formou animácie, v ktorej je každá zaznamenaná zmena reprezentovaná v jednom snímku snímku animácie. Dodatočná funkcionalita je možnosť

pozastaviť záznam a tak vytvoriť skokovú animáciu, kedy sa jednotlivé zmeny na plátne prejavia ako skupina vo výslednej animácii. Modul ďalej podporuje zastavenie sledovania a zmenu sledovaného objektu bez straty zaznamenaných dát.

Modul je vhodný pre ďalšie rozšírenia hlavne z dôvodu, že sa nepodarilo implementovať úpravu zachytených zmien a to z dôvodu komplikovaného riešenia tejto problematiky. Ďalej chýba perzistentné ukladanie nazbieraných dát hlavne z dôvodu časovej náročnosti implementácie a potreby riešiť kompatibilitu medzi jednotlivými verziami editoru. V poslednom rade nebolo možné použiť vstavaný spôsob sledovania zmien, pretože v súčasnej verzii sleduje len zmeny jednoduchých vlastností objektov. Ak bude v budúcnosti rozšírené sledovanie, bolo vhodné nahradiť ním súčasné riešenie, ktoré je v porovnaní so vstavaným spôsobom sledovania viac výkonnostne náročné.

Túto prácu som si vybral pretože mám skúsenosti s tvorbou počítačovej grafiky zo Základnej umeleckej školy a zaujala ma možnosť vyvinúť konkrétne rozšírenie pre grafický editor. Pri práci som sa zoznámil s vnútorným fungovaním editorov a vyskúšal si proces vývoja zásuvných modulov. Zároveň som si zlepšil svoje zručnosti v jazyku Python, čo beriem ako prínos pre svoju budúcu profesionálnu kariéru.

Literatúra

[1] Blain, J. M. *Blender 2D Animation: The Complete Guide to the Grease Pencil*. 1. vyd. Taylor & Francis Ltd, 2021. ISBN 1032110325.

[2] Bungartz, H.-J., Griebel, M. a Zenger, W. *Introduction to Computer Graphics*. 2. vyd. Charles River Media, 2004. ISBN 9781584503323.

[3] Flavell, L. *Beginning Blender: Open Source 3D Modeling, Animation, and Game Design*. 1. vyd. Apress, 2010. ISBN 9781430231271.

[4] Foundation, B. *Developer Documentation* [online]. [cit. 2022-01-25]. Dostupné z: https://wiki.blender.org/wiki/Main_Page.

[5] Foundation, B. *Python API Documentation for Blender* [online]. [cit. 2022-01-18]. Dostupné z: https://docs.blender.org/api/current/index.html.

[6] Leung, J. a Lara, D. M. Grease Pencil: Integrating Animated Freehand Drawings into 3D Production Environments. Association for Computing Machinery. 2015. SA '15. DOI: 10.1145/2820903.2820924. Dostupné z: https://doi.org/10.1145/2820903.2820924.

[7] Petrovic, S. *Krita Manual* [online]. [cit. 2022-05-13]. Dostupné z: https://docs.krita.org/en/.